哈尔滨工业大学

<<计算机网络>>

实验报告

(2018 年度春季学期)

姓名：	许家乐
学号：	1150310329
学院：	计算机学院
教师：	李全龙

实验三 可靠数据传输协议-GBN 协议的设计与实现

一、实验目的

理解滑动窗口协议的基本原理；掌握 GBN 的工作原理；掌握基于 UDP 设计并实现一个 GBN 协议的过程与技术。

二、实验内容

(1) 基于 UDP 设计一个简单的 GBN 协议，实现单向可靠数据传输（服务器到客户的数据传输）。

(2) 模拟引入数据包的丢失，验证所设计协议的有效性。

(3) 改进所设计的 GBN 协议，支持双向数据传输；（选作内容，加分项目）

(4) 将所设计的 GBN 协议改进为 SR 协议。（选作内容，加分项目）

三、实验过程及结果

1. GBN 协议实现原理

(1) 数据分组格式

seq	flag	checksum	data

seq：序列号，取值范围 0~255；

flag：传输结束标志，若是最后一个数据分组则为 1，不是则为 0；

checksum：数据校验和，长度为 8bit。

data：传输的数据，长度为 2048 字节。

(2) 确认分组格式

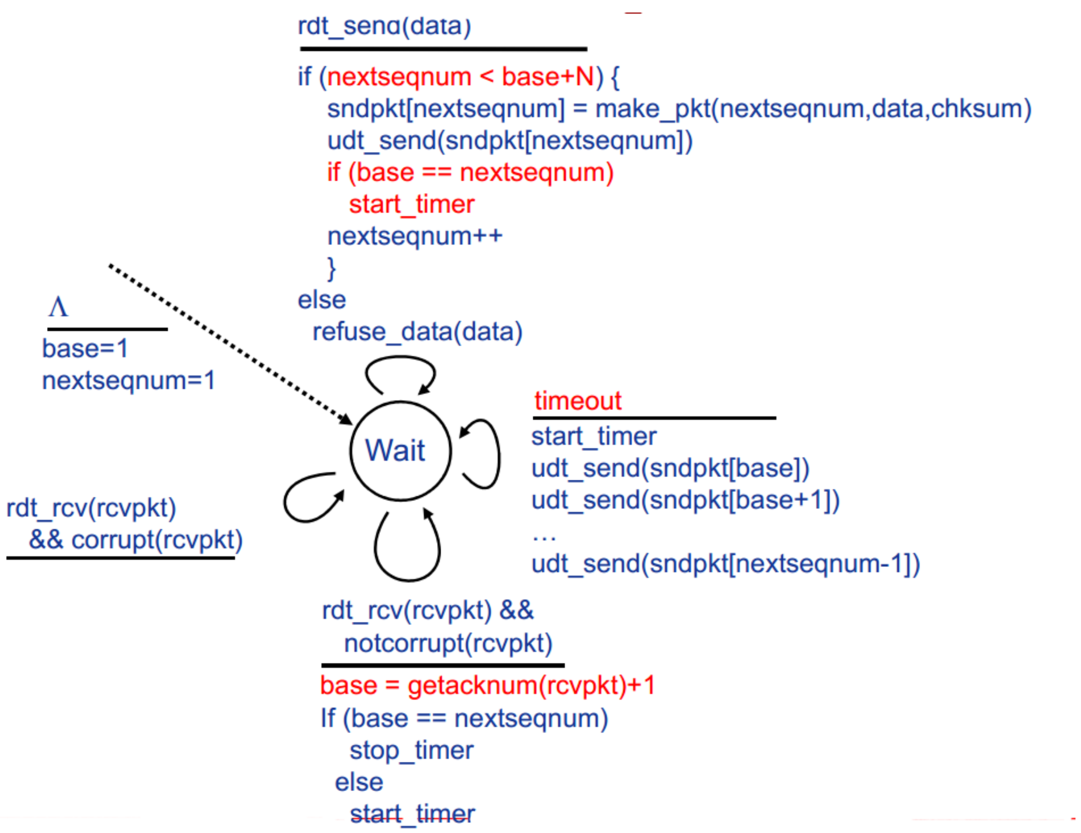
ack_seq	expect_seq

ack_seq：最近一次确认的数据分组的序列号，取值范围 0~255；

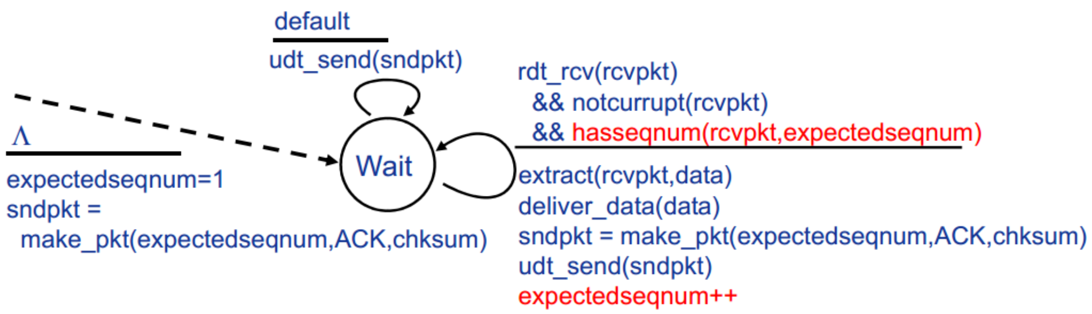
expect_seq：接收端期望收到的数据分组的序列号，取值 0~255。

(3) 协议两端程序流程图

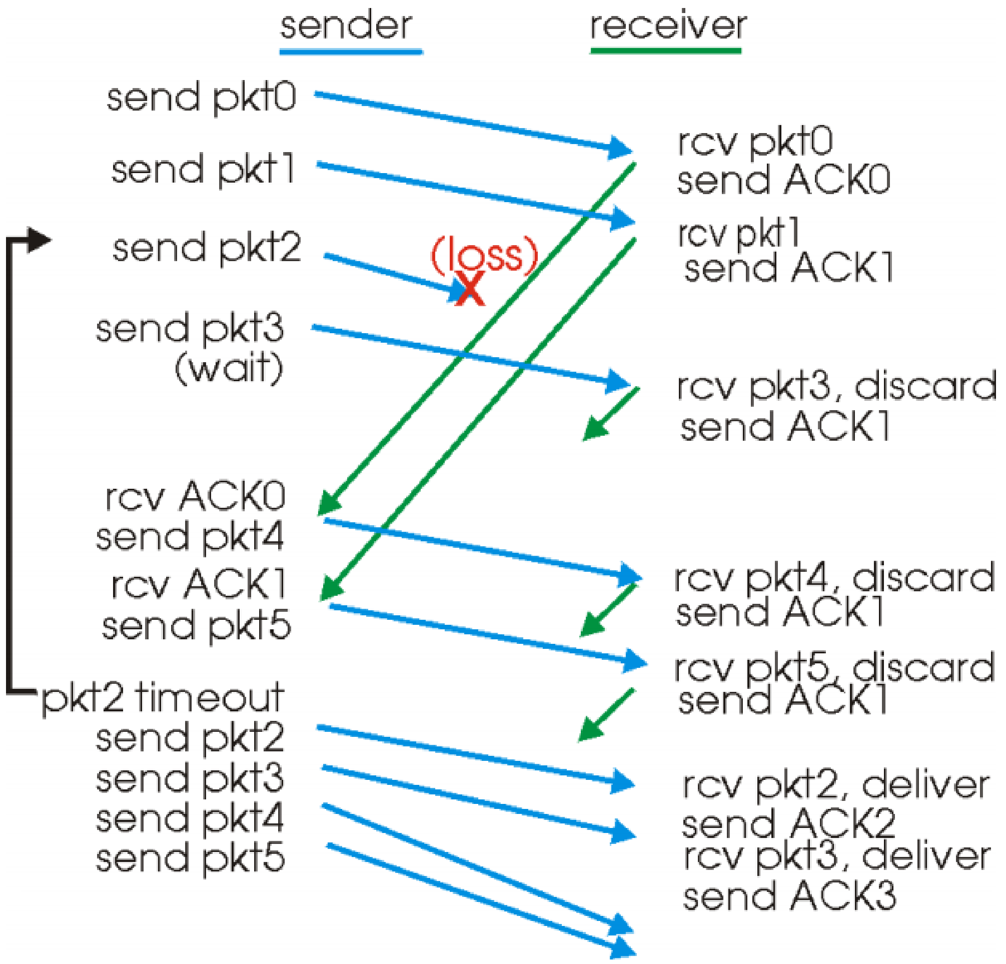
发送端(client)：



接收端(server)：



(4) 典型交互过程



2. SR 协议实现原理

(1) 数据分组格式

seq	flag	checksum	data

seq：序列号，取值范围 0~255；

flag：传输结束标志，若是最后一个数据分组则为 1，不是则为 0；

checksum：数据校验和，长度为 8bit。

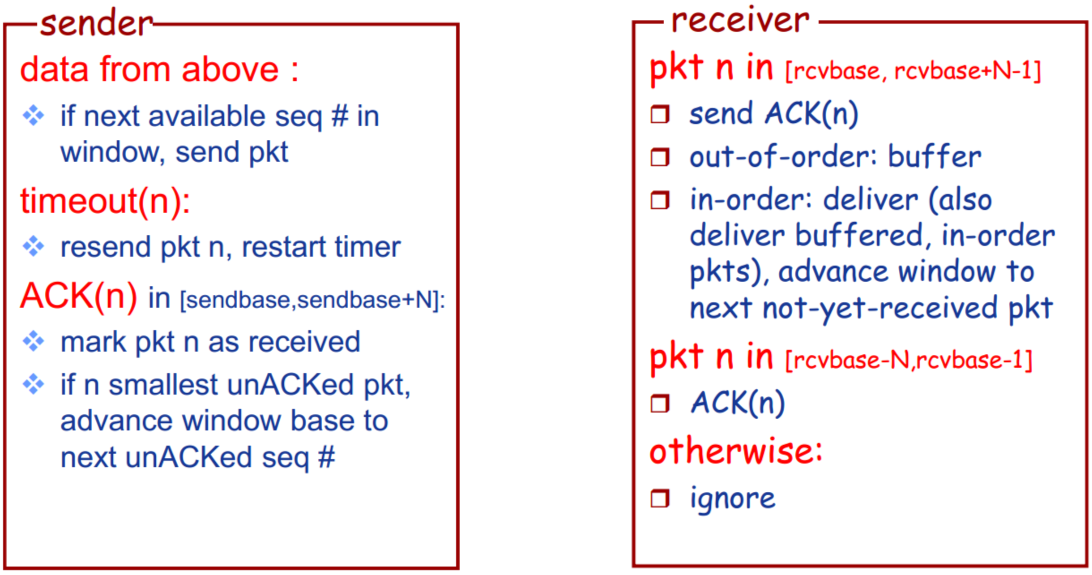
data：传输的数据，长度为 2048 字节。

(2) 确认分组格式

ack_seq：最近一次确认的数据分组的序列号，取值范围 0~255；

expect_seq：接收端期望收到的数据分组的序列号，取值 0~255。

(3) 协议两端程序流程图



sender

data from above :

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n) in [sendbase,sendbase+N]:

- mark pkt n as received
- if n smallest unACKed pkt, advance window base to next unACKed seq #

receiver

pkt n in [rcvbase, rcvbase+N-1]

❒ send ACK(n)

❒ out-of-order: buffer

❒ in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

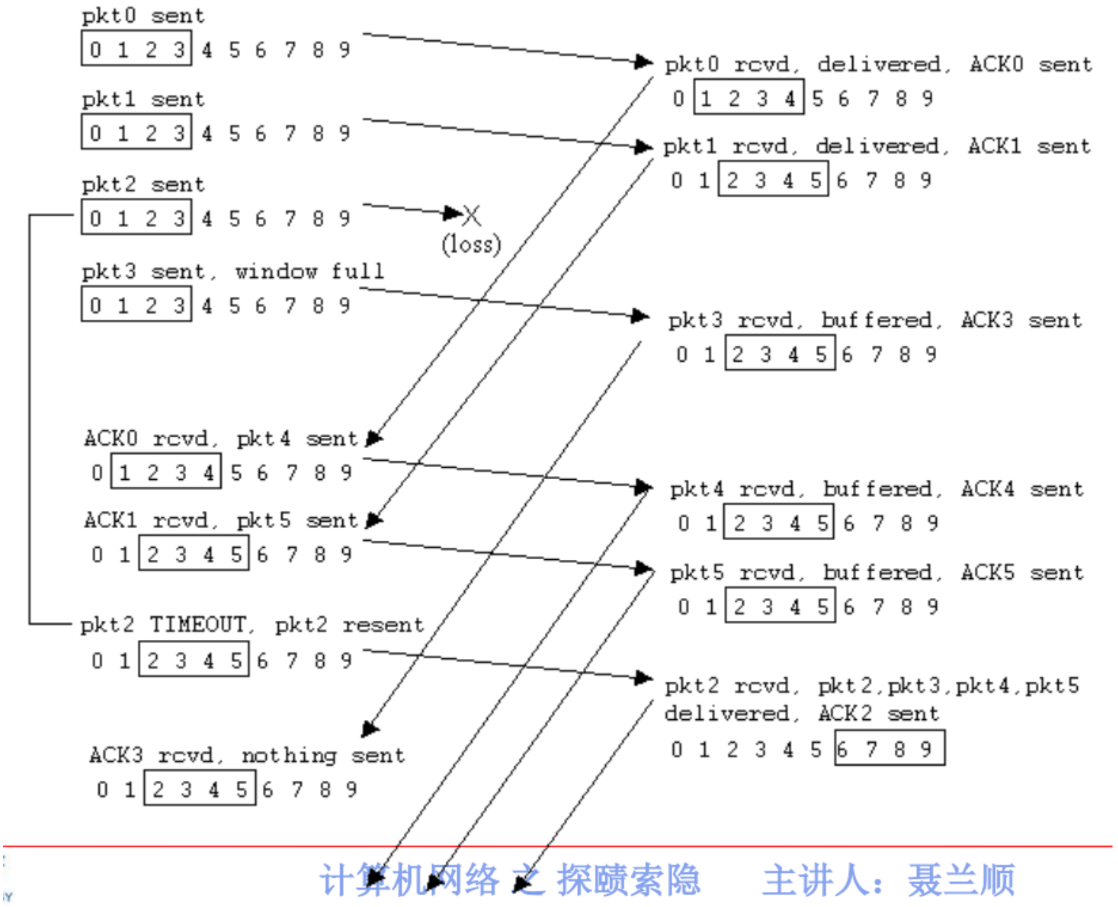
pkt n in [rcvbase-N,rcvbase-1]

❒ ACK(n)

otherwise:

❒ ignore

(4) 典型交互过程



3. 分组丢失模拟方法

首先，设置分组丢失概率 LOSS_RATE，取值范围为[0, 1)，在每次使用 UDP 发送数据分组之前，产生一个取值范围为[1, 1/LOSS_RATE]的随机整数，若该整数恰好等于 1，则该分组丢失，不再发送；若该整数不等于 1，则正常发送。确认分组 ACK 丢失模拟方法同理。

4. GBN 协议双向数据传输的实现原理

在客户端(client)与服务器端(server)分别建立一个 GBN 发送端(sender)和一个 GBN 接收端(receiver)，使用两对会话模拟双向数据传输。

5. 程序中的主要类及函数

(1) gbn.py

此文件中主要实现了 GBN 协议的发送端和接收端类。

①函数 getChecksum：计算数据的校验和

②类 GBNSender：GBN 协议的发送端

属性：

self.sender_socket：发送端套接字

self.timeout：超时时间

self.address：目的地址（IP 地址，端口号）

self.window_size：窗口大小

self.loss_rate：数据分组丢失概率

self.send_base：窗口头部序列号

self.next_seq：下一个可用序列号

self.packets：数据分组

方法：

self.udp_send：使用 udp 发送数据分组

self.wait_ack：收到确认分组后进行的处理操作

self.make_pkt：打包生成数据分组

self.analyse_pkt：分析收到的确认分组

③类 GBNReceiver：GBN 协议的接收端

属性：

self.receiver_socket：接收端套接字

self.timeout：超时时间

self.loss_rate：确认分组丢失概率

self.expect_seq：期望接收到的数据分组序列号

self.target：确认分组的发送目标地址（即发送端地址）

方法：

self.udp_send：使用 udp 发送确认分组

self.wait_data：收到数据分组后进行的操作

self.make_pkt：打包生成确认分组

self.analyse_pkt：分析收到的数据分组

(2) sr.py

此文件中主要实现了 SR 协议的发送端和接收端类。

② 函数 getChecksum：计算数据的校验和

②类 SRSender：SR 协议的发送端

属性：

self.sender_socket：发送端套接字

self.timeout：超时时间

self.address：目的地址（IP 地址，端口号）

self.window_size：发送窗口大小

self.loss_rate：数据分组丢失概率

self.send_base：窗口头部序列号

self.next_seq：下一个可用序列号

self.packets：数据分组

self.acks：标记数据分组是否已被确认

方法：

self.udp_send：使用 udp 发送数据分组

self.wait_ack：收到确认分组后进行的处理操作

self.make_pkt：打包生成数据分组

self.analyse_pkt：分析收到的确认分组

③ 类 SRReceiver：SR 协议的接收端

属性：

self.receiver_socket：接收端套接字

self.timeout：超时时间

self.window_size：接收窗口大小

self.loss_rate：确认分组丢失概率

self.recv_base：接收窗口头部序列号

self.recvs：收到的数据分组

self.target：确认分组的发送目标地址（即发送端地址）

方法：

self.udp_send：使用 udp 发送确认分组

self.wait_data：收到数据分组后进行的操作

self.make_pkt：打包生成确认分组

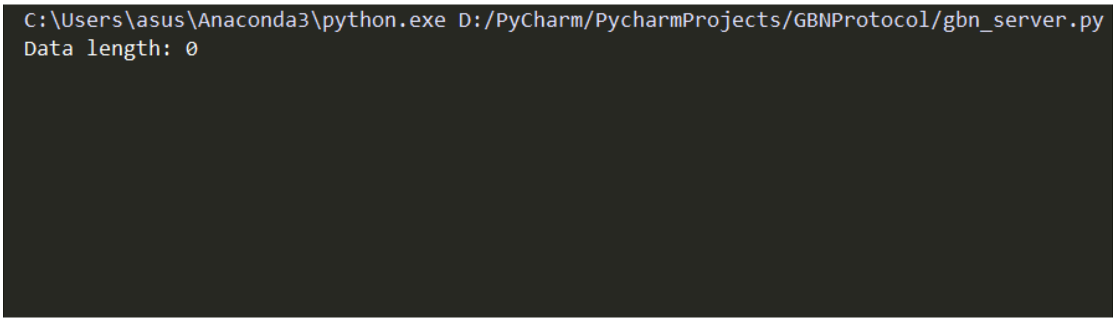
self.analyse_pkt：分析收到的数据分组

6. 实验结果

(1) GBN 协议测试：

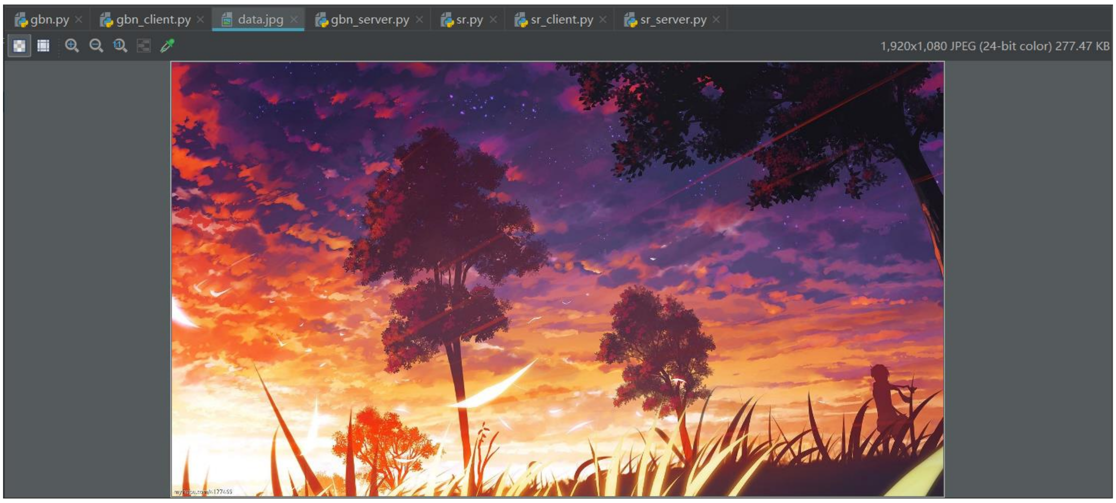
首先，将分组丢失概率设置为 0.1。

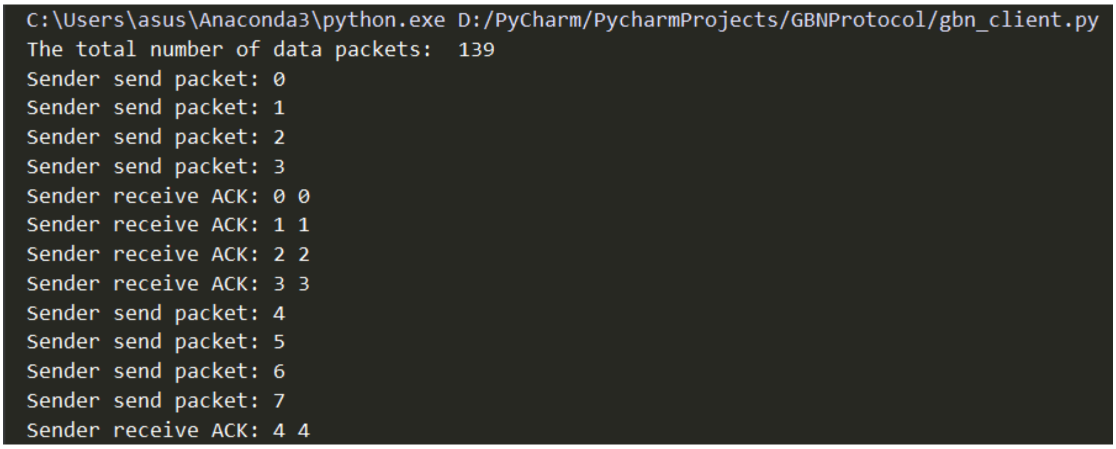
运行 gbn_server.py，使接收端处于监听状态：



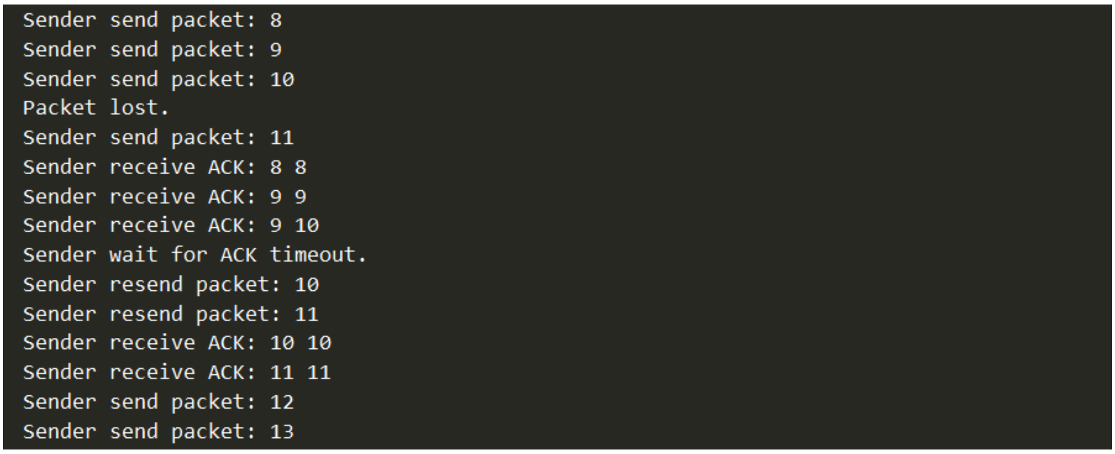
```
C:\Users\asus\Anaconda3\python.exe D:/PyCharm/PycharmProjects/GBNProtocol/gbn_server.py
Data length: 0
```

接着，运行 gbn_client.py，通过发送端向接收端发送数据文件 client/data.jpg（文件大小 284127 字节，因此数据分组个数为 284127/2048=139）：

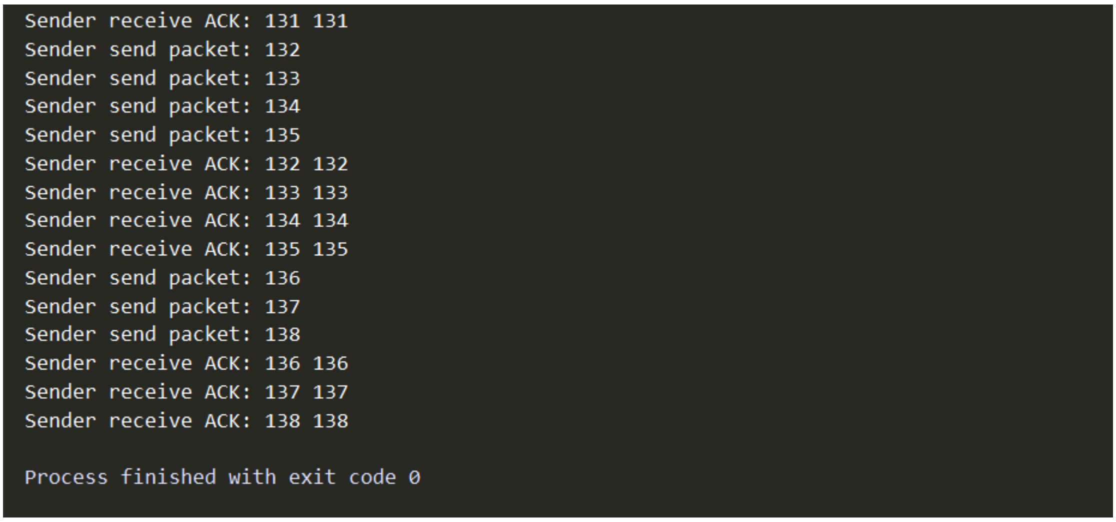




```
C:\Users\asus\Anaconda3\python.exe D:/PyCharm/PycharmProjects/GBNProtocol/gbn_client.py
The total number of data packets:  139
Sender send packet: 0
Sender send packet: 1
Sender send packet: 2
Sender send packet: 3
Sender receive ACK: 0 0
Sender receive ACK: 1 1
Sender receive ACK: 2 2
Sender receive ACK: 3 3
Sender send packet: 4
Sender send packet: 5
Sender send packet: 6
Sender send packet: 7
Sender receive ACK: 4 4
```

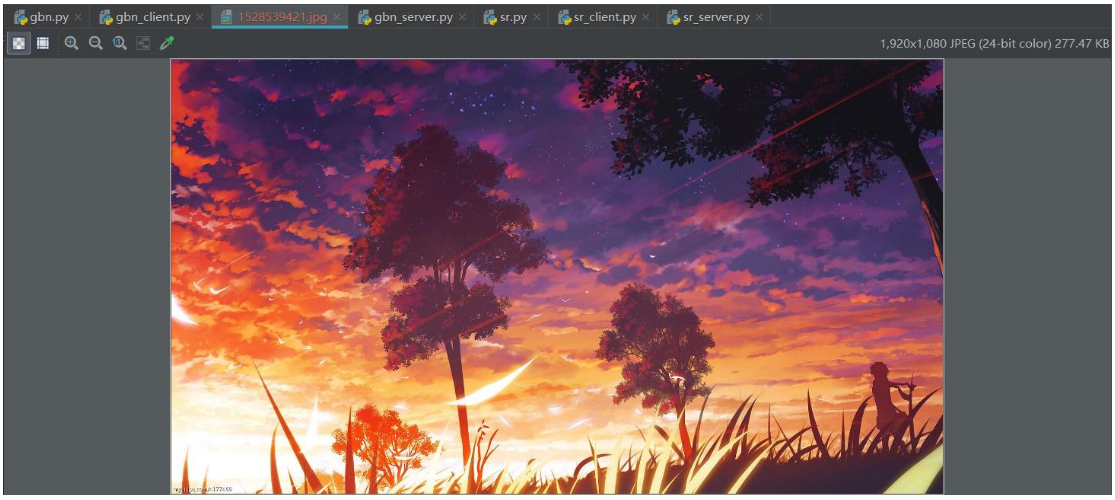


```
Sender send packet: 8
Sender send packet: 9
Sender send packet: 10
Packet lost.
Sender send packet: 11
Sender receive ACK: 8 8
Sender receive ACK: 9 9
Sender receive ACK: 9 10
Sender wait for ACK timeout.
Sender resend packet: 10
Sender resend packet: 11
Sender receive ACK: 10 10
Sender receive ACK: 11 11
Sender send packet: 12
Sender send packet: 13
```



```
Sender receive ACK: 131 131
Sender send packet: 132
Sender send packet: 133
Sender send packet: 134
Sender send packet: 135
Sender receive ACK: 132 132
Sender receive ACK: 133 133
Sender receive ACK: 134 134
Sender receive ACK: 135 135
Sender send packet: 136
Sender send packet: 137
Sender send packet: 138
Sender receive ACK: 136 136
Sender receive ACK: 137 137
Sender receive ACK: 138 138

Process finished with exit code 0
```

传输结果（server/1528539421.jpg，文件使用时间戳命名）：

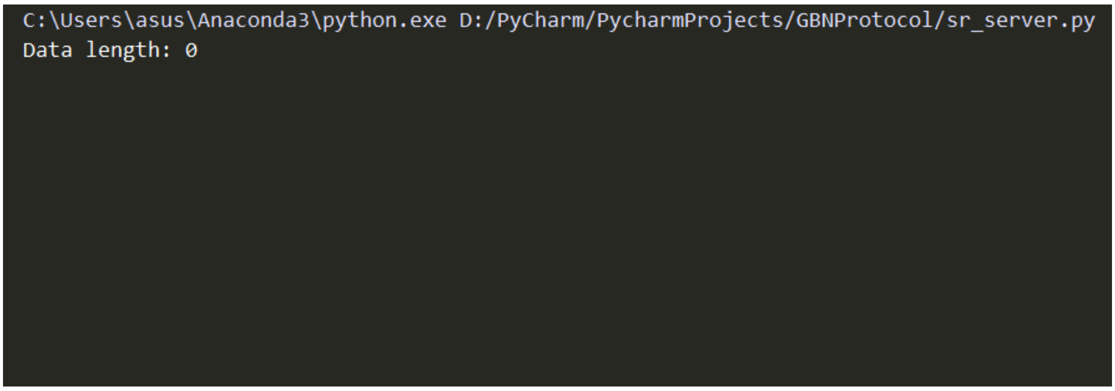


可见数据传输成功，接收端正确地接收到了发送端发送的数据文件。

(2) SR 协议测试

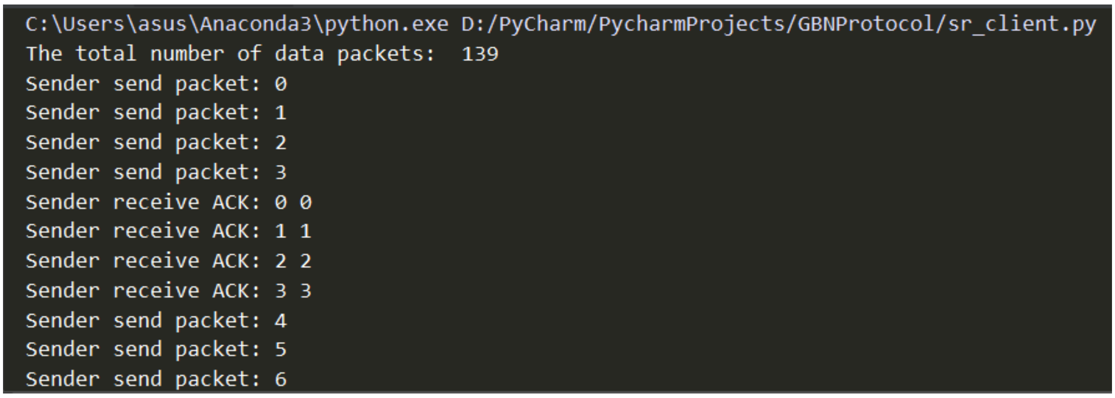
首先，将分组丢失概率设置为 0.1。

运行 sr_server.py，使得接收端处于监听状态：

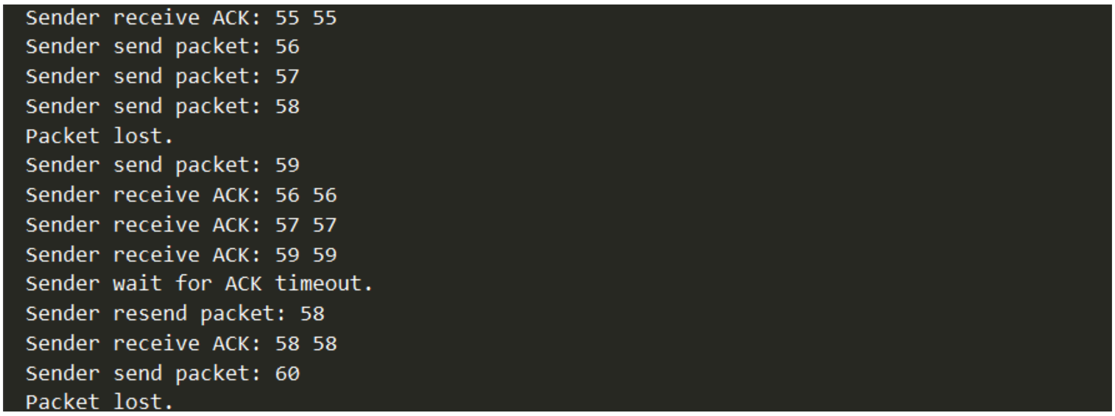


```
C:\Users\asus\Anaconda3\python.exe D:/PyCharm/PycharmProjects/GBNProtocol/sr_server.py
Data length: 0
```

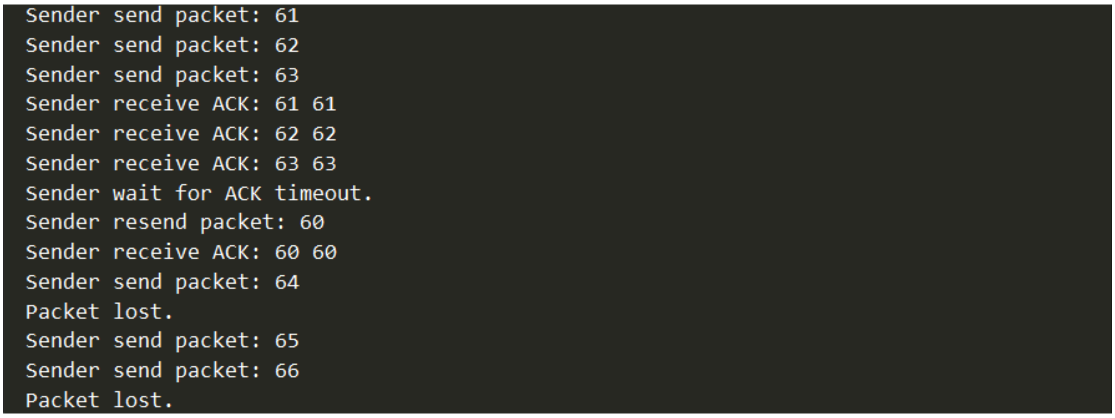
接着，运行 sr_client.py，通过发送端向接收端发送数据文件 client/data.jpg（文件大小 284127 字节，因此数据分组个数为 284127/2048=139）：



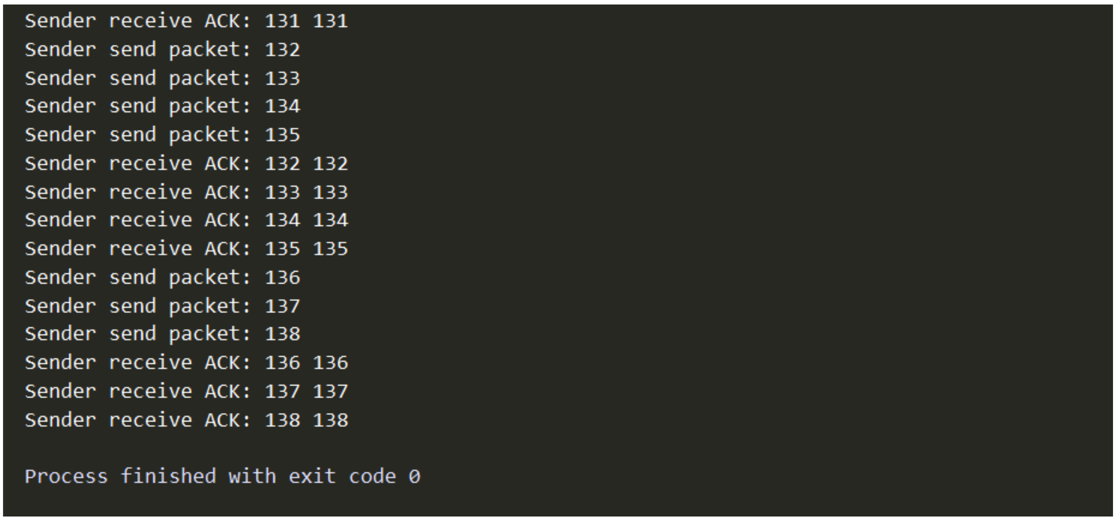
```
C:\Users\asus\Anaconda3\python.exe D:/PyCharm/PycharmProjects/GBNProtocol/sr_client.py
The total number of data packets:  139
Sender send packet: 0
Sender send packet: 1
Sender send packet: 2
Sender send packet: 3
Sender receive ACK: 0 0
Sender receive ACK: 1 1
Sender receive ACK: 2 2
Sender receive ACK: 3 3
Sender send packet: 4
Sender send packet: 5
Sender send packet: 6
```



```
Sender receive ACK: 55 55
Sender send packet: 56
Sender send packet: 57
Sender send packet: 58
Packet lost.
Sender send packet: 59
Sender receive ACK: 56 56
Sender receive ACK: 57 57
Sender receive ACK: 59 59
Sender wait for ACK timeout.
Sender resend packet: 58
Sender receive ACK: 58 58
Sender send packet: 60
Packet lost.
```

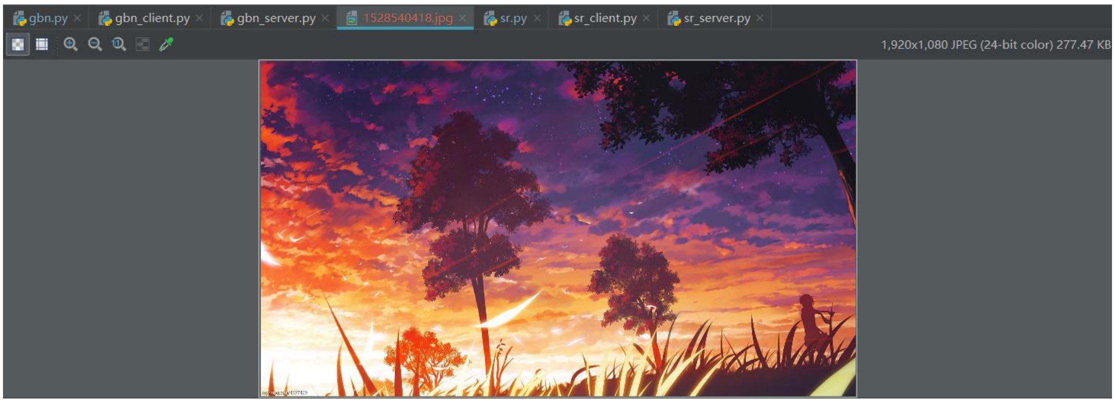


```
Sender send packet: 61
Sender send packet: 62
Sender send packet: 63
Sender receive ACK: 61 61
Sender receive ACK: 62 62
Sender receive ACK: 63 63
Sender wait for ACK timeout.
Sender resend packet: 60
Sender receive ACK: 60 60
Sender send packet: 64
Packet lost.
Sender send packet: 65
Sender send packet: 66
Packet lost.
```



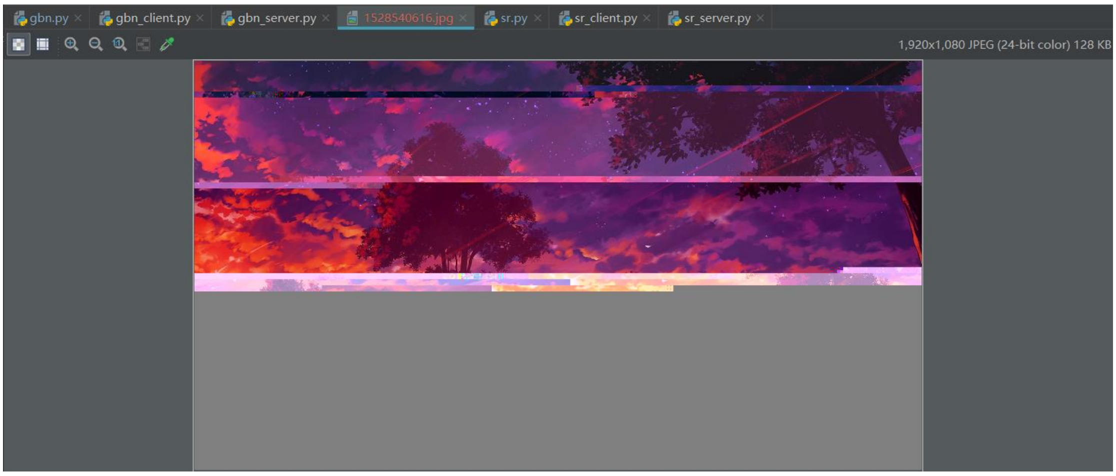
```
Sender receive ACK: 131 131
Sender send packet: 132
Sender send packet: 133
Sender send packet: 134
Sender send packet: 135
Sender receive ACK: 132 132
Sender receive ACK: 133 133
Sender receive ACK: 134 134
Sender receive ACK: 135 135
Sender send packet: 136
Sender send packet: 137
Sender send packet: 138
Sender receive ACK: 136 136
Sender receive ACK: 137 137
Sender receive ACK: 138 138

Process finished with exit code 0
```

传输结果（server/1528540418.jpg，文件使用时间戳命名）：



可见数据传输成功，接收端正确地接收到了发送端发送的数据文件。

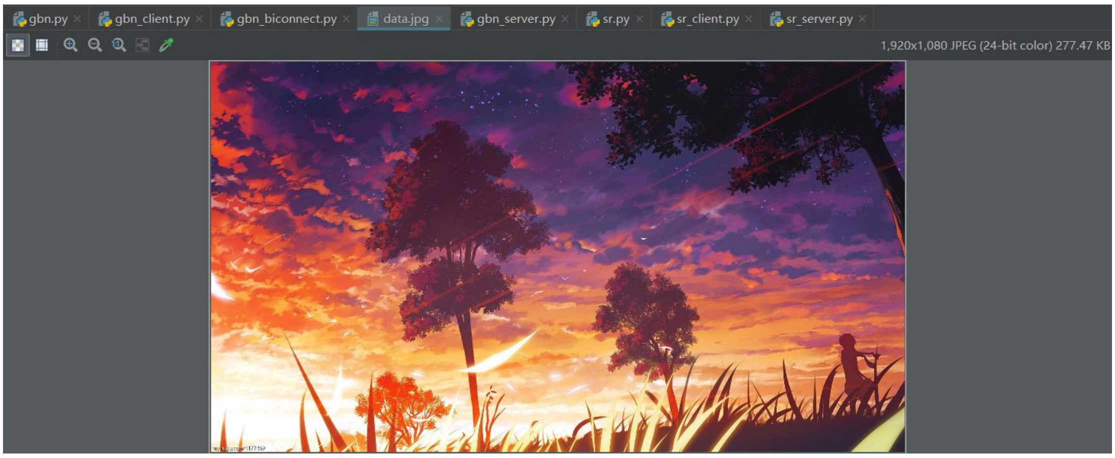
附：数据文件未传输完毕时的图像，证明传输过程的有效性



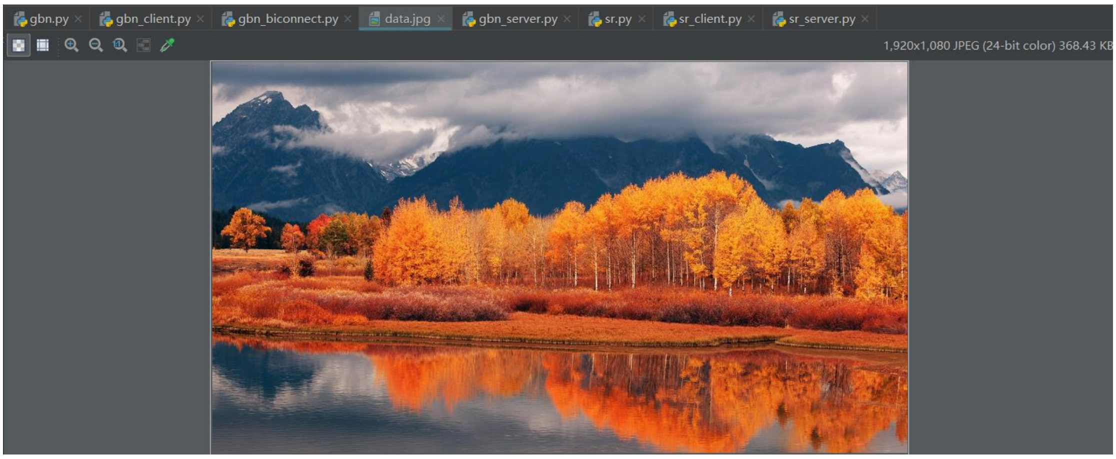
(3) GBN 协议双向数据传输测试

运行 gbn_biconnect.py，进行双向数据传输测试。

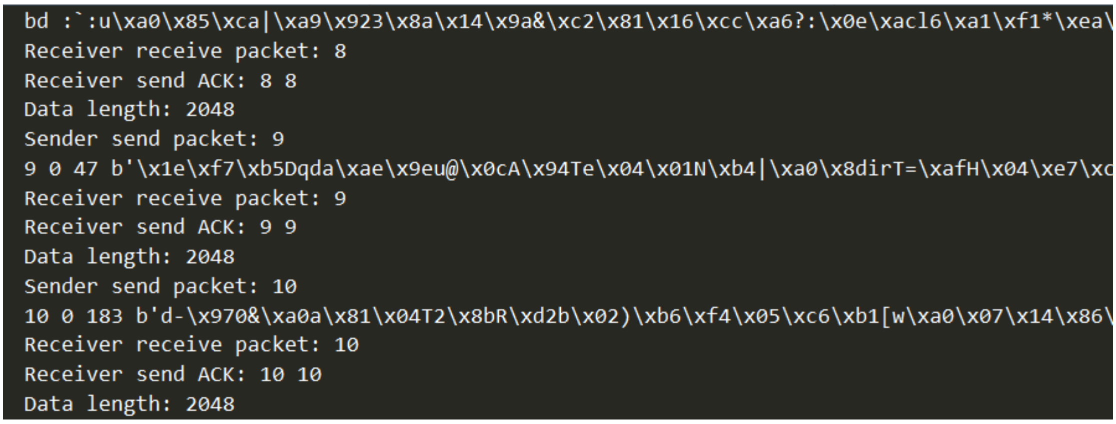
其中，client 向 server 发送 client/data.jpg 文件（大小：284127 字节）：



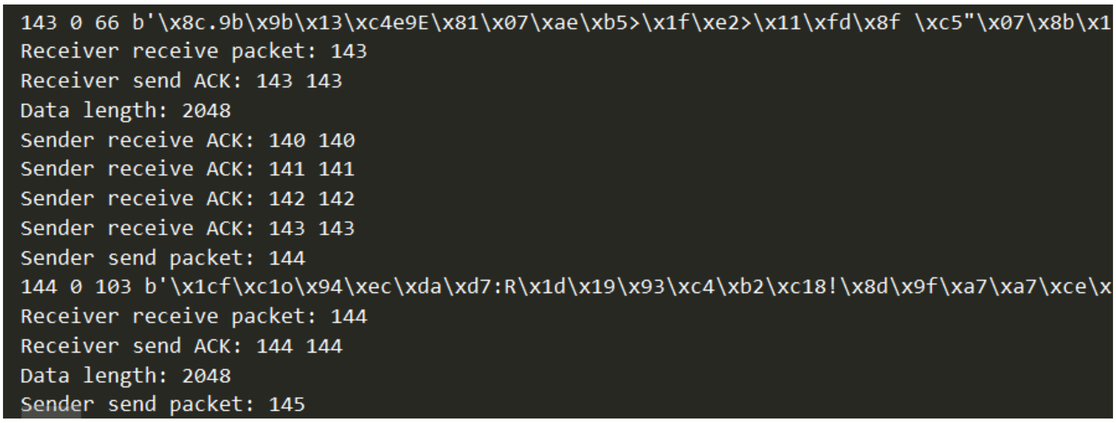
server 向 client 发送 server/data.jpg 文件（大小：377271 字节）：



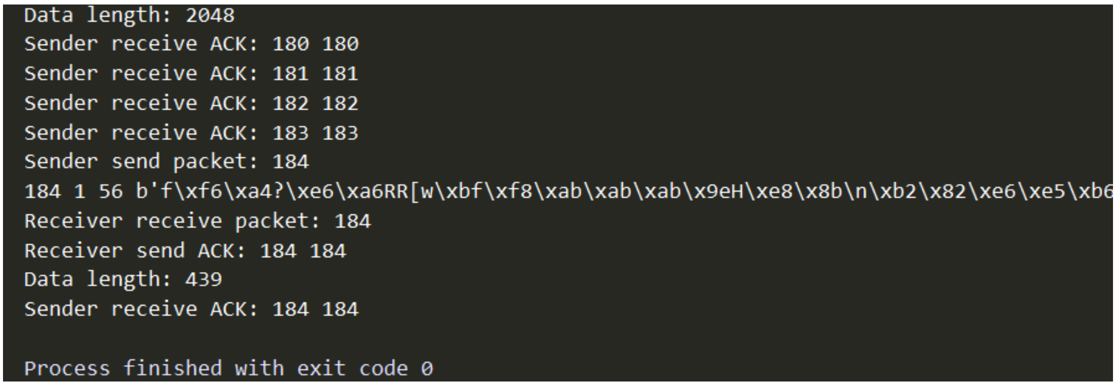
传输过程：



```
bd :`:u\xa0\x85\xca|\xa9\x923\x8a\x14\x9a&\xc2\x81\x16\xcc\xa6?:\x0e\xacl6\xa1\xf1*\xea\
Receiver receive packet: 8
Receiver send ACK: 8 8
Data length: 2048
Sender send packet: 9
9 0 47 b'\x1e\xf7\xb5Dqda\xae\x9eu@\x0cA\x94Te\x04\x01N\xb4|\xa0\x8dirT=\xafH\x04\xe7\xc
Receiver receive packet: 9
Receiver send ACK: 9 9
Data length: 2048
Sender send packet: 10
10 0 183 b'd-\x970&\xa0a\x81\x04T2\x8bR\xd2b\x02)\xb6\xf4\x05\xc6\xb1[w\xa0\x07\x14\x86\
Receiver receive packet: 10
Receiver send ACK: 10 10
Data length: 2048
```



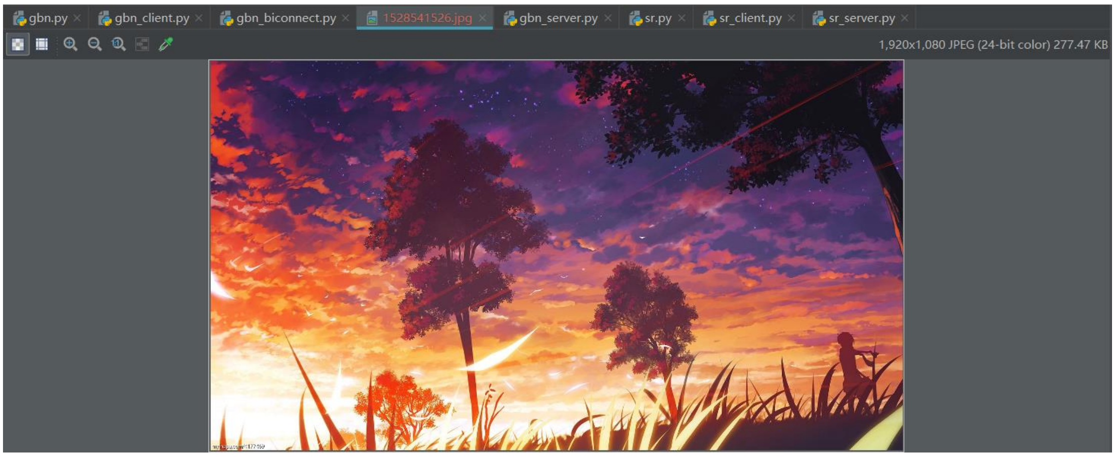
```
143 0 66 b'\x8c.9b\x9b\x13\xc4e9E\x81\x07\xae\xb5>\x1f\xe2>\x11\xfd\x8f \xc5"\x07\x8b\x1
Receiver receive packet: 143
Receiver send ACK: 143 143
Data length: 2048
Sender receive ACK: 140 140
Sender receive ACK: 141 141
Sender receive ACK: 142 142
Sender receive ACK: 143 143
Sender send packet: 144
144 0 103 b'\x1cf\xc1o\x94\xec\xda\xd7:R\x1d\x19\x93\xc4\xb2\xc18!\x8d\x9f\xa7\xa7\xce\x
Receiver receive packet: 144
Receiver send ACK: 144 144
Data length: 2048
Sender send packet: 145
```



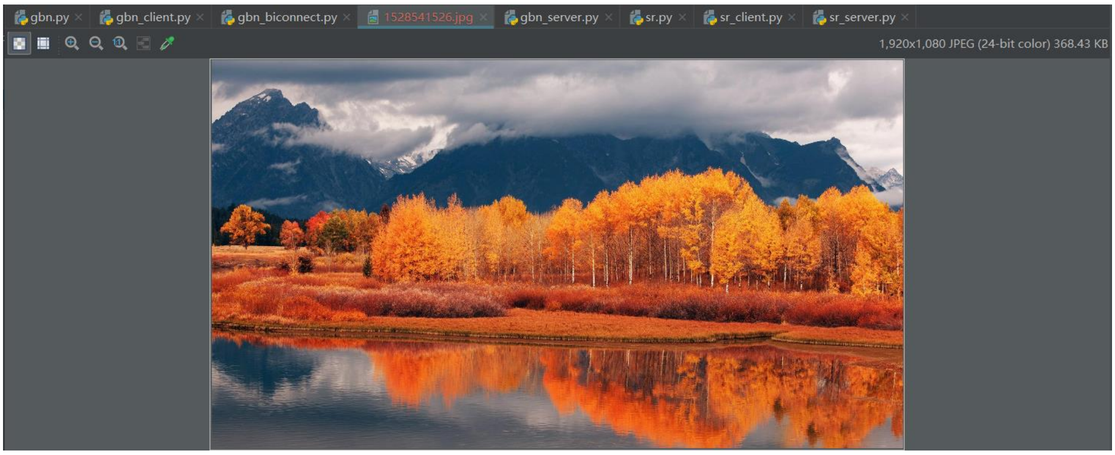
```
Data length: 2048
Sender receive ACK: 180 180
Sender receive ACK: 181 181
Sender receive ACK: 182 182
Sender receive ACK: 183 183
Sender send packet: 184
184 1 56 b'f\xf6\xa4?\xe6\xa6RR[w\xbf\xf8\xab\xab\xab\x9eH\xe8\x8b\n\xb2\x82\xe6\xe5\xb6
Receiver receive packet: 184
Receiver send ACK: 184 184
Data length: 439
Sender receive ACK: 184 184

Process finished with exit code 0
```

传输结果：

server 收到数据后写入 server/1528541526.jpg 文件：



client 收到数据后写入 client/1528541526.jpg 文件：



可见双向数据传输成功。

四、实验心得

通过本次实验，我有以下几点收获：

①理解了滑动窗口协议的基本原理；

②掌握了 GBN 协议的工作原理；

③掌握了 SR 协议的工作原理；

④掌握了基于 UDP 设计并实现一个可靠数据传输协议的过程与技术；

⑤进一步掌握了使用 Python 语言进行 socket 编程的方法和技术。

附录：源代码

1. gbn.py

```
import random
import socket
import struct
import time


BUFFER_SIZE = 4096
TIMEOUT = 10
WINDOW_SIZE = 4
LOSS_RATE = 0


def getChecksum(data):
    """
    char_checksum 按字节计算校验和。每个字节被翻译为无符号整数
    @param data: 字节串
    """
    length = len(str(data))
    checksum = 0
    for i in range(0, length):
        checksum += int.from_bytes(bytes(str(data)[i], encoding='utf-8'),
byteorder='little', signed=False)
        checksum &= 0xFF  # 强制截断

    return checksum


class GBNSender:
    def __init__(self, senderSocket, address, timeout=TIMEOUT,
                 windowSize=WINDOW_SIZE, lossRate=LOSS_RATE):
        self.sender_socket = senderSocket
        self.timeout = timeout
        self.address = address
        self.window_size = windowSize
        self.loss_rate = lossRate
        self.send_base = 0
        self.next_seq = 0
        self.packets = [None] * 256

    def udp_send(self, pkt):
        if self.loss_rate == 0 or random.randint(0, int(1 / self.loss_rate)) != 1:
            self.sender_socket.sendto(pkt, self.address)
        else:
            print('Packet lost.')
        time.sleep(0.3)

    def wait_ack(self):
        self.sender_socket.settimeout(self.timeout)

        count = 0
        while True:
            if count >= 10:
                # 连续超时 10 次，接收方已断开，终止
                break
            try:
                data, address = self.sender_socket.recvfrom(BUFFER_SIZE)

                ack_seq, expect_seq = self.analyse_pkt(data)
                print('Sender receive ACK:', ack_seq, expect_seq)

                if (self.send_base == (ack_seq + 1) % 256):
                    # 收到重复确认, 此处应当立即重发
                    pass
                    # for i in range(self.send_base, self.next_seq):
                    #     print('Sender resend packet:', i)
                    #     self.udp_send(self.packets[i])

                self.send_base = max(self.send_base, (ack_seq + 1) % 256)
                if self.send_base == self.next_seq:  # 已发送分组确认完毕
                    self.sender_socket.settimeout(None)
                    return True

            except socket.timeout:
                # 超时，重发分组.
                print('Sender wait for ACK timeout.')
                for i in range(self.send_base, self.next_seq):
                    print('Sender resend packet:', i)
                    self.udp_send(self.packets[i])
                self.sender_socket.settimeout(self.timeout)  # reset timer
                count += 1
        return False

    def make_pkt(self, seqNum, data, checksum, stop=False):
        """
        将数据打包
        """
        flag = 1 if stop else 0
        return struct.pack('BBB', seqNum, flag, checksum) + data
    def analyse_pkt(self, pkt):
        """
        分析数据包
        """
        ack_seq = pkt[0]
        expect_seq = pkt[1]
        return ack_seq, expect_seq


class GBNReceiver:
    def __init__(self, receiverSocket, timeout=10, lossRate=0):
        self.receiver_socket = receiverSocket
        self.timeout = timeout
        self.loss_rate = lossRate
        self.expect_seq = 0
        self.target = None

    def udp_send(self, pkt):
        if self.loss_rate == 0 or random.randint(0, 1 / self.loss_rate) != 1:
            self.receiver_socket.sendto(pkt, self.target)
            print('Receiver send ACK:', pkt[0], pkt[1])
        else:
            print('Receiver send ACK:', pkt[0], pkt[1], ', but lost.')

    def wait_data(self):
        """
        接收方等待接受数据包
        """
        self.receiver_socket.settimeout(self.timeout)

        while True:
            try:
                data, address = self.receiver_socket.recvfrom(BUFFER_SIZE)
                self.target = address

                seq_num, flag, checksum, data = self.analyse_pkt(data)
                print('Receiver receive packet:', seq_num)
                # 收到期望数据包且未出错
                if seq_num == self.expect_seq and getChecksum(data) == checksum:
                    self.expect_seq = (self.expect_seq + 1) % 256
                    ack_pkt = self.make_pkt(seq_num, seq_num)
                    self.udp_send(ack_pkt)
                    if flag:    # 最后一个数据包
```

```
                        return data, True
                    else:
                        return data, False
                else:
                    ack_pkt = self.make_pkt((self.expect_seq - 1) % 256, self.expect_seq)
                    self.udp_send(ack_pkt)
                    return bytes('', encoding='utf-8'), False

            except socket.timeout:
                return bytes('', encoding='utf-8'), False

    def analyse_pkt(self, pkt):
        '''
        分析数据包
        '''
        # if len(pkt) < 4:
        # print 'Invalid Packet'
        # return False
        seq_num = pkt[0]
        flag = pkt[1]
        checksum = pkt[2]
        data = pkt[3:]
        print(seq_num, flag, checksum, data)
        return seq_num, flag, checksum, data

    def make_pkt(self, ackSeq, expectSeq):
        """
        创建 ACK 确认报文
        """
        return struct.pack('BB', ackSeq, expectSeq)
```

2. gbn_client.py

```
import os
import socket
import gbn



HOST = '127.0.0.1'
PORT = 8888
ADDR = (HOST, PORT)
CLIENT_DIR = os.path.dirname(__file__) + '/client'



senderSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
sender = gbn.GBNSender(senderSocket, ADDR)


fp = open(CLIENT_DIR + '/data.jpg', 'rb')

dataList = []
while True:
    data = fp.read(2048)
    if len(data) <= 0:
        break
    dataList.append(data)
print('The total number of data packets: ', len(dataList))

pointer = 0
while True:
    while sender.next_seq < (sender.send_base + sender.window_size):
        if pointer >= len(dataList):
            break
        # 发送窗口未被占满
        data = dataList[pointer]
        checksum = gbn.getChecksum(data)
        if pointer < len(dataList) - 1:
            sender.packets[sender.next_seq] = sender.make_pkt(sender.next_seq, data,
checksum,
                                                                              stop=False)
        else:
            sender.packets[sender.next_seq] = sender.make_pkt(sender.next_seq, data,
checksum,
                                                                              stop=True)
        print('Sender send packet:', pointer)
        sender.udp_send(sender.packets[sender.next_seq])
        sender.next_seq = (sender.next_seq + 1) % 256
        pointer += 1
    flag = sender.wait_ack()
    if pointer >= len(dataList):
        break

fp.close()
```

3. gbn_server.py

```
import os
import socket
import time

```

```python
import gbn


HOST = ''
PORT = 8888
ADDR = (HOST, PORT)
SERVER_DIR = os.path.dirname(__file__) + '/server'


receiverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
receiverSocket.bind(ADDR)
receiver = gbn.GBNReceiver(receiverSocket)


fp = open(SERVER_DIR + '/' + str(int(time.time())) + '.jpg', 'ab')
reset = False
while True:
    data, reset = receiver.wait_data()
    print('Data length:', len(data))
    fp.write(data)
    if reset:
        receiver.expect_seq = 0
        fp.close()
        break
```

4. gbn_biconnect.py

```python
import os
import socket
import threading
import time

import gbn


CLIENT_SEND_HOST = '127.0.0.1'
CLIENT_SEND_PORT = 8888
CLIENT_SEND_ADDR = (CLIENT_SEND_HOST, CLIENT_SEND_PORT)
CLIENT_RECV_HOST = '127.0.0.1'
CLIENT_RECV_PORT = 8989
CLIENT_RECV_ADDR = (CLIENT_RECV_HOST, CLIENT_RECV_PORT)
CLIENT_DIR = os.path.dirname(__file__) + '/client'

SERVER_SEND_HOST = '127.0.0.1'
SERVER_SEND_PORT = 8989
```

```
SERVER_SEND_ADDR = (SERVER_SEND_HOST, SERVER_SEND_PORT)
SERVER_RECV_HOST = '127.0.0.1'
SERVER_RECV_PORT = 8888
SERVER_RECV_ADDR = (SERVER_RECV_HOST, SERVER_RECV_PORT)
SERVER_DIR = os.path.dirname(__file__) + '/server'


def send(sender, directory):
    fp = open(directory + '/data.jpg', 'rb')

    dataList = []
    while True:
        data = fp.read(2048)
        if len(data) <= 0:
            break
        dataList.append(data)
    print('The total number of data packets: ', len(dataList))

    pointer = 0
    while True:
        while sender.next_seq < (sender.send_base + sender.window_size):
            if pointer >= len(dataList):
                break
            # 发送窗口未被占满
            data = dataList[pointer]
            checksum = gbn.getChecksum(data)
            if pointer < len(dataList) - 1:
                sender.packets[sender.next_seq] = sender.make_pkt(sender.next_seq, data,
checksum,
                                                            stop=False)
            else:
                sender.packets[sender.next_seq] = sender.make_pkt(sender.next_seq, data,
checksum,
                                                            stop=True)
            print('Sender send packet:', pointer)
            sender.udp_send(sender.packets[sender.next_seq])
            sender.next_seq = (sender.next_seq + 1) % 256
            pointer += 1
        flag = sender.wait_ack()
        if pointer >= len(dataList):
            break

    fp.close()
```

```python
def receive(receiver, directory):
    fp = open(directory + '/' + str(int(time.time())) + '.jpg', 'ab')
    reset = False
    while True:
        data, reset = receiver.wait_data()
        print('Data length:', len(data))
        fp.write(data)
        if reset:
            receiver.expect_seq = 0
            fp.close()
            break


clientReceiverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
clientReceiverSocket.bind(CLIENT_RECV_ADDR)
clientReceiver = gbn.GBNReceiver(clientReceiverSocket)
thread1 = threading.Thread(target=receive, args=(clientReceiver, CLIENT_DIR))
thread1.start()

serverReceiverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
serverReceiverSocket.bind(SERVER_RECV_ADDR)
serverReceiver = gbn.GBNReceiver(serverReceiverSocket)
thread2 = threading.Thread(target=receive, args=(serverReceiver, SERVER_DIR))
thread2.start()


clientSenderSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
clientSender = gbn.GBNSender(clientSenderSocket, CLIENT_SEND_ADDR)

serverSenderSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
serverSender = gbn.GBNSender(serverSenderSocket, SERVER_SEND_ADDR)

_ = input('Press key to continue:')

send(clientSender, CLIENT_DIR)
send(serverSender, SERVER_DIR)
```

5. sr.py

```python
import random
import socket
import struct
import time
```

```
BUFFER_SIZE = 4096
TIMEOUT = 10
WINDOW_SIZE = 4
LOSS_RATE = 0


def getChecksum(data):
    """
    char_checksum 按字节计算校验和。每个字节被翻译为无符号整数
    @param data: 字节串
    """
    length = len(str(data))
    checksum = 0
    for i in range(0, length):
        checksum += int.from_bytes(bytes(str(data)[i], encoding='utf-8'),
byteorder='little', signed=False)
        checksum &= 0xFF  # 强制截断

    return checksum


class SRSender:
    def __init__(self, senderSocket, address, timeout=TIMEOUT,
                 windowSize=WINDOW_SIZE, lossRate=LOSS_RATE):
        self.sender_socket = senderSocket
        self.timeout = timeout
        self.address = address
        self.window_size = windowSize
        self.loss_rate = lossRate
        self.send_base = 0
        self.next_seq = 0
        self.packets = [None] * 256
        self.acks = [False] * 256

    def udp_send(self, pkt):
        if self.loss_rate == 0 or random.randint(0, int(1 / self.loss_rate)) != 1:
            self.sender_socket.sendto(pkt, self.address)
        else:
            print('Packet lost.')
        time.sleep(0.3)

    def wait_ack(self):
        self.sender_socket.settimeout(self.timeout)
        count = 0
        while True:
            if count >= 10:
                # 连续超时 10 次，接收方已断开，终止
                break
            try:
                data, address = self.sender_socket.recvfrom(BUFFER_SIZE)

                ack_seq, expect_seq = self.analyse_pkt(data)
                print('Sender receive ACK:', ack_seq, expect_seq)

                if ack_seq in range(self.send_base, self.send_base + self.window_size):
                    self.acks[ack_seq] = True

                if ack_seq == self.send_base:
                    # 滑动窗口
                    while self.acks[self.send_base]:
                        self.send_base = (self.send_base + 1) % 256
                        # 新滑动进来的窗口单元需要初始化
                        self.acks[self.send_base + self.window_size] = False

                if self.send_base == self.next_seq:  # 已发送分组确认完毕
                    self.sender_socket.settimeout(None)
                    return True

            except socket.timeout:
                # 超时，重发分组.
                print('Sender wait for ACK timeout.')
                for i in range(self.send_base, self.next_seq):
                    if not self.acks[i]:    # 只重发未确认的分组
                        print('Sender resend packet:', i)
                        self.udp_send(self.packets[i])
                self.sender_socket.settimeout(self.timeout)  # reset timer
                count += 1
        return False

    def make_pkt(self, seqNum, data, checksum, stop=False):
        """
        将数据打包
        """
        flag = 1 if stop else 0
        return struct.pack('BBB', seqNum, flag, checksum) + data
    def analyse_pkt(self, pkt):
        """
        分析数据包
        """
        ack_seq = pkt[0]
        expect_seq = pkt[1]
        return ack_seq, expect_seq


class SRReceiver:
    def __init__(self, receiverSocket, timeout=10, windowSize=WINDOW_SIZE, lossRate=0):
        self.receiver_socket = receiverSocket
        self.timeout = timeout
        self.window_size = windowSize
        self.loss_rate = lossRate
        self.recv_base = 0
        self.recvs = [None] * 256
        self.target = None

    def udp_send(self, pkt):
        if self.loss_rate == 0 or random.randint(0, 1 / self.loss_rate) != 1:
            self.receiver_socket.sendto(pkt, self.target)
            print('Receiver send ACK:', pkt[0], pkt[1])
        else:
            print('Receiver send ACK:', pkt[0], pkt[1], ', but lost.')

    def wait_data(self):
        """
        接收方等待接受数据包
        """
        self.receiver_socket.settimeout(self.timeout)

        while True:
            try:
                data, address = self.receiver_socket.recvfrom(BUFFER_SIZE)
                self.target = address

                seq_num, flag, checksum, data = self.analyse_pkt(data)
                print('Receiver receive packet:', seq_num)
                # 收到期望数据包且未出错
                if seq_num in range(self.recv_base, self.recv_base + self.window_size)
\
                        and getChecksum(data) == checksum:
                    # 写入缓存，返回 ACK
                    self.recvs[seq_num] = data
                    ack_pkt = self.make_pkt(seq_num, seq_num)
                    self.udp_send(ack_pkt)

                    while self.recvs[self.recv_base] is not None:
                        # 滑动窗口并递交数据
                        self.recv_base = (self.recv_base + 1) % 256
                        self.recvs[self.recv_base + self.window_size] = None  # 新划入的单元要初始化

                    if flag:    # 最后一个数据包
                        return data, True
                    else:
                        return data, False
                else:
                    # 只返回 ACK，不缓存
                    ack_pkt = self.make_pkt(seq_num, seq_num)
                    self.udp_send(ack_pkt)

            except socket.timeout:
                return bytes('', encoding='utf-8'), False

    def analyse_pkt(self, pkt):
        '''
        分析数据包
        '''
        # if len(pkt) < 4:
        # print 'Invalid Packet'
        # return False
        seq_num = pkt[0]
        flag = pkt[1]
        checksum = pkt[2]
        data = pkt[3:]
        print(seq_num, flag, checksum, data)
        return seq_num, flag, checksum, data

    def make_pkt(self, ackSeq, expectSeq):
        """
        创建 ACK 确认报文
        """
        return struct.pack('BB', ackSeq, expectSeq)
```

6. sr_client.py

```python
import os
import socket

import sr


HOST = '127.0.0.1'
PORT = 8888
ADDR = (HOST, PORT)
CLIENT_DIR = os.path.dirname(__file__) + '/client'


senderSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sender = sr.SRSender(senderSocket, ADDR)


fp = open(CLIENT_DIR + '/data.jpg', 'rb')

dataList = []
while True:
    data = fp.read(2048)
    if len(data) <= 0:
        break
    dataList.append(data)
print('The total number of data packets: ', len(dataList))

pointer = 0
while True:
    while sender.next_seq < (sender.send_base + sender.window_size):
        if pointer >= len(dataList):
            break
        # 发送窗口未被占满
        data = dataList[pointer]
        checksum = sr.getChecksum(data)
        if pointer < len(dataList) - 1:
            sender.packets[sender.next_seq] = sender.make_pkt(sender.next_seq, data, checksum,
                                                               stop=False)
        else:
            sender.packets[sender.next_seq] = sender.make_pkt(sender.next_seq, data, checksum,
                                                               stop=True)
        print('Sender send packet:', pointer)
        sender.udp_send(sender.packets[sender.next_seq])
        sender.next_seq = (sender.next_seq + 1) % 256
        pointer += 1
    flag = sender.wait_ack()
    if pointer >= len(dataList):
        break

fp.close()
```

7. sr_server.py

```
import os
import socket
import time

import sr


HOST = ''
PORT = 8888
ADDR = (HOST, PORT)
SERVER_DIR = os.path.dirname(__file__) + '/server'


receiverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
receiverSocket.bind(ADDR)
receiver = sr.SRReceiver(receiverSocket)


fp = open(SERVER_DIR + '/' + str(int(time.time())) + '.jpg', 'ab')
reset = False
while True:
    data, reset = receiver.wait_data()
    print('Data length:', len(data))
    fp.write(data)
    if reset:
        receiver.recv_base = 0
        receiver.recvs = [None] * 256
        receiver.acks = [False] * 256
        fp.close()
        break
```