

100071011 Computer Networks 2022–2023–2

Project–1

Reliable File Transfer using Go–Back–N protocol  
Specification

学号 (Student ID)	1120202944
姓名 (Name)	董若扬
班号 (Class No.)	07112005
授课教师 (Instructor)	胡琳梅

School of Computer

Beijing Institute of Technology

April 12, 2023

## 1. Requirement Analysis

This project aims to implement a reliable file transfer protocol using the Go-Back-N (GBN) protocol. The protocol needs to ensure the reliable transmission of data even in the presence of errors or lost packets. The project requires establishing a connection between the sender and receiver, segmenting the file into appropriate segments, transmitting segments using the GBN protocol, and reassembling the file at the receiver. The project also needs to handle error messages and timeout events to ensure the reliability and integrity of data transmission.

GBN is a reliable data transmission protocol commonly used for transmitting data over unreliable channels. The GBN protocol achieves reliability by using a sliding window mechanism. The sliding window mechanism allows the sender to send multiple packets before receiving acknowledgments, and the receiver can cache these packets and deliver them in the correct order to the upper-layer protocol. If the receiver detects errors, it sends an acknowledgment message to the sender indicating the need to retransmit the lost packets. The GBN protocol can be used for reliable file transfer because it can detect and recover from lost and erroneous packets.

Main issue and key function:

- Establishing and maintaining connection for data transmission
- Segmenting the file into appropriate chunks for transmission
- Implementing the GBN protocol to detect and retransmit lost packets
- Reassembling the file at the receiver
- Handling error messages and timeout events to ensure the reliability and integrity of data transmission.

## 2. Design

This project involves the following tasks:

- 1) Frame structure: Design and implement the structure of data frames, including fields such as frame header, frame body, checksum, etc., to facilitate the grouping and assembly of data packets during data transmission.

```
class UDT:
    def __init__(self, lost, err):
        random.seed(time.time())
        self.LOST_PROB = lost
        self.ERR_PROB = err
```

- 2) Checksum generation and verification: Design and implement the mechanism for generating and verifying checksums to ensure the integrity and accuracy of data during transmission.

```
def calculate_crc(data, crc=0):
    if not data:
        raise ValueError("The input data cannot be empty.")
    for byte in data:
        if not isinstance(byte, int):
            raise TypeError("The input data can only contain integers.")
        if byte < 0 or byte > 255:
            raise ValueError(
                "The integers in the input data must be within the range [0,
255].")
        crc = ((crc << 8) & 0xff00) ^ CRC_TABLE[((crc >> 8) & 0xff) ^ byte]
    return crc & 0xffff
```

- 3) Sliding window setting and control: Design and implement the sliding window mechanism, including window size, sender and receiver window control, etc., to facilitate the transmission and reception of data packets.

```
while next_frame_to_send < ack_expected + window_size:
    if next_frame_to_send >= len(packets):
        break
    if overtime_flag == 0:
        log_file.write(
            f"{time.ctime()}: Send
PDU={next_frame_to_send}, STATUS=New, ACKed={ack_expected} to
{receiver_addr}\n")
```

- 4) Cache: Design and implement the cache mechanism, including cache size, cache data structure, etc., to facilitate the storage and management of data packets.

```
while True:
    data = file.read(512)
    if not data:
        break
    crc_num = crc.calculate_crc(data)
    pdu = packet.packing(seq_num, crc_num, data)
    packets.append(pdu)
    seq_num += 1
```

- 5) Sequence space and sequence number setting and control: Design and implement the mechanism for setting and controlling sequence space and sequence number to ensure the order and integrity of data packets.

- 6) Acknowledgement rules: Design and implement the acknowledgement rules, including sender and receiver acknowledgement mechanism, acknowledgement timeout, etc., to ensure reliable transmission of data packets.
- 7) Retransmission: Design and implement the retransmission mechanism, including sender and receiver retransmission control, handling of lost data packets, etc., to ensure reliable transmission of data packets.
- 8) Timeout timer maintenance: Design and implement the mechanism for maintaining the timeout timer to ensure timely transmission and retransmission of data packets.

```

while ack_expected < len(packets):
    mutex.acquire()
    while next_frame_to_send < ack_expected + window_size:
        if next_frame_to_send >= len(packets):
            break
        if overtime_flag == 0:
            log_file.write(
                f"{time.ctime()}: Send
PDU={next_frame_to_send},STATUS=New,ACKed={ack_expected} to
{receiver_addr}\n")
            elif overtime_flag == 1:
                log_file.write(
                    f"{time.ctime()}: Send
PDU={next_frame_to_send},STATUS=T0,ACKed={ack_expected} to
{receiver_addr}\n")
                send_timer.start(next_frame_to_send)
                UDTER.send(packets[next_frame_to_send], sock, receiver_addr)
                next_frame_to_send += 1
            overtime_flag = 0
        if send_timer.overtime(ack_expected):
            overtime_flag = 1
            next_frame_to_send = ack_expected

```

- 9) UDP socket: Design and implement the UDP socket to facilitate the transmission and reception of data packets.
- 10) System model: Design and implement the system model, including sender and receiver data flow, control flow, etc., to facilitate reliable transmission of data packets.
- 11) Function or class definition: Design and implement the definition of functions or classes to implement the processing, storage, and management of data packets.

- 12) Flowchart, state diagram, and timing diagram: Design and implement the flowchart, state diagram, and timing diagram to describe and analyze the transmission and processing flow of data packets.
- 13) Various possible errors and simulations: Design and implement various possible errors and simulations to test the reliability and robustness of the transmission and processing mechanism of data packets.
- 14) Logging: Design and implement the logging mechanism to record the transmission and processing process of data packets for troubleshooting and problem analysis.
- 15) Configuration file and parameters: Design and implement the configuration file and parameters, including UDP port, sequence number bit size, timeout timer value, etc., to facilitate system initialization and configuration.

```
[send]
ip = 10.194.156.11
port = 2020
filename = Computer network.txt
[receive]
ip = 10.194.156.11
port = 2944
filename = 1.txt
[protocol]
window_size = 511
data_length = 1024
initial_seq_num = 0
timeout = 5
Error_rate = 1
Lost_rate = 1
```

- 16) Multiprocess/multithread: Design and implement the multiprocess/multithread mechanism to achieve concurrent processing and transmission of data packets.
- 17) Queue: Design and implement the queue mechanism to facilitate the storage and management of data packets.
- 18) Reading configuration files for system initialization: Design and implement the mechanism for reading configuration files for system initialization to facilitate user configuration and use of the system.

### 3. Development and Implementation

OS:

Version: Windows 11 Pro

Build: 22H2

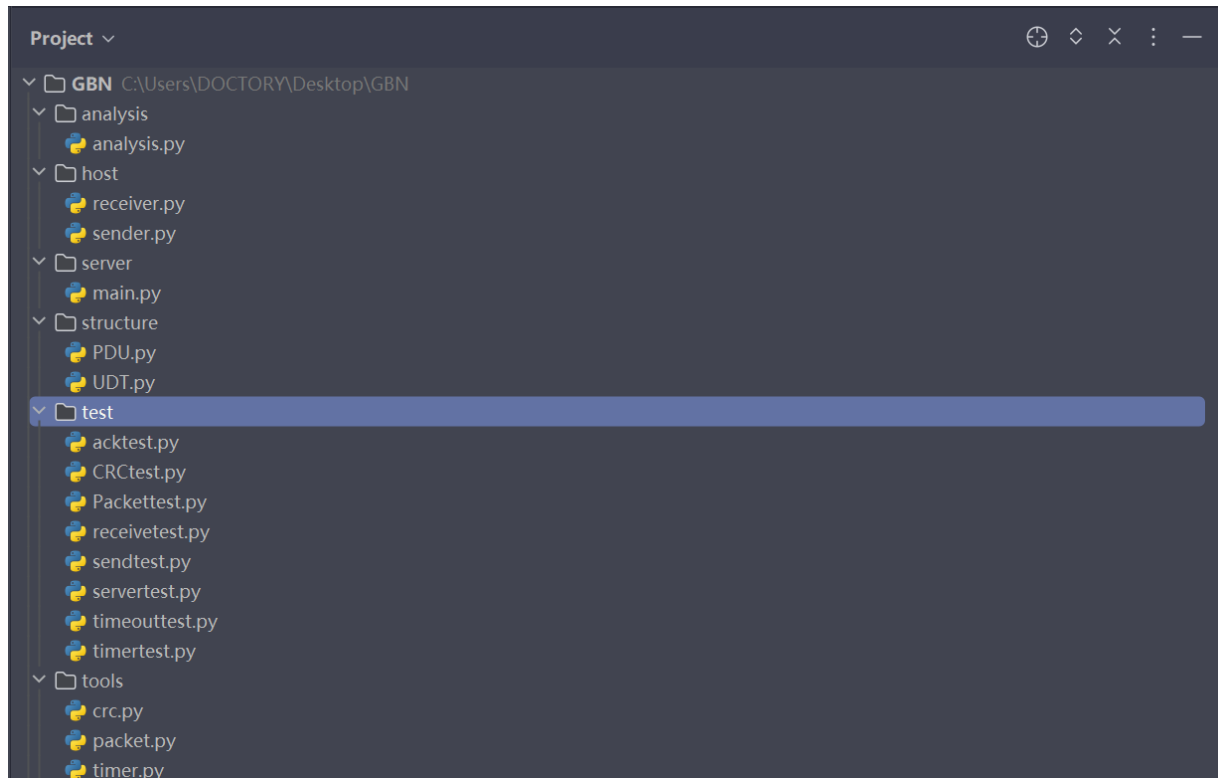
Installation Date: March 22, 2023

OS Version: 22621.1616

Experience: Windows Feature Experience Pack 1000.22641.1000.0  
Language: Python

Tools: Visual Studio Code, Pycharm Professional

project structure:



–PDU.py: Implements an unreliable data transfer protocol. The constructor of this class requires the loss rate and error rate as input parameters, which are used to simulate random packet loss and errors in network transmission.

- seq\_num: Sequence number, packet identifier.
- data: Represents the data, the content of the transmitted message.
- crc: Represents the CRC checksum of the data, used to detect errors during data transmission.
- start\_time: Represents the start time of data transmission, used to record the transmission delay of the packet.
- init(self, seq\_num, data=b''): Constructor, requires the sequence number and data as input parameters, and automatically calculates the CRC checksum of the data.

- `str(self)`: Overrides the `__str__` method, returns the string representation of the current object for easy printing of debugging information.

– `UDT.py`: This class implements an unreliable data transfer protocol, used to simulate random packet loss and errors in network transmission. The constructor of this class requires the loss rate and error rate as input parameters, which are used to simulate random packet loss and errors in network transmission.

- `LOST_PROB`: Represents the loss rate.
- `ERR_PROB`: Represents the error rate.
- `init(self, lost, err)`: Constructor, initializes UDT with loss rate `lost` and error rate `err`.

– `send(self, packet, sock, addr)`: Sends the packet. If the random number is less than the error rate, an error is added to the packet. If the random number is greater than the loss rate, the packet is sent.

- `recv(sock)`: Receives the packet.
- `sendack(self, ack, sock, addr)`: Converts the ack to a byte stream and sends it.
- `recvack(sock)`: Receives the ack.
- `make_error(packet)`: Adds an error to the packet to simulate errors in network transmission.

– `crc.py`:

– `CRC_TABLE`: A 16-bit CRC (cyclic redundancy check) lookup table used to calculate the CRC checksum value of a given byte string.

– `calculate_crc(data, crc=0)`: Calculates the 16-bit CRC checksum value of the given byte string. If the input data is empty, it raises a `ValueError`. If the input data contains any non-integer elements, it raises a `TypeError`. If the input data contains integers outside the range `[0, 255]`, it raises a `ValueError`. This method updates the CRC value using the XMODEM algorithm and the `CRC16_XMODEM_TABLE` lookup table, and returns the final 16-bit CRC checksum value.

– `packet.py`

– `packing(seq_num, crc_num, data=b'')`: Packs the sequence number, CRC number, and data into a byte string, converting the sequence number and CRC to bytes using little-endian byte order. If the provided data is not a byte string, a `TypeError` exception will be raised. The function returns the packed data.

– `pack_empty()`: Returns an empty byte string. If an exception occurs during function execution, the function logs the exception and returns an empty byte string.

–unpacking(packet): Extracts the sequence number, CRC number, and data from a byte string and returns them as a tuple. If the provided byte string is less than 8 bytes, the function returns None. The function extracts the sequence number from the first 4 bytes using little–endian byte order and the CRC from the next 4 bytes. The remaining bytes are treated as data.

–timer.py

–\_\_init\_\_(self, \_interval): Constructor that creates a timer object, where the \_interval parameter represents the interval time of the timer.

–start(self, seq): Starts the timer, where the seq parameter represents the sequence number.

–get\_time(): Static method that gets the current time.

–overtime(self, seq): Checks if the timer for the given sequence number has timed out. Returns True if the timer has timed out, False otherwise. If the provided sequence number is greater than the maximum sequence number in the timer, it will be decremented.

–sender.py

–configparser: Used for parsing configuration files.

–\_thread: Used for creating locks.

–threading: Used for creating multiple threads.

–time: Used for time calculations.

–UDT: Custom transport layer protocol.

–crc: Used for calculating the checksum of data.

–packet: Custom data packet format.

–timer: Custom timer class.

–ack\_expected: Expected frame number.

–num\_packets: Sent frame number.

–send\_timer: Timer object.

–log\_filename: Log file name.

–mutex: Mutex lock.

–UDTER: Custom transport protocol object.

–Send: Used to send file data. This function first reads the file and packs the data into data packets (using a custom data packet format), which are then sent out. The function uses a sliding window protocol to send data to ensure reliable transmission. The function also uses a timer to detect if a data packet has timed out for timely retransmission.



–Receive: Used for receiving ACKs. This function receives ACKs in a loop and updates the expected frame number if the received ACK is greater than or equal to the expected frame number. When the expected frame number is greater than or equal to the sent frame number, it means that all frames have been successfully sent, and the function exits.

–receiver.py

–receive(sock, filename, ip\_port): Receive data from the specified socket `sock` and write it to a file `filename`. `ip\_port` is a tuple containing the IP address and port number of the receiver.

To implement this function, the following steps can be taken:

1. Create a UDT object.

```
config = configparser.ConfigParser()
config.read('config.ini')
lost = config.getfloat('protocol', 'Error_rate') / 100.0
err = config.getfloat('protocol', 'Lost_rate') / 100.0
udter = UDT.UDT(lost, err)
```

2. Open a file to write the received data into.

```
file = open(filename, "wb")
log_filename = f"{ip_port[0]}_{ip_port[1]}_log_file.txt"
log_file = open(log_filename, "a+")
```

3. Create a log file to record the receiving process.

```
log_message = "{time}: Receiving file: {filename}\n".format(time=time.ctime(), filename=filename)
log_file.write(log_message)
```

4. Initialize the expected frame number to 0.
5. Write into the log file, recording the name of the file to be received.
6. Use the `recv()` method of the UDT object to receive data until the end flag is received.

```
pdu, addr = udter.recv(sock)
```

7. For each received packet, extract the sequence number, checksum, and data, and verify the checksum of the data is correct.

```
seq_num, crc_num, data = packet.unpacking(pdu)
crc_expected = crc.calculate_crc(data)
```

8. If the sequence number of the data is the same as the expected sequence number, record the data as correct, send an ACK confirmation, and write the data into the file. Then increment the expected frame sequence number.

```
if seq_num == frame_expected:
    log_message = "{time}: Receive PDU={seq_num}, STATUS=OK,
FRAME_EXPECTED={frame_expected} from {addr}\n".format(
        time=time.ctime(), seq_num=seq_num,
frame_expected=frame_expected, addr=str(addr))
    log_file.write(log_message)
    udter.sendack(frame_expected, sock, addr)
    frame_expected += 1
    file.write(data)
```

9. If the sequence number of the data is different from the expected sequence number, send the previous ACK confirmation.

```
else:
    log_message = "{time}: Receive PDU={seq_num}, STATUS=NoErr,
FRAME_EXPECTED={frame_expected} from {addr}\n".format(
        time=time.ctime(), seq_num=seq_num,
frame_expected=frame_expected, addr=str(addr))
    log_file.write(log_message)
    udter.sendack(frame_expected - 1, sock, addr)
```

10. Write into the log file, recording the situation of the received data packet.
11. After the receiving process is completed, write into the log file to indicate the completion of the program execution.
12. Close the log file and data file.
13. Output the prompt.

–main(): This function implements the logical control of the program. It reads the parameters from the configuration file, and performs sending, receiving, or exiting operations based on the user's input. When sending or receiving files, it uses multithreading to implement parallel processing.

–analysis.py

–`read_log_file(file_path)`: This function reads the contents of a file specified by the given file path and returns a list of strings containing the file content.

–`analyze_log(log_lines)`: This function analyzes the log file, counts the number of occurrences of "STATUS=OK", "STATUS=NoErr", and "STATUS=DataErr", and returns these three pieces of data.

–`plot_data(new_count, timeout_count, retransmit)`: This function uses matplotlib and seaborn to draw a pie chart and adds a title and configuration information to the chart.

Critical function:

`Receiver.receive()`:

This function is a receiving function used to receive and write data transmitted via UDP protocol to a file. The function's parameters include a socket object, a filename, and a tuple of an IP address and port number.

The function first creates a UDT object "udter" and opens a file and a log file to record the transmission status of the data. Then it initializes the expected frame number to 0 and writes a log message recording the file being received. Next, it uses the UDT object's `recv()` method to receive data, and if no data is received, the loop ends.

When data is received, the function uses the `packet.unpacking()` function to extract the packet's sequence number, checksum, and data, and calculates the checksum of the data. If the checksum is not equal to the received checksum, the function records a log message indicating that the data is incorrect and skips that packet.

If the sequence number of the data is equal to the expected sequence number, the function records a log message indicating that the data is correct, sends an ACK to confirm the receipt of that packet, increases the expected sequence number by 1, and then writes the data to the file.

If the sequence number of the data is not equal to the expected sequence number, the function sends an ACK to confirm the receipt of the previous packet and records a log message indicating that the data arrived out of order.

Finally, the function writes a log message indicating that the program execution is complete, closes the log file and file, and prints a message indicating that the program execution is complete.

The UDT object in the function is a custom class that encapsulates the UDP protocol and provides reliable data transmission. The packet module provides functions for packing and unpacking data packets, and the crc module provides functions for calculating checksums.

Send():

implements file transfer based on reliable transmission protocol, including fragmentation, checksum, confirmation frame, retransmission, progress bar, log recording, etc.

Use the configparser library to read parameters from the config.ini configuration file, including window size.

Open the corresponding file, record the log, and split the file into a series of data packets with a size of 512 bytes, and calculate the CRC checksum of each data packet. Pack these data packets and store them in a list.

```
packets = []
seq_num = 0
while True:
    data = file.read(512)
    if not data:
        break
    crc_num = crc.calculate_crc(data)
    pdu = packet.packing(seq_num, crc_num, data)
    packets.append(pdu)
    seq_num += 1
```

Create a new thread to receive data packets.

```
thread = threading.Thread(target=receive, args=(sock,))
thread.start()
```

Initialize some parameters, including the sending window size, timer, expected confirmation frame sequence number, number of data packets, etc.

In a loop, send all data packets in the sending window until all data packets are sent and confirmed.

If a confirmation is received, update the confirmation frame sequence number and continue sending data packets in the sending window.

If no confirmation is received, retransmit the earliest unconfirmed data packet.

While sending data packets, log information is recorded and the transmission progress bar is output to the console.

After all data packets have been sent, an empty data packet is sent to the receiving party to indicate the end of the transmission. Close the file and log recording.

```
UDTER.send(packet.pack_empty(), sock, receiver_addr)
log_file.write("File sent successfully.\n")
```

send.receive():

Implements the reception and processing of acknowledgement frames during the transmission process, and updates the expected sequence number of the acknowledgement frame. This function is usually called in another thread of the send function to implement data transmission based on a reliable transport protocol.

In a loop, it continuously receives UDP packets and extracts the acknowledgement frame sequence number from them.

If the received acknowledgement frame sequence number is greater than or equal to the expected sequence number, the expected sequence number of the acknowledgement frame is updated.

To avoid multiple threads modifying shared variables simultaneously, a mutex lock is used to ensure that the modification of the expected sequence number of the acknowledgement frame is thread-safe.

If the received acknowledgement frame sequence number equals the total number of data packets in the file, the loop is exited and the reception process ends.

```
while True:
    ack, _ = UDTER.recvack(sock)

    if ack >= ack_expected:
        mutex.acquire()
        ack_expected = ack + 1
        mutex.release()
    if ack_expected >= num_packets:
        break
```

Server.main(): Implements a simple network file transfer program that continuously reads user input to implement file transfer sending and receiving functions.

Reads the configuration file config.ini to obtain the IP address, port number, file name, and other parameters of the sender/receiver.

Provides three options: send, receive, and close. Users can choose the corresponding option by entering send, receive, or close.

```
while True:
    option = input("Please input your option: send/receive/close: ")
    if option.lower() not in ["send", "receive", "close"]:
        print("Invalid option. Please enter 'send', 'receive', or 'close'.")
    else:
        break
```

If the user chooses to send, the program creates a thread and calls the `sender.send()` function to send the file.

If the user chooses to receive, the program creates a thread and calls the `receiver.receive()` function to receive the file.

If the user chooses to close, the program closes the socket and exits the loop.

During user input, input validation and error handling are performed to ensure the validity of the input.

When sending and receiving files, a mutex lock is used to ensure that access to shared variables between multiple threads is thread-safe.

```
lock = threading.Lock()
lock.acquire()
lock.release()
```

#### 4. System Deployment, Startup, and Use

System Requirements:

Python 3.x is installed on the machine.

The `Config.ini` file contains the following sections and keys:

The `[send]` section contains the following keys:

ip: a valid IP address

port: an integer between 1 and 65535

filename: a valid file path

The `[receive]` section contains the following keys:

ip: a valid IP address

port: an integer between 1 and 65535

filename: a valid directory path

Deployment:

Download the code and save it in a directory on the machine.

Create the Config.ini file and add the necessary sections and keys as described above.

Save the Config.ini file in the same directory as the code.

#### Startup:

Open a command prompt or terminal and navigate to the directory where the code and Config.ini file are located.

Run the following command:

```
python .\main.py
```

#### Usage:

After startup, select the 'send' option to send a file or the 'receive' option to receive a file.

Enter any necessary information as prompted.

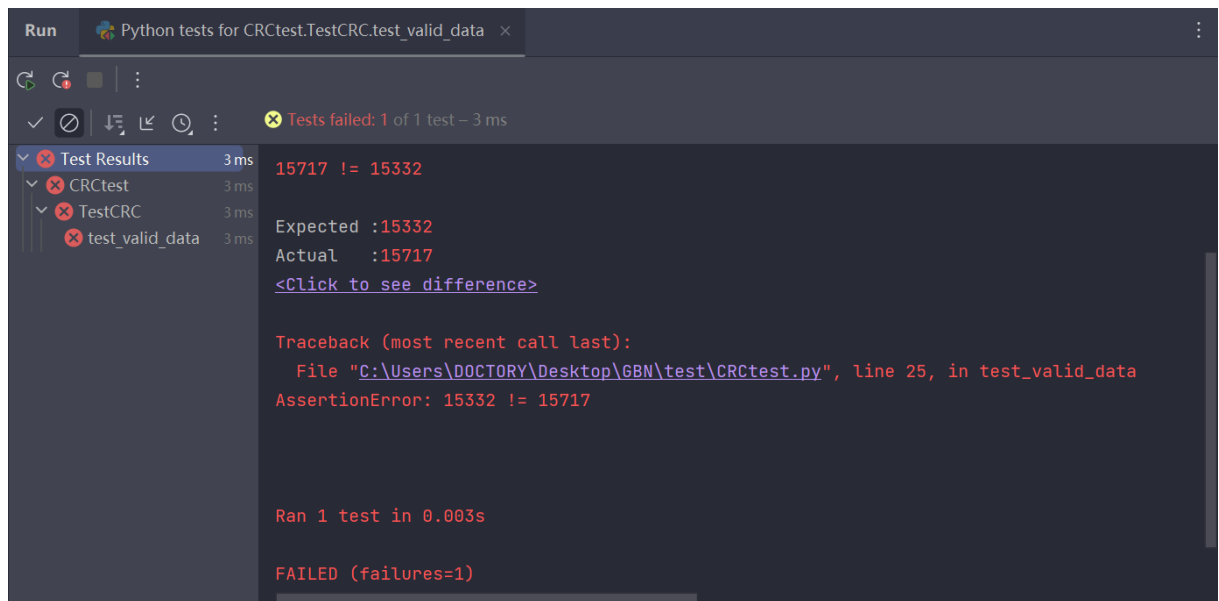
The system will send the specified file and display the transfer progress.

After the file transfer is complete, the host can choose to continue sending or receiving or to shut down the host.

## 5. System Test

Unit test:

CRC Test:



```
Run Python tests for CRCtest.TestCRC.test_valid_data x
Tests failed: 1 of 1 test - 3 ms

Test Results 3 ms
  CRCtest 3 ms
    TestCRC 3 ms
      test_valid_data 3 ms

15717 != 15332
Expected :15332
Actual :15717
<Click to see difference>

Traceback (most recent call last):
  File "C:\Users\DOCTORY\Desktop\GBN\test\CRCtest.py", line 25, in test_valid_data
AssertionError: 15332 != 15717

Ran 1 test in 0.003s

FAILED (failures=1)
```

Packet Test:

```
56 ▶ class TestPackingAndUnpacking(unittest.TestCase):
57 ▶     def test_packing(self):
58         # 测试正常输入
59         self.assertEqual(packing(1, 1234, b'hello'), b'\x01\x00\x00\x00\xd2\x04\x00\x00hello')
60
61         # 测试非整数类型的输入
62         with self.assertRaises(TypeError):
63             packing('1', 1234, b'hello')
64         with self.assertRaises(TypeError):
65             packing(1, '1234', b'hello')
66
67         # 测试非字节串类型的输入
68         with self.assertRaises(TypeError):
69             packing(1, 1234, 'hello')
70
```

TestPackingAndUnpacking > test\_packing() > with self.assertRaises(TypeError)...

Run Python tests for Packettest.TestPackingAndUnpacking.test\_p... ×

✓ Tests passed: 1 of 1 test - 0 ms

✓ Test Results 0ms C:\Users\DOCTORY\Desktop\GBN\venv\Scripts\python.exe D:/Jetbrains/apps/PyCharm-P/ch-0/231. Testing started at 0:09 ... Launching unittests with arguments python -m unittest Packettest.TestPackingAndUnpacking.t

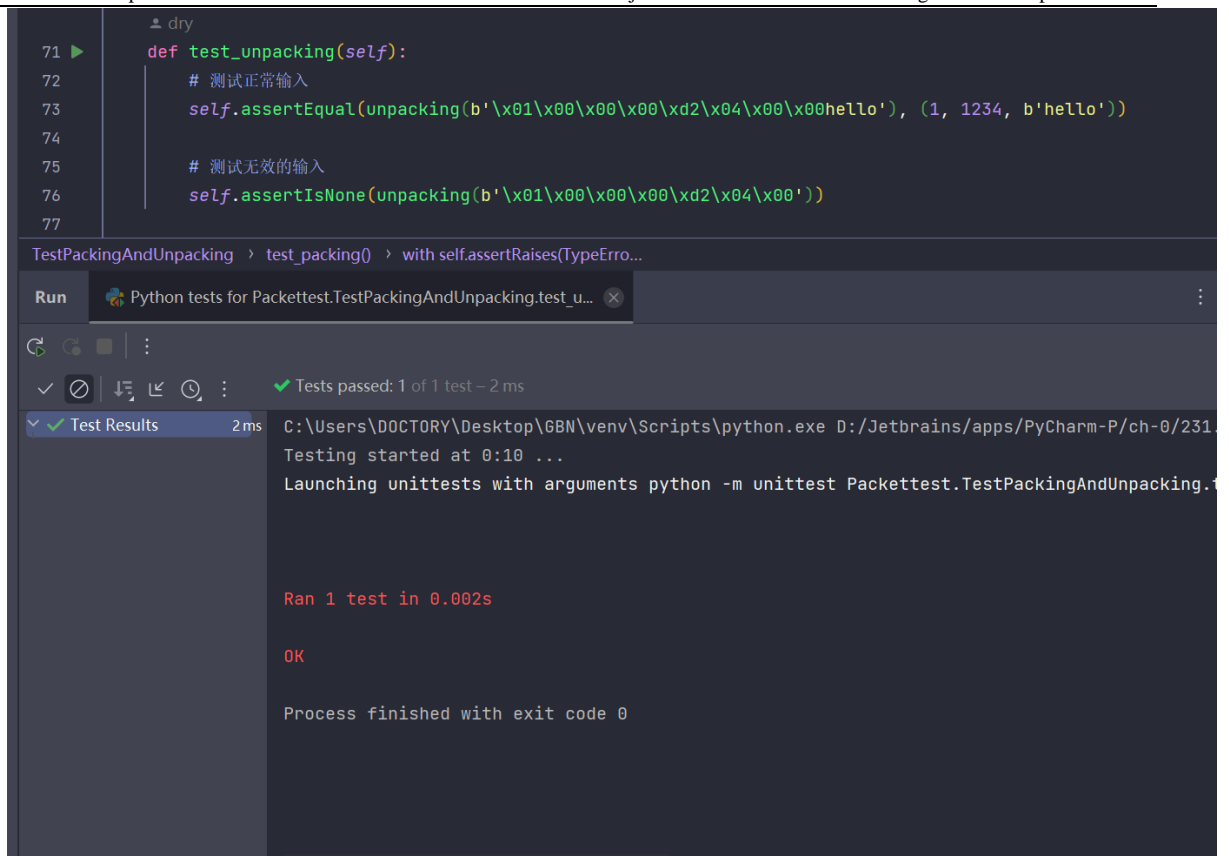
Ran 1 test in 0.002s

OK

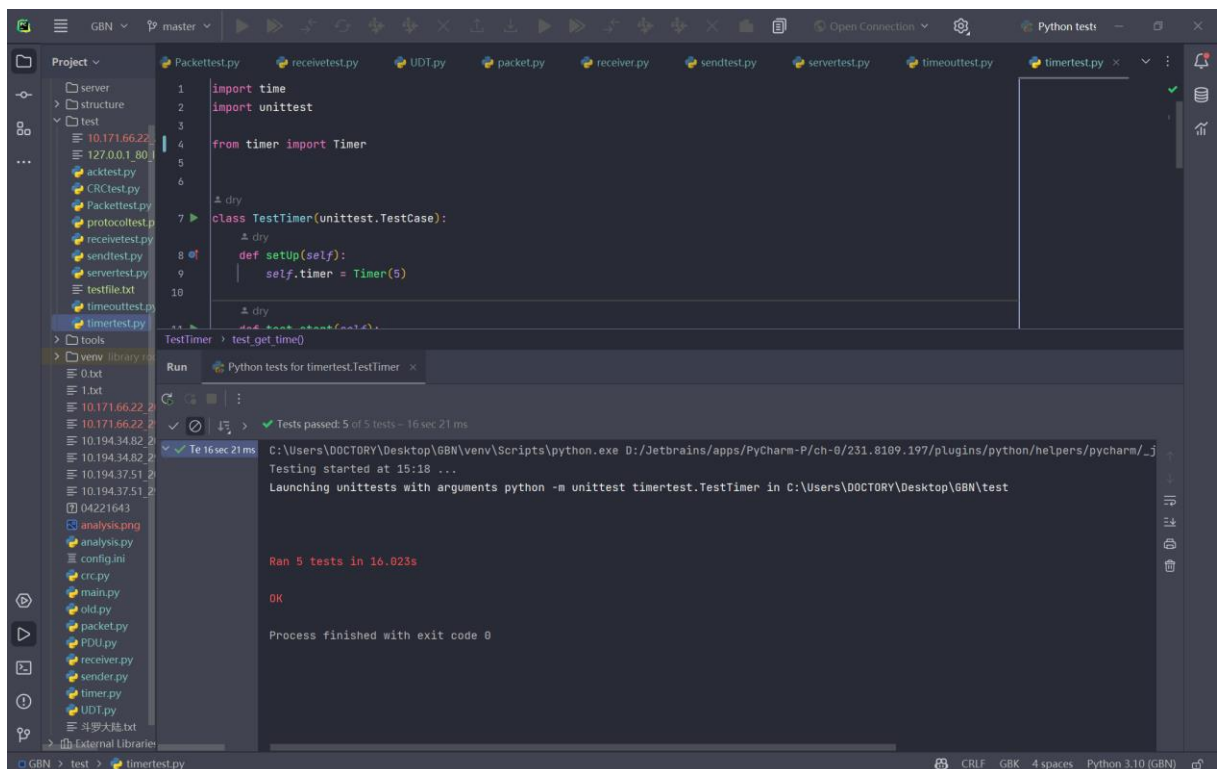
Process finished with exit code 0

Unpacket Test:





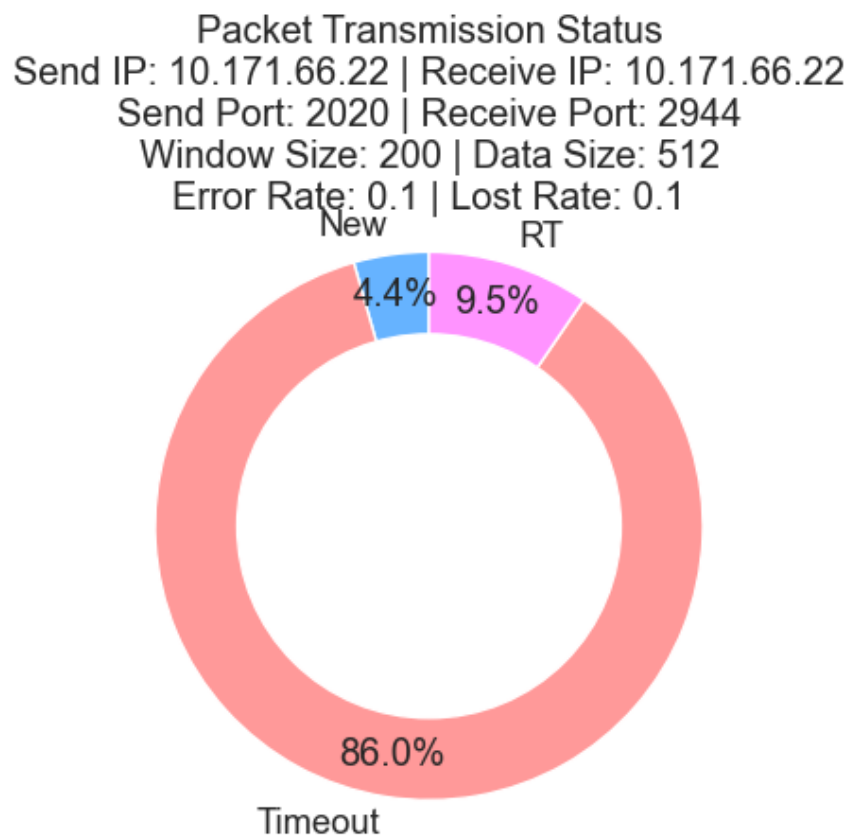
### Timer Test:



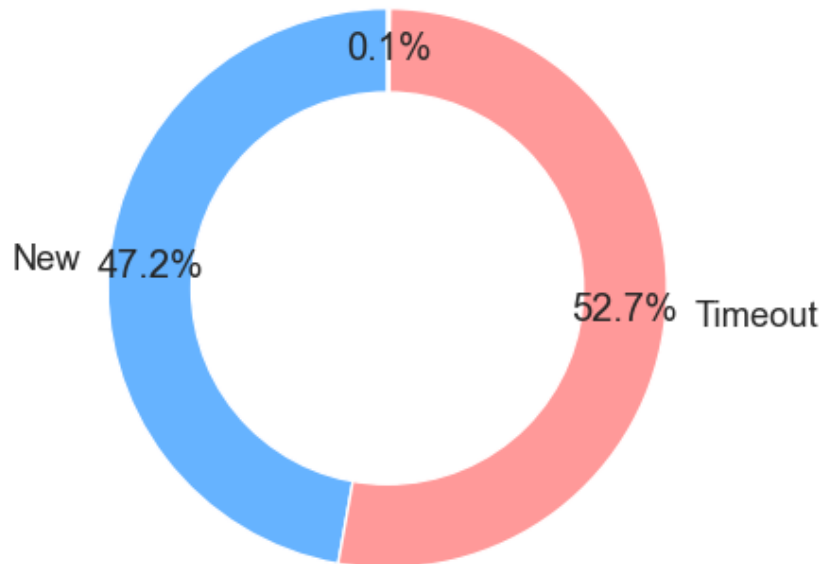
### Integrated test:

```
Terminal Local x Local (2) x + v  
Windows PowerShell  
版权所有 (C) Microsoft Corporation。保留所有权利。  
  
安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows  
  
(venv) PS C:\Users\DOCTORY\Desktop\6BN> python .\main.py  
Please input your option: send/receive/close: send  
Progress: ████████████████████████████████████████████████████████████ 100.0% (11713/11713) | Elapsed Time: 97.00s  
Task completed successfully!  
  
Terminal Local x Local (2) x + v  
Windows PowerShell  
版权所有 (C) Microsoft Corporation。保留所有权利。  
  
安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows  
  
(venv) PS C:\Users\DOCTORY\Desktop\6BN> python .\main.py  
Please input your option: send/receive/close: send  
Progress: ████████████████████████████████████████████████████████████ 100.0% (11713/11713) | Elapsed Time: 284.03s  
Task completed successfully!
```

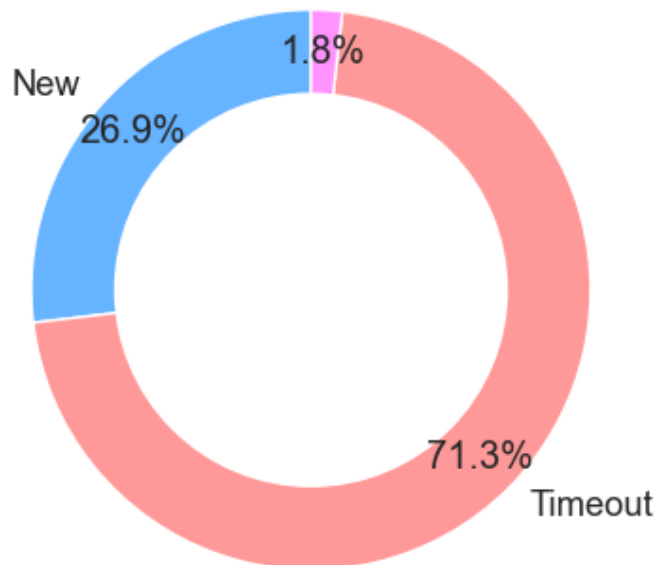
## 6. Performance and Analysis



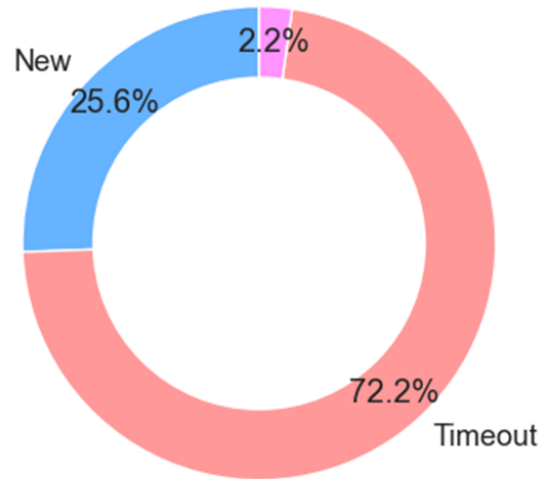
Packet Transmission Status  
Send IP: 10.194.156.11 | Receive IP: 10.194.156.11  
Send Port: 2020 | Receive Port: 2944  
Window Size: 200 | Data Size: 512  
Error Rate: 0.001 | Lost Rate: 0.001  
RT



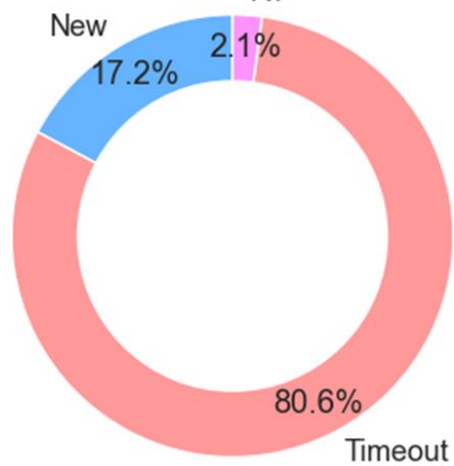
Packet Transmission Status  
Send IP: 10.194.156.11 | Receive IP: 10.194.156.11  
Send Port: 2020 | Receive Port: 2944  
Window Size: 511 | Data Size: 1024  
Error Rate: 0.01 | Lost Rate: 0.01  
RT

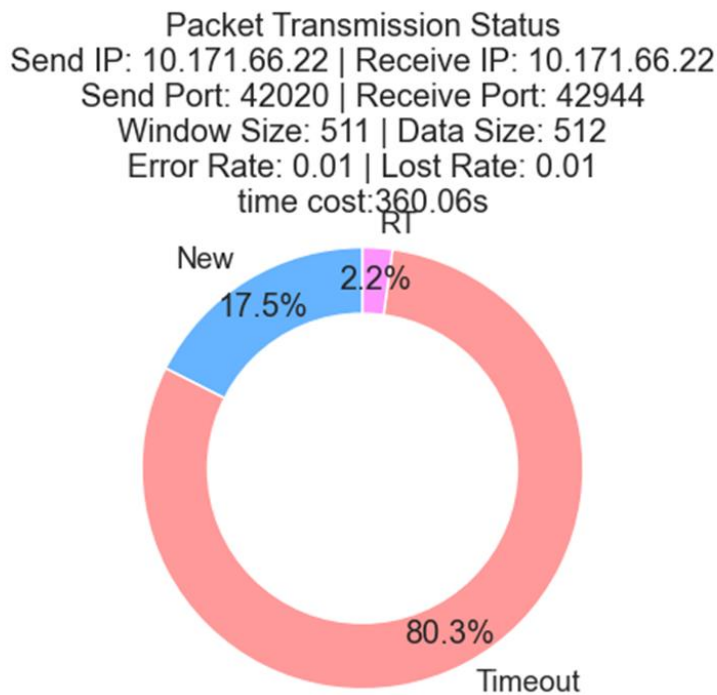


Packet Transmission Status  
Send IP: 10.171.66.22 | Receive IP: 10.171.66.22  
Send Port: 42020 | Receive Port: 42944  
Window Size: 1023 | Data Size: 512  
Error Rate: 0.01 | Lost Rate: 0.01  
RT



Packet Transmission Status  
Send IP: 10.171.66.22 | Receive IP: 10.171.66.22  
Send Port: 42020 | Receive Port: 42944  
Window Size: 1023 | Data Size: 256  
Error Rate: 0.01 | Lost Rate: 0.01  
time cost: 686.82s  
RT





Obviously, the transmission efficiency significantly decreases with the increase of error rate and loss rate, which is the disadvantage of the GBN protocol: If a data packet is lost, all data packets in the window must be resent, which may waste a lot of network bandwidth and time. Therefore, in a high packet loss rate network, the efficiency of the GBN protocol may be affected.

Within a certain range, the smaller the packet size, the more packets need to be transmitted, and the more likely transmission errors will occur, leading to retransmissions. Thus, when the packet size is halved, the total transmission time also increases by about twice. Smaller packet sizes lead to higher packet loss rates and lower transmission efficiency. Therefore, when choosing packet sizes, one needs to consider both packet loss rates and network bandwidth limitations to select an appropriate packet size to improve transmission efficiency.

Within a certain range, the more sending windows, the higher the transmission efficiency, because in the absence of errors, more data packets are sent in parallel, and fewer ACKs need to be processed. The larger the window size  $W$ , the higher the transmission efficiency. However,  $W$  that is too large can cause network congestion and increase packet loss rate, thereby reducing transmission efficiency. Therefore, when choosing window size, a balance and adjustment are necessary.

## 7. Summary or Conclusions

The purpose of this experiment is to design and implement a reliable file transfer system based on the Go-Back-N (GBN) protocol. This system can achieve bidirectional file transfer, support full-duplex communication, and can configure communication parameters through a configuration file, record communication status, and perform statistical analysis. In this system, we define a frame (PDU) structure and add a checksum field at the end of the PDU. We use the CRC-CCITT standard to implement the checksum. We use the UDP Socket API to simulate and implement the sending and receiving of PDUs. Each UDP datagram encapsulates a PDU. At the same time, we also implement a generator or method that allows PDUs to be randomly generated with errors and losses based on the percentage given in the configuration file to simulate problems that may occur in a real network environment. In the implementation process, we use the GBN protocol to solve the problems that may occur during data transmission, such as packet loss, errors, and retransmission. We also implement sliding windows, acknowledgment mechanisms, timeout retransmission, and other technical solutions to improve the reliability and efficiency of the system. Tests show that the system can correctly transfer and receive files and can handle a large number of PDUs, showing good robustness and reliability. After the file transfer is complete, the received and saved file is identical to the original file sent. Additionally, we also implemented a program to read the log file and perform statistical analysis of the communication status. We compared the communication efficiency under different data sizes, window sizes, PDU error rates, PDU loss rates, and timeout values, and obtained analytical conclusions, providing a reliable and efficient solution for research in the field of network data transmission.

## 8. References

- [1] Tanenbaum, A. S. (2011). Computer Networks. Prentice Hall. (Chapter 3, Protocol 5)
- [2] Kurose, J. F., & Ross, K. W. (2017). Computer Networking: A Top-Down Approach. Pearson. (Chapter 3)
- [3] Forouzan, B. A. (2013). Data Communications and Networking. McGraw-Hill. (Chapter 18)
- [4] Wetherall, D., & LeBlanc, T. (2007). Computer Networks: A Systems Approach. Morgan Kaufmann. (Chapter 3)
- [5] Forouzan, B. A., & Fegan, S. C. (2017). TCP/IP Protocol Suite. McGraw-Hill. (Chapter 20)

## 9. Comments

This experiment involves a lot of content, requiring not only knowledge of computer networking but also a certain understanding of network implementation in programming languages. Additionally, statistical analysis of data will be necessary, which tests knowledge reserves in data science. For a personal project, a few weeks of operation time may be somewhat limited, and the workload may even be greater than that of some previous course designs. At the beginning of the project, there were many ideas, but the engineering was too massive, and it was difficult to know where to start after the project took shape. For third-year students, teamwork experience is required, and if team work can be allocated reasonably, the completion requirements of this project can be even higher. Ultimately, the project produced will be more practical.