哈爾濱工業大學
Harbin Institute of Technology

计算机网络
课程实验报告

实验名称	可靠数据传输协议-GBN 协议的设计与实现				
姓名	姚舜宇		院系	计算学部	
班级	1903602		学号	1190202107	
任课教师	李全龙		指导教师	李全龙	
实验地点	格物 207		实验时间	2021.11.6	
实验课表现	出勤、表现得分(10)		实验报告得分(40)	实验总分	
	操作结果得分(50)				
教师评语					

计算机科学与技术学院 SINCE 1956...
School of Computer Science and Technology

实验目的：

理解可靠数据传输和华东窗口协议的基本原理，掌握停等协议、GBN 协议、SR 协议的工作原理，掌握基于 UDP 设计并实现 GBN 和 SR 协议的过程与技术

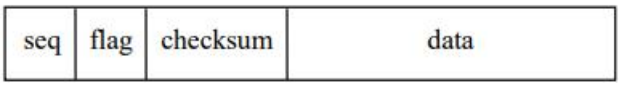
实验内容：

1. 基于UDP设计一个简单的GBN协议，实现单向可靠数据传输
2. 模拟引入数据包的丢失，验证所设计协议的有效性
3. 将GBN协议改进成支持双向数据传输
4. 将GBN协议改进为SR协议

实验过程：

一．GBN协议的实现

数据分组pkt格式：



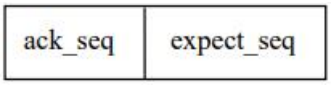
Seq：序列号

Flag：=1表示是最后一个分组，=0表示不是最后一个

Checksum：校验和

Data：传输的数据

确认分组ack格式：



Ack_seq：最近一次确认的数据分组的序列号

Expect_seq：接收端期望收到的确认分组的序列号

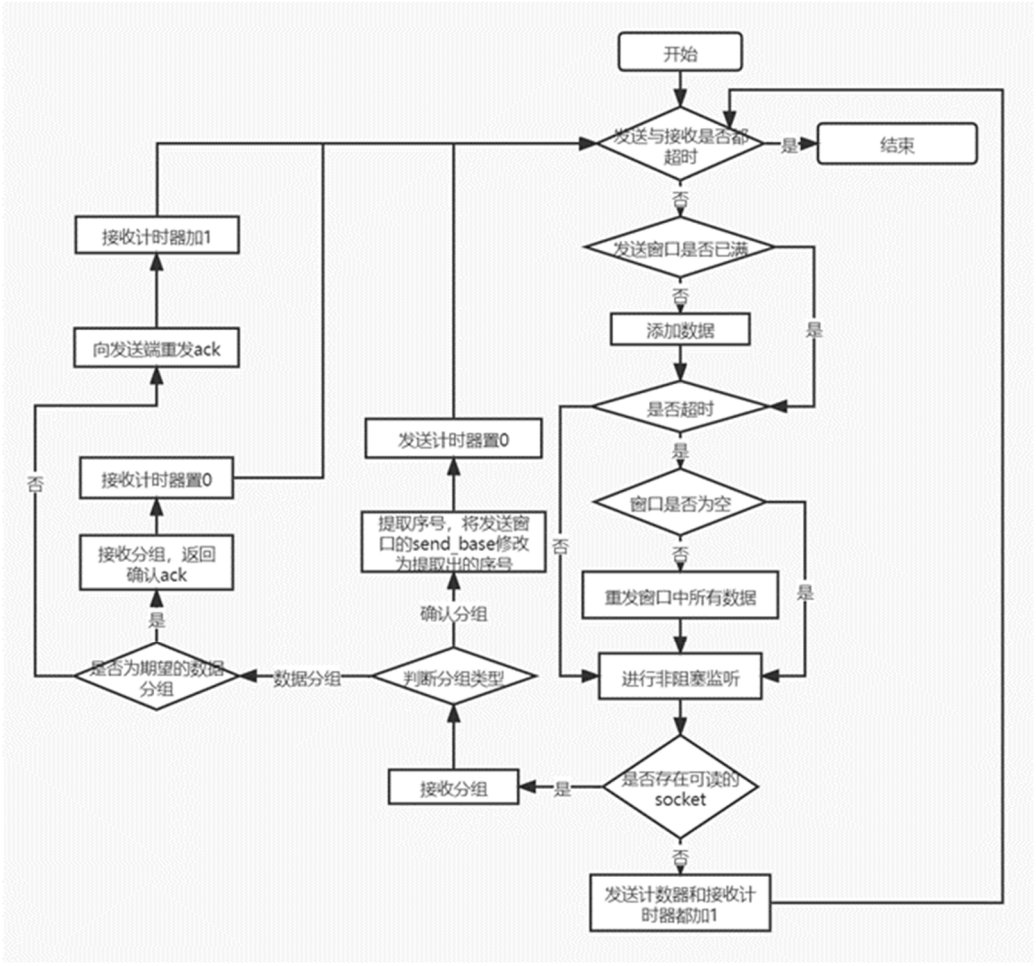
GBN协议的流程

GBN协议中，发送方在发完一个数据包之后，继续发送若干个数据包，即使在连续发送过程中收到了接收方发来的确认消息，也可以继续发送。发送方在每发送完一个数据包都会设置超时定时器，如果在规定时间内未收到确认消息，则重发对应的数据包。接收方只能按照顺序接收数据包。如果收到乱序到达的数据包，则直接丢弃。

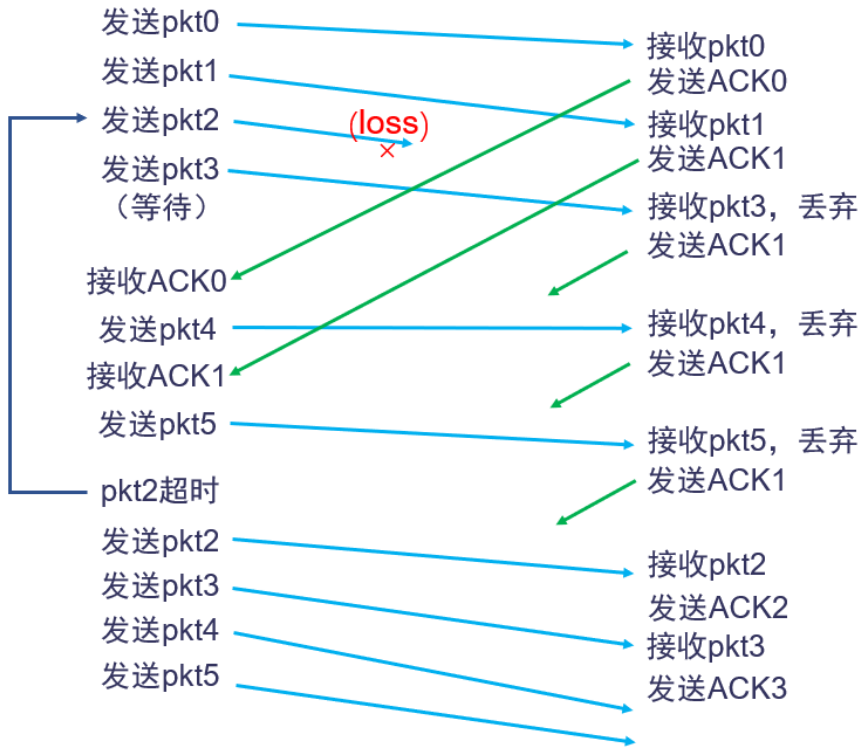
GBN发送方：发送方必须检查发送窗口是否已满，若满，会延迟一段时间后发送。若未满，则直接发送数据。累计确认：指的是当对n号数据包确认之后，代表n号之前的所有数据包都得到了确认。超时事件：没有收到的数据包以及没收到的数据包之后的数据包会被重传。

GBN接收方：如果正确收到n号数据包，并且其之前的数据包都按序到达，那么只发送一个n号确认消息给发送方。如果有数据包丢失，其之后的数据包会直接丢弃，发送丢弃数据包之前的确认消息给发送方，等待发送方重传。

GBN协议的流程图如下：



GBN协议的交互过程示例：



二．SR协议的实现

数据分组pkt格式：

seq	flag	checksum	data

Seq：序列号

Flag：=1表示是最后一个分组，=0表示不是最后一个

Checksum：校验和

Data：传输的数据

确认分组ack格式：

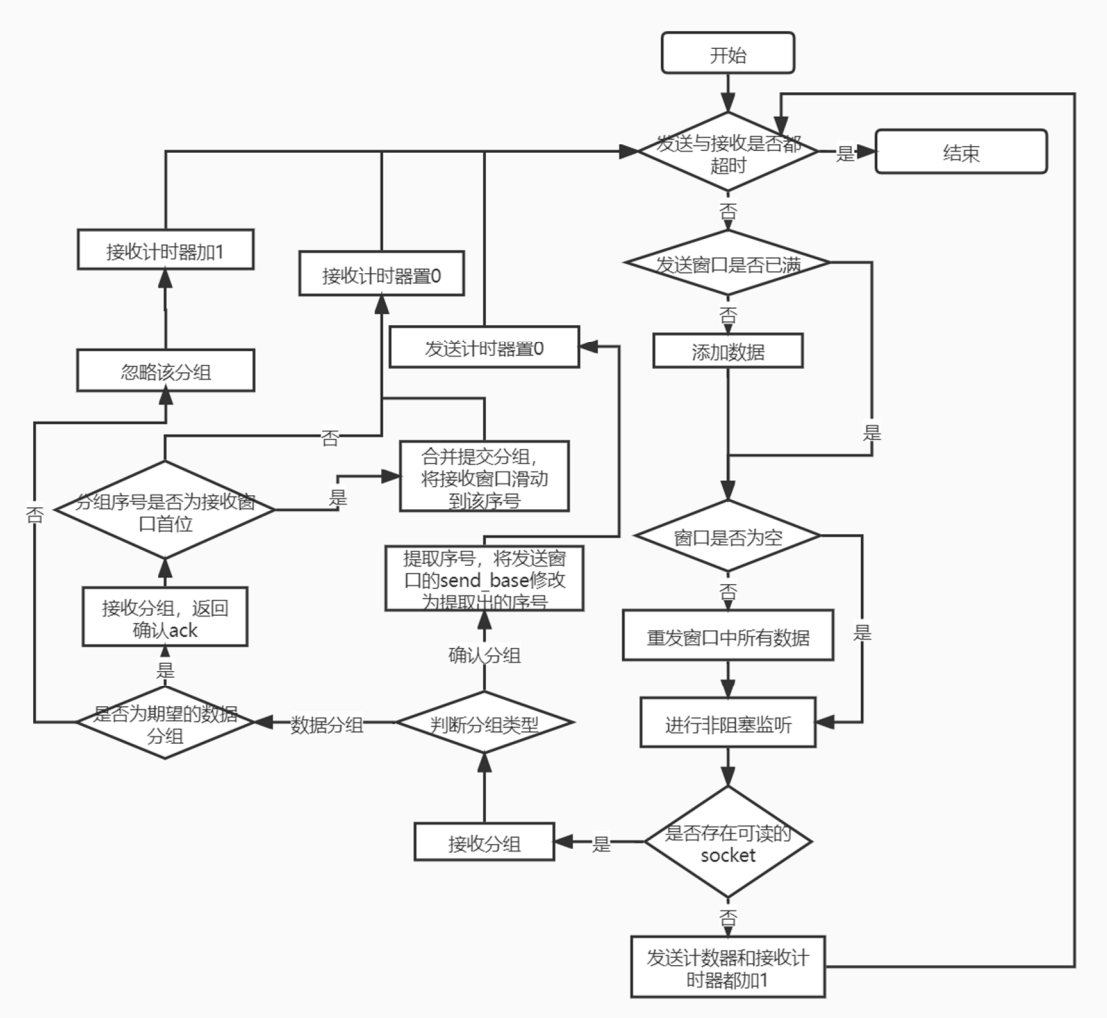
ack_seq	expect_seq

Ack_seq：最近一次确认的数据分组的序列号

Expect_seq：接收端期望收到的确认分组的序列号

SR协议和GBN协议的主要区别在于接收方有缓存。当接收方发现某个数据包错误后，其后继续送来的正确的数据包不能直接接收，只能缓存下来，同时要求重传错误的数据包。一旦收到重传的数据包，则和缓冲区的其余数据包一同按照正确顺序递交到高层。所以，SR协议和GBN协议相比减少了浪费，但对缓冲区有了一定的要求。

SR协议的流程图如下：

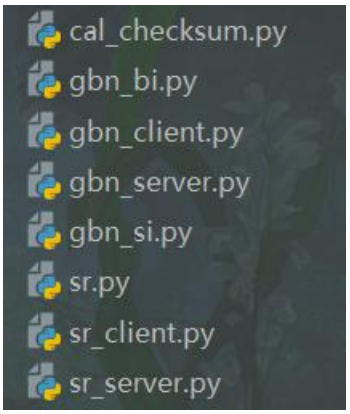


SR协议的交互过程示例：

三．模拟丢包过程

设置一个丢包率r，每次发送分组之前，产生一个随机数，如果满足丢包概率的条件，则不发送分组并输出提示信息，否则正常发送分组。

四．程序介绍



Gbn_si.py是GBN协议单向传输的核心实现。有两个类。

类1：gbn_sender 表示gbn的发送方。

属性有：

Sender_socket：发送方socket

Time_out：超时时间

Address：IP地址和端口号

Size：窗口大小

Rate：丢包率

Send_base：窗口头部序号

Next_seq：下一个可用序列号

Pkt：数据分组

方法有：

Send_pkt：发送数据包

Wait_for_ack：等待确认消息

Make_pkt：制作数据包
Get_ack_info：获得确认消息的信息
类2：gbn_receiver 表示gbn的接收方。
属性有：
Receiver_socket：接收方socket
Time_out：超时时间
Rate：丢包率
Expect_seq：期待收到的分组序列号
Target：确认分组的发送目标地址
方法有：
Send_ack：发送确认消息
Wait_for_data：等待数据包
Make_ack：制作确认消息
Get_pkt_info：获得数据包的信息

sr.py是SR协议单向传输的核心实现。有两个类。
类1：sr_sender 表示sr的发送方。
属性有：
Sender_socket：发送方socket
Time_out：超时时间
Address：IP地址和端口号
Size：窗口大小
Rate：丢包率
Send_base：窗口头部序号
Next_seq：下一个可用序列号
Pkt：数据分组
Ack：确认消息分组
方法有：
Send_pkt：发送数据包
Wait_for_ack：等待确认消息
Make_pkt：制作数据包
Get_ack_info：获得确认消息的信息
类2：sr_receiver 表示sr的接收方。
属性有：
Receiver_socket：接收方socket
Time_out：超时时间
Size：窗口大小
Rate：丢包率
Recv_base：接收窗口头部序列号
Recvs：收到的数据分组
Target：确认分组的发送目标地址
方法有：
Send_ack：发送确认消息
Wait_for_data：等待数据包

Make_ack：制作确认消息

Get_pkt_info：获得数据包的信息

Gbn_client.py和gbn_server.py分别是使用GBN协议CS架构的客户端和服务器端，用于实验GBN协议的准确性。Sr_client.py和sr_server.py分别是使用SR协议CS架构的客户端和服务器，用于验证SR协议的准确性。Gbn_bi.py是利用单向GBN协议实现的双向GBN协议进行传输的函数。

实验结果：

1. 使用GBN单向传输

目标将client文件夹下的图片data.jpg通过GBN协议传输到server文件夹下。

在客户端程序中，将数据分组，批量发送之后等待确认消息。

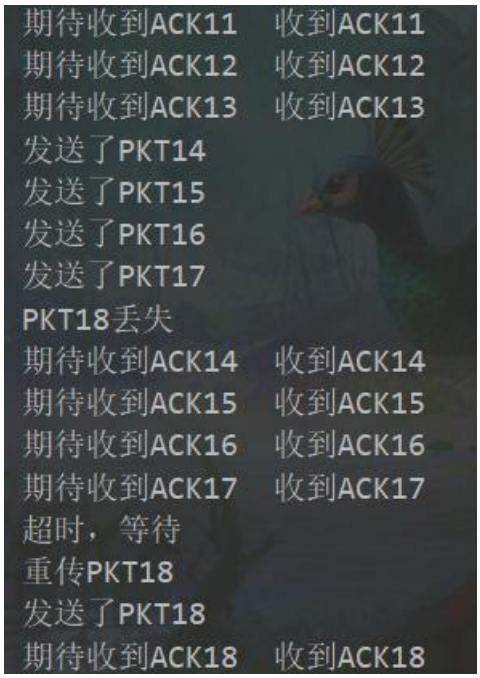
```
data_list = []  # 存储数据
file = open('./client/data.jpg', 'rb')

while True:
    d = file.read(2048)
    if len(d) <= 0:
        break
    data_list.append(d)
file.close()
print('PKT的总数：' + str(len(data_list)))
```

```
index = 0
while True:
    while sender.next_seq < (sender.send_base + sender.size):
        if index >= len(data_list):
            break
        data = data_list[index]
        checksum = gbn_si.cal_checksum(data)
        if index < len(data_list) - 1:  # 不是最后一个数据包
            sender.pkt[sender.next_seq] = sender.make_pkt(sender.next_seq, data, checksum, False)
        else:  # 是最后一个数据包
            sender.pkt[sender.next_seq] = sender.make_pkt(sender.next_seq, data, checksum, True)
        sender.send_pkt(sender.pkt[sender.next_seq])  # 发送数据
        sender.next_seq = (sender.next_seq + 1) % 256
        index += 1
    sender.wait_for_ack()  # 每次发送数据后等待ack
    if index >= len(data_list):
        break
```

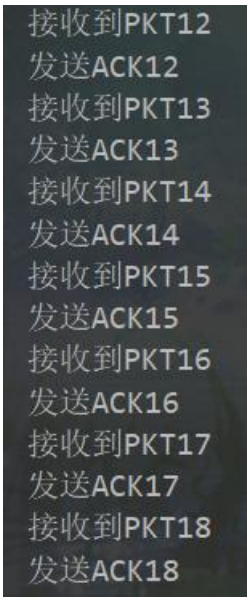
在服务器端程序中，等待客户端发送的数据并进行确认消息的发送。

```
receiver_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
receiver_socket.bind(address)
receiver = sr.sr_receiver(receiver_socket)

file = open('./server/sr_' + str(int(time.time())) + '.jpg', 'ab')
while True:
    data, reset = receiver.wait_for_data()  # 等待客户端发送的数据包
    file.write(data)
    if reset:  # 发送结束
        receiver.recv_base = 0
        receiver.recvs = [None] * 256
        receiver.ack = [False] * 256
        file.close()
        break
```

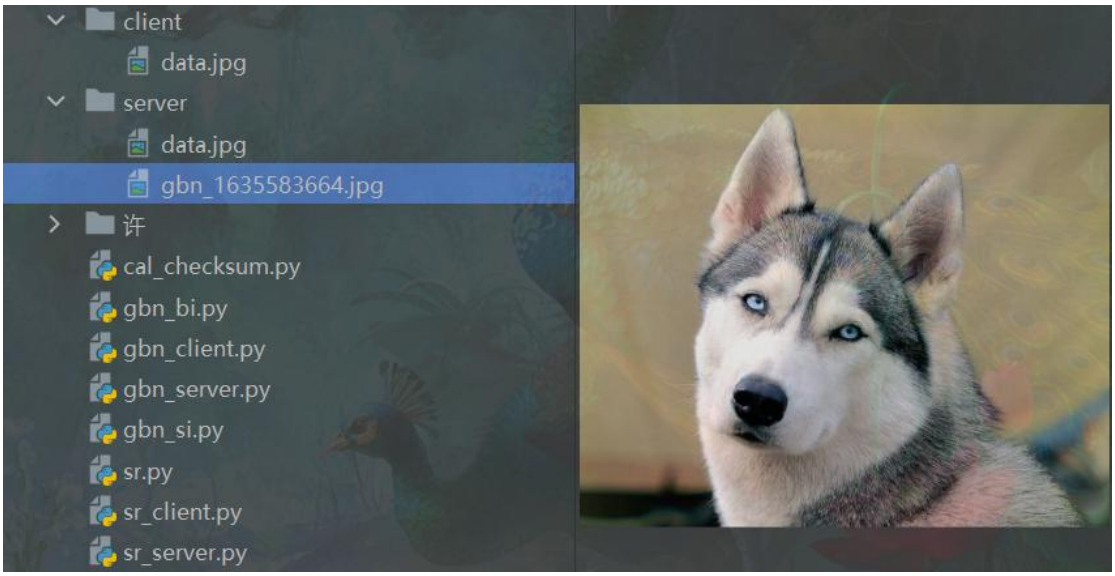
客户端运行过程：



服务器端运行过程：



传输结果：



命名根据时间戳，防止重复。

2. 使用GBN双向传输

双向传输的实现，使用了两个函数。Send函数用于发送数据包，需要传入一个gbn_sender对象。Receive函数用于接收数据包，需要传入一个gbn_receiver对象。

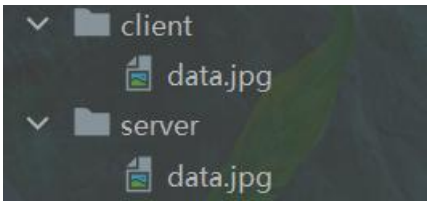
在实验中，需要同时测试客户端向服务器端传数据和服务器端向客户端传数据，所以开启了两个线程，分别是客户端接收数据的线程和服务器端接收数据的线程。

```
client_Receiver_Socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client_Receiver_Socket.bind(config_client['ADDRESS_RECEIVE'])
client_Receiver = gbn_si.gbn_receiver(client_Receiver_Socket)
# 客户端接收数据线程
thread1 = threading.Thread(target=receive, args=(client_Receiver, config_client['DIR']))
thread1.start()

server_Receiver_Socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_Receiver_Socket.bind(config_server['ADDRESS_RECEIVE'])
server_Receiver = gbn_si.gbn_receiver(server_Receiver_Socket)
# 服务器端接收数据线程
thread2 = threading.Thread(target=receive, args=(server_Receiver, config_server['DIR']))
thread2.start()
```

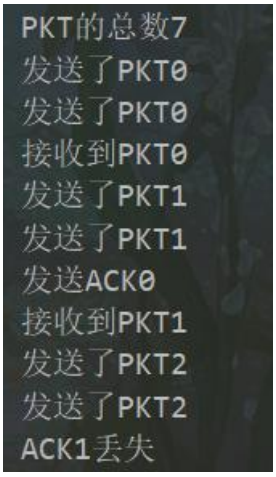
然后定义了客户端发送数据对象和服务器端发送数据对象，并调用send函数进行数据传输。

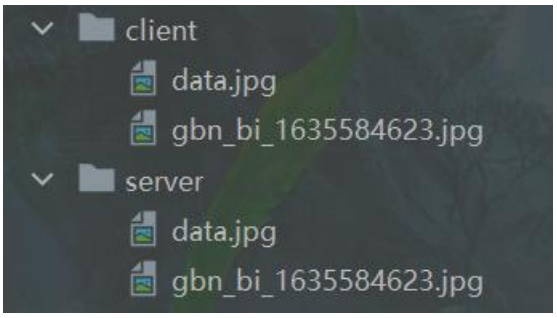
```
client_Sender_Socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# 客户端发送数据对象
client_Sender = gbn_si.gbn_sender(client_Sender_Socket, config_client['ADDRESS_SEND'])

server_Sender_Socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# 服务器端发送数据对象
server_Sender = gbn_si.gbn_sender(server_Sender_Socket, config_server['ADDRESS_SEND'])

send(client_Sender, config_client['DIR'])  # 客户端发送数据
send(server_Sender, config_server['DIR'])  # 服务器端发送数据
```

运行前目录如下：

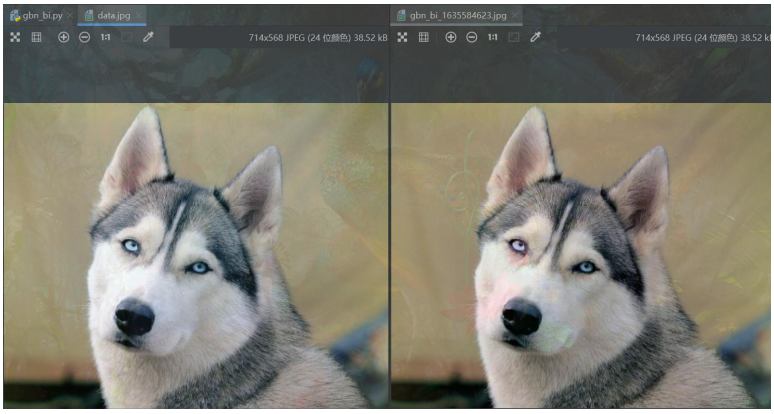


运行过程：

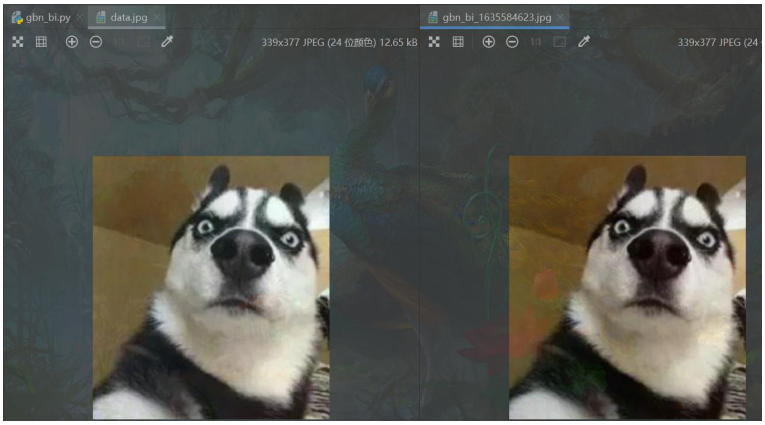




成功将client内的data.jpg传输到server内。



成功将server内的data.jpg传输到client内。

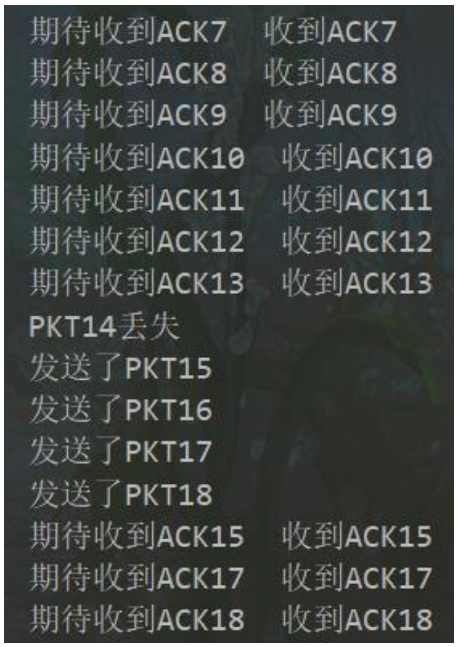


3. 使用SR传输

SR传输和GBN类似，也使用了客户端程序和服务器端程序。由于程序逻辑相似，在此不重复叙述。

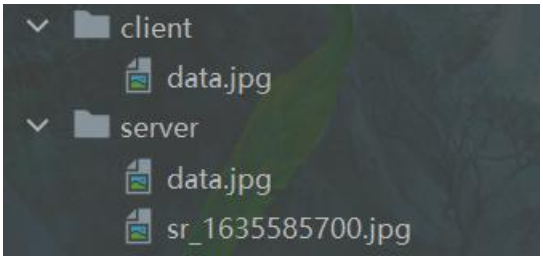
程序运行过程：

客户端：

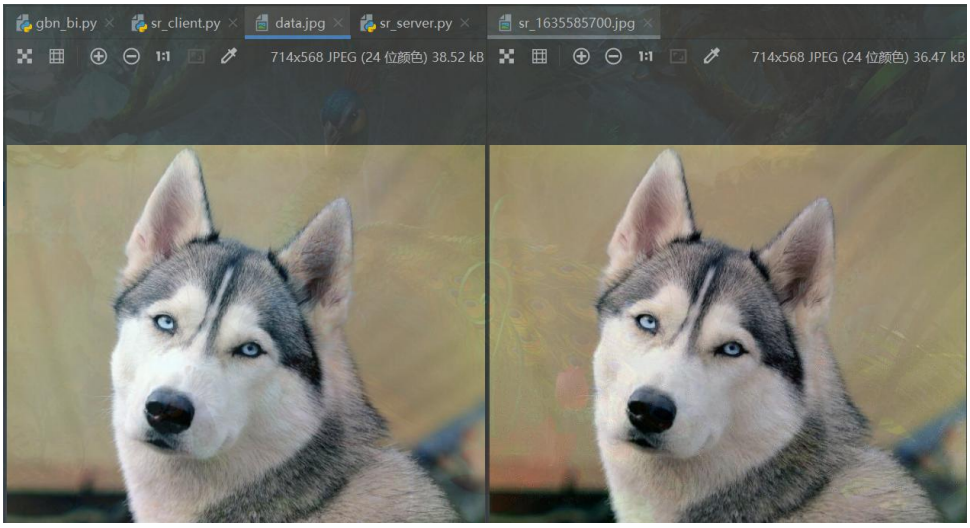


服务器端：

运行结果：



成功将client内的data.jpg传输到server内。



问题讨论：

首先关于丢包的引入，都是采用随机数的方法，但也有不同的模式，如生成一个0到1的随机数和概率进行比较，或生成一个1到1/r的向下取整的整数，看其是否为1。

对于双向数据传输，实际上就是此时服务器和客户端没有明确的区分，二者都可以实现文件的发送与接收。

心得体会：

通过本次实验，首先更加熟悉了python下的socket编程，其次是对于GBN协议和SR协议，理解更加深入了。这两种传输协议都是对停等协议的改良，停等协议只能一个一个的数据包进行传送，效率及其低下，中间有大量的时间都被浪费。通过引入重传的分组恢复机制，大大提高了数据传输效率。另外，SR协议也是GBN协议的改进，在接收方引入缓存，存储乱序到达的分组，能够减少重传的次数，提高传输效率。