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	王丙昊		院系	计算机科学与技术	
班级	1603108		学号	1160300302	
任课教师	聂兰顺		指导教师	聂兰顺	
实验地点	格物 207		实验时间	2018.11.7	
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：

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

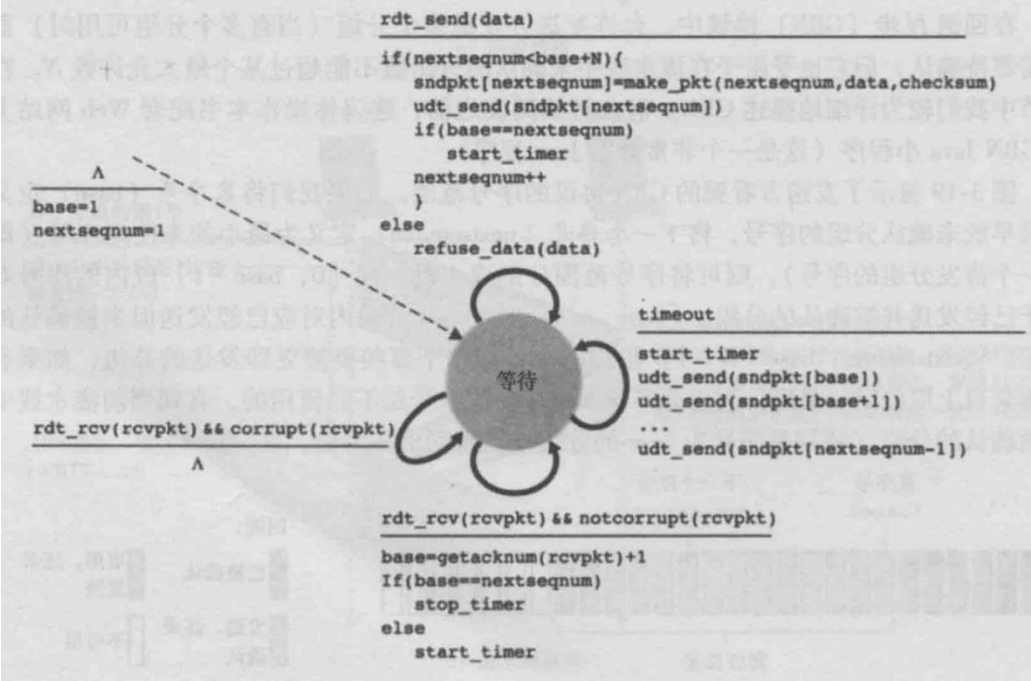
实验内容：

1) 基于UDP设计一个简单的GBN协议，实现单向可靠数据传输（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）
4) 将所设计的GBN协议改进为SR协议（选作内容）

实验过程：

1. GBN协议的实现

1）GBN发送方GBNSender



rdt_send():

从上层接收数据并发送给接收方：上层调用rdt_sent()时，发送方首先检查发送窗口是否已满。如果窗口未满，则产生一个分组并将其发送，并相应地更新变量；如果窗口已满，发送方不发送数据，并隐式地提示上层该窗口已满。

```
checksum = self.get_checksum(data_send)
if count < len(data) - 1:
    # 不是最后一个数据分组，则标志位为0
    self.buffer[self.nextseqnum] = struct.pack('BBB', self.nextseqnum, 0, checksum) + data_send
else:
    # 如果是最后一个数据分组，则标志位为1
    self.buffer[self.nextseqnum] = struct.pack('BBB', self.nextseqnum, 1, checksum) + data_send
print('发送端发送数据分组：', count)
# 使用udp传输数据
self.udp_send(self.buffer[self.nextseqnum])
# 序列号取值范围设为 0~255
self.nextseqnum = (self.nextseqnum + 1) % 256
```

wait_ack():

收到确认分组后进行的处理操作：GBN协议采用的是**累计确认**机制，如果wait_ack()接收到了接收方确认收到序号为n的分组，则表明接收方收到序号为n的以前且包括n在内的所有分组。

```
ack_seq = data_rcv[0]       # 接收方最近一次确认的数据分组的序列号
expect_seq = data_rcv[1]    # 接收方期望收到的数据分组的序列号
print('发送端接收ACK:', "ack", ack_seq, "expect", expect_seq)
self.base = max(self.base, (ack_seq + 1) % 256)
# 已发送分组确认完毕，则停止定时器
if self.base == self.nextseqnum:
    self.sender_socket.settimeout(None)
    break
```

超时事件：

在GBN协议中，会维护一个定时器，如果出现超时现象，发送方重传所有已经发送但还未被确认过的分组。GBN中发送方仅使用一个定时器，它可以被当作最早的已发送但未被确认的分组所使用的定时器。如果超时次数超过一定程度，则终止发送

```
# 如果发生超时现象，重发所有未确认的分组
except socket.timeout:
    count += 1
    print('发送端等待ack超时')
    # 重发未确认的分组
    for i in range(self.base, self.nextseqnum):
        print('发送端重新发送数据分组：', i)
        self.udp_send(self.buffer[i])
    # 重新启动定时器
    self.sender_socket.settimeout(self.timeout)
# 如果多次超时，则终止
if count >= 8:
    break
```

get_checksum():

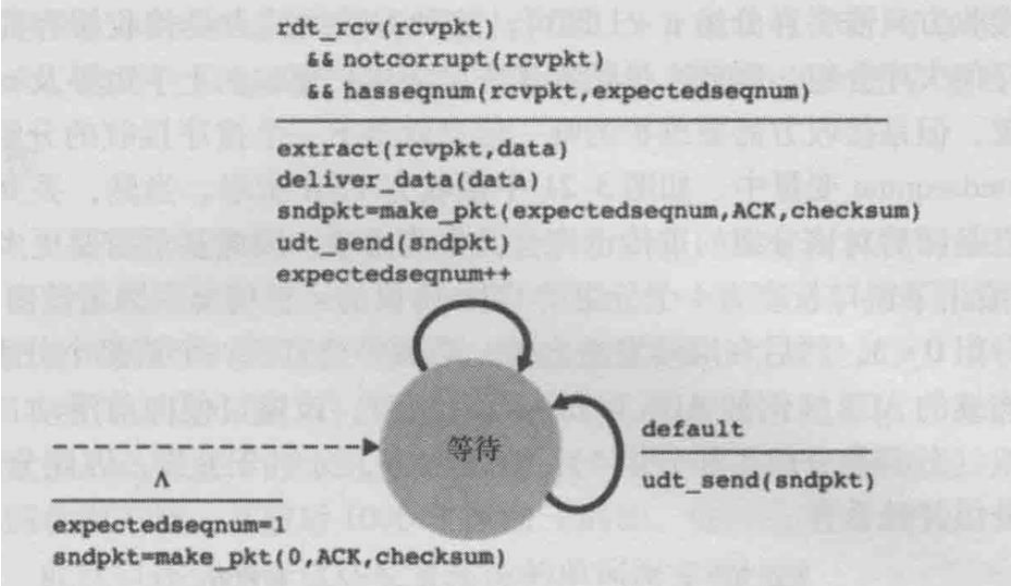
计算校验和

GBN发送方数据报文格式

seq	flag	checksum	data

seq为数据报文的序号
flag为标志位，用于标识该数据分组是否为最后一个数据分组
checksum校验和，用于接收方确认数据报文在传输过程中是否发生错误
data为传输的数据

2）GBN接收方GBNReceiver



wait_data():

GBN接收方收到数据分组后进行的操作：如果一个序号为n的分组被正确接收到，并且按序，则接收方为分组n发送一个ACK，并将该分组中的数据部分交付给上层。在其他情况下，丢弃该分组，为**最近**按序接收的分组重新发送ACK

```
# 如果收到期望数据包且未出错时，才会将数据交给接收方
if ack == self.expectseqnum and self.get_checksum(data_deliver) == checksum:
    self.expectseqnum = (self.expectseqnum + 1) % 256
    ack_pkt = struct.pack('BB', ack, self.expectseqnum)
    self.udp_send(ack_pkt)
    # 如果是最后一个数据分组，通告上层
    if flag:
        return data_deliver, True
    else:
        return data_deliver, False
# 如果有错的话，则再发送最近按序接收的分组的ACK
else:
    ack_pkt = struct.pack('BB', (self.expectseqnum - 1) % 256, self.expectseqnum)
    self.udp_send(ack_pkt)
```

3）GBN分组丢失模拟方法

GBN通过udp发送数据，实验过程中，设计数据报丢失率LOSS_RATE[0,1]，每次使用udp发送数据前，通过**生成一个0到1之间的随机数**，如果该数大于LOSS_RATA，则可以发送数据，否则不发送并提示发送失败。GBN接收方的确认报文的分组丢失采用同样的方法

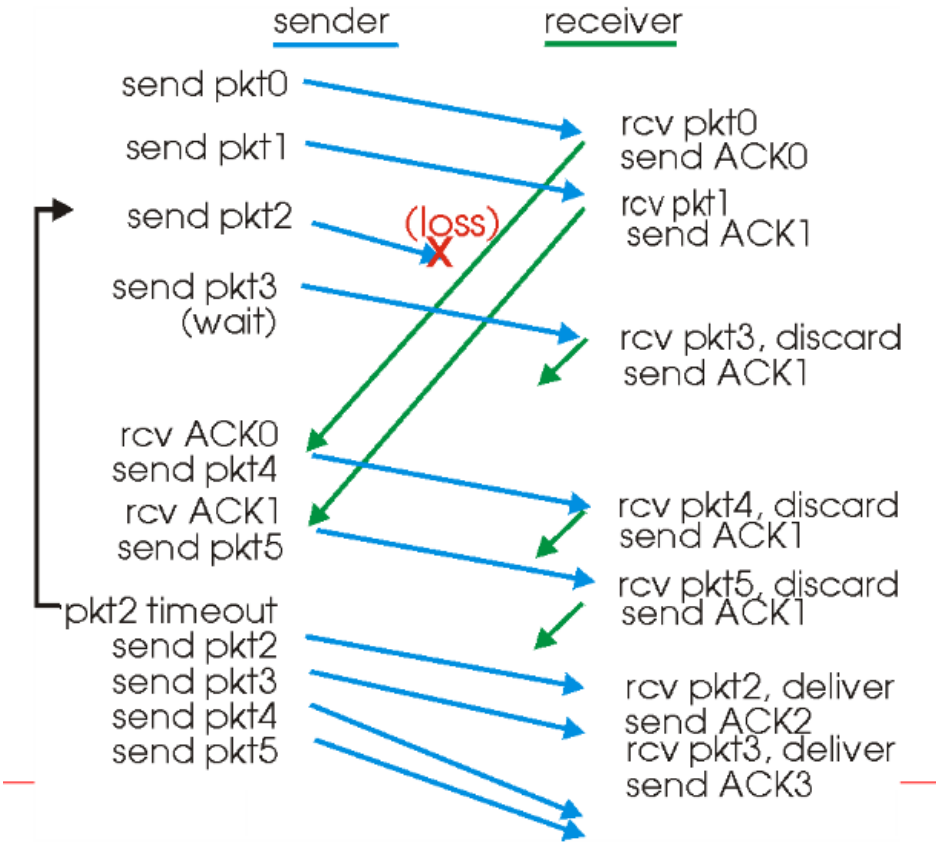
```
# 使用udp发送分组
def udp_send(self, data):
    if random.random() >= self.loss_rate:
        self.sender_socket.sendto(data, self.address)
    else:
        print('分组丢失')
    time.sleep(0.3)
```

4）GBN双向数据传输的实现

在客户端（client）与服务器端（server）分别建立一个GBN发送端和GBN接收端，使用两对绘画模拟双向数据传输

```
# client发送方
client_senderSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client_sender = gbn.GBNSender(client_senderSocket, '127.0.0.1', 8888)

# client接收方
client_receiverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client_receiverSocket.bind(('127.0.0.1', 8080))
client_receiver = gbn.GBNReceiver(client_receiverSocket)
threading.Thread(target=receive, args=(client_receiver, os.path.dirname(__file__) + '/client/')).start()

# server发送方
server_senderSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_sender = gbn.GBNSender(server_senderSocket, '127.0.0.1', 8080)

# server接收方
server_receiverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_receiverSocket.bind(('127.0.0.1', 8888))
server_receiver = gbn.GBNReceiver(server_receiverSocket)
threading.Thread(target=receive, args=(server_receiver, os.path.dirname(__file__) + '/server/')).start()

send(client_sender, os.path.dirname(__file__) + '/client/')
send(server_sender, os.path.dirname(__file__) + '/server/')
```

5）典型交互过程



窗口大小为4，发送方发送分组0~3，然后在继续发送之前，必须等待直到一个或多个分组被确认。当接收到每一个**连续的ACK时，该窗口便向前滑动**（累计确认）；在接收方，分组2丢失，因此分组3、4和5被发现是**失序分组并被丢弃**

2. SR协议的实现

1）SR发送方SRSender

rdt_sent():

从上层接收数据并发送给接收方，实现与GBN相同：从上层收到数据后，SR发送方检查下一个可用与该分组的序号。如果序号位于发送方的窗口内，则将数据打包。

wait_ack():

如果收到ack，倘若该分组序号在窗口内，则SR发送方将那个被确认的分组标记为已接收。

```
# 如果是已经发送但还未确认的分组，则记为确认收到ack
if ack_seq in range(self.base, self.base + self.window_size):
    self.acks[ack_seq] = True
```

如果该分组的序号等于self.base(窗口头部序列号)，则窗口基序号向前移动到具有最小序号的未确认分组处

```
# 滑动窗口，滑至从左到右第一个未确认的分组
if ack_seq == self.base:
    while self.acks[self.base]:
        self.base = (self.base + 1) % 256
        self.acks[self.base + self.window_size] = False
```

超时现象：

在SR协议中，每个分组有自己的逻辑定时器，但在本实验中，我采用的是用一个定时器来模拟多个逻辑定时器。当一个分组在超时时间内没有收到确认ACK，则只重传没有收到ACK的分组，并不重传其他已经收到确认ACK的分组

```
# 如果发生超时现象，只重传没收到ack的分组
except socket.timeout:
    count += 1
    print('发送端等待ack超时')
    # 只重传没收到ack的分组
    for i in range(self.base, self.nextseqnum):
        # 如果没有收到ack，则重传该分组
        if self.acks[i] is False:
            print('发送端重新发送数据分组：', i)
            self.udp_send(self.buffer[i])
    self.sender_socket.settimeout(self.timeout)
# 如果超时次数超过一定数目，则终止
if count >= 8:
    break
```

2）SR接收方SRReceiver

wait_data():

SR接收方确认一个正确接收的分组不管其是否按序。失序的分组将被缓存，直到所有丢失分组（序号更小的分组）都被收到，这时接收方将这些分组一并交给上层。

```
# 滑动窗口，滑至从左到右第一个没有收到数据的位置，并交给上层
while self.rcv_buffer[self.base] is not None:
    tmp = self.base
    self.base = (self.base + 1) % 256
    self.rcv_buffer[self.base + self.window_size] = None
    # 如果是最后一个数据分组，通告上层
    return self.rcv_buffer[tmp], False
```

SR接收方非常重要的一点是，如果收到了**重复ACK**，接收方会重新确认，而不是简单的忽略

```
# 如果有错的话，则重新确认那些序号小于当前窗口基序号的分组，此时不缓存
else:
    ack_pkt = struct.pack('B', ack)
    self.udp_send(ack_pkt)
```

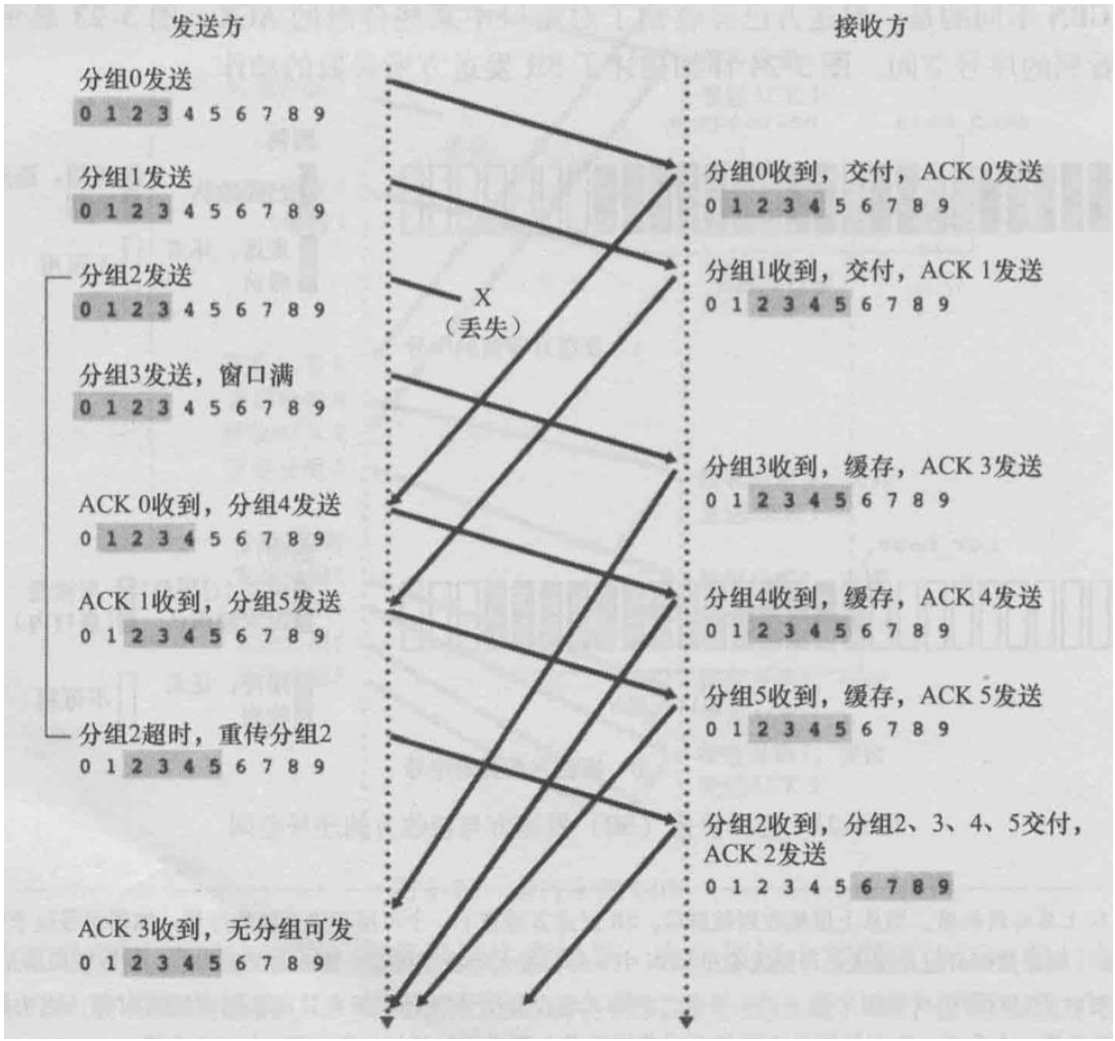
3）SR分组丢失模拟方法

与GBN相同，为SR的发送方和接收方设置LOSS_RATA，使用udp发送数据时，如果生成的随机数大于LOSS_RATA，则发送数据

4）SR双向数据传输的实现

与GBN相同

5）典型交互过程

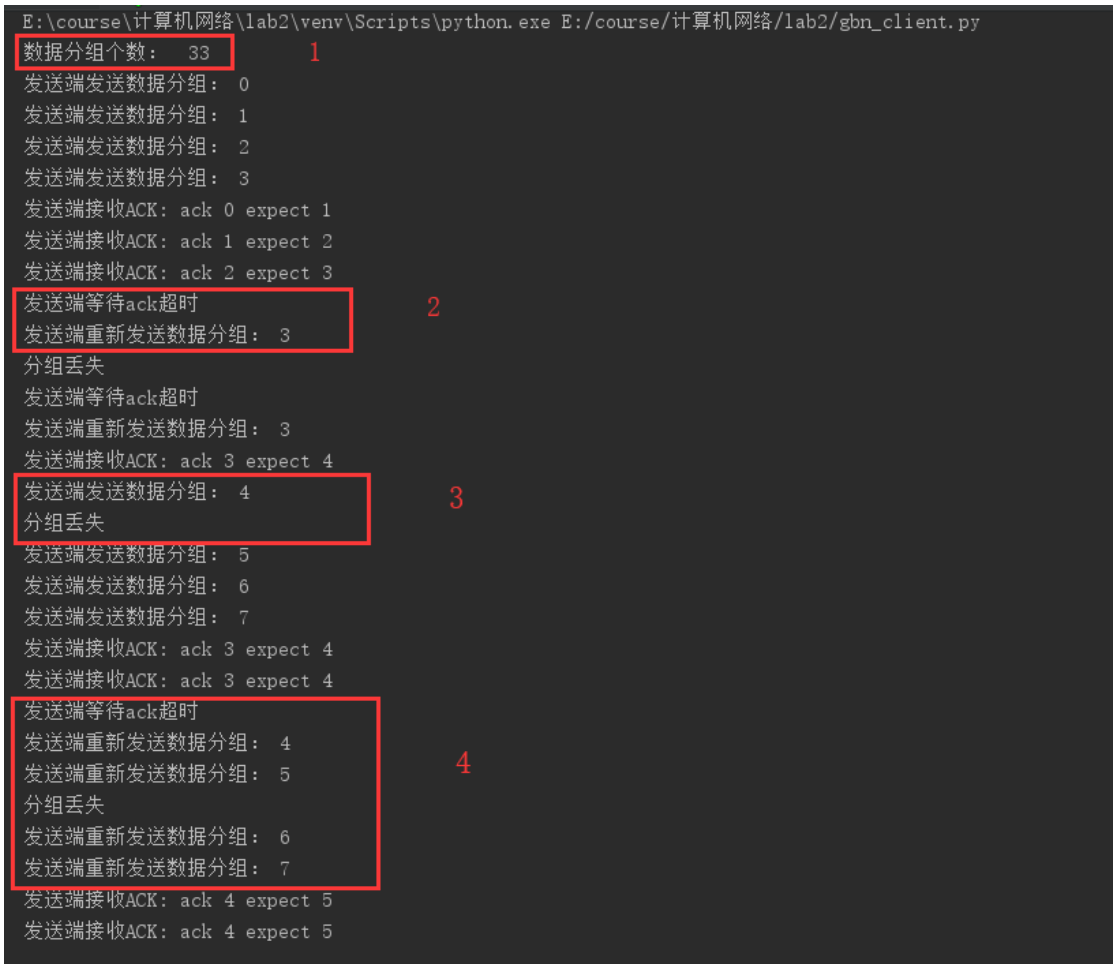


接收方收到乱序到达的分组时，缓存这些分组（如上图的分组3、4和5），并在最终收到分组2时，才将它们**一并交付给上层**。并且注意到接收方重新确认（而不是忽略）已收到过的那些序号小于当前窗口基序号的分组。

实验结果：

1） GBN协议测试

将GBN协议的发送方和接收方丢包概率设为0.1。首先指定接收方的IP地址（采用回环测试地址127.0.0.1）以及端口号，使其处于监听状态。然后运行GBN的发送方。发送方将下面的图片发送给接收方

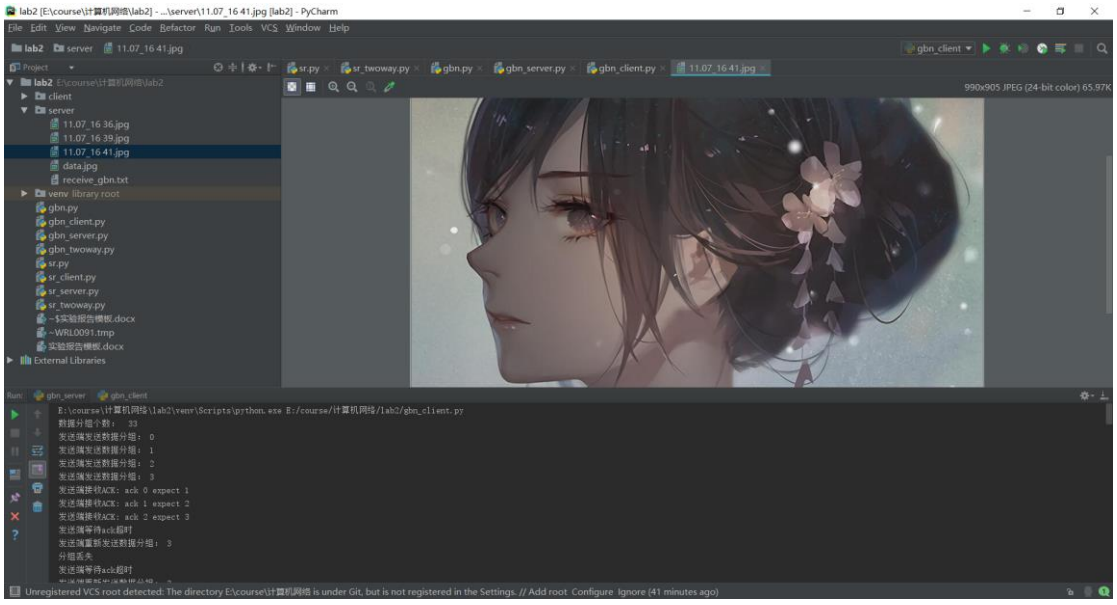


如图：

1. 发送方向接收方发送client/data.jpg，将文件分为了33个数据分组
2. 此处发生的是，接收方发出的确认报文丢失，导致接收方没有收到引起的超时，但是因为接收方之前已经收到了ack2，所以self.base移动到了3，所以只会重发数据分组3
3. 此处，发送方发送数据分组4时发生了数据分组丢失
4. 因为数据分组4丢失，即使分组5、6和7成功到达了接收方，但是会**因为乱序而**

丢弃，返还给发送方ack3。然后发送方会**重发窗口内的所有的数据分组**

传输结果：结果位于/server/时间.jgp

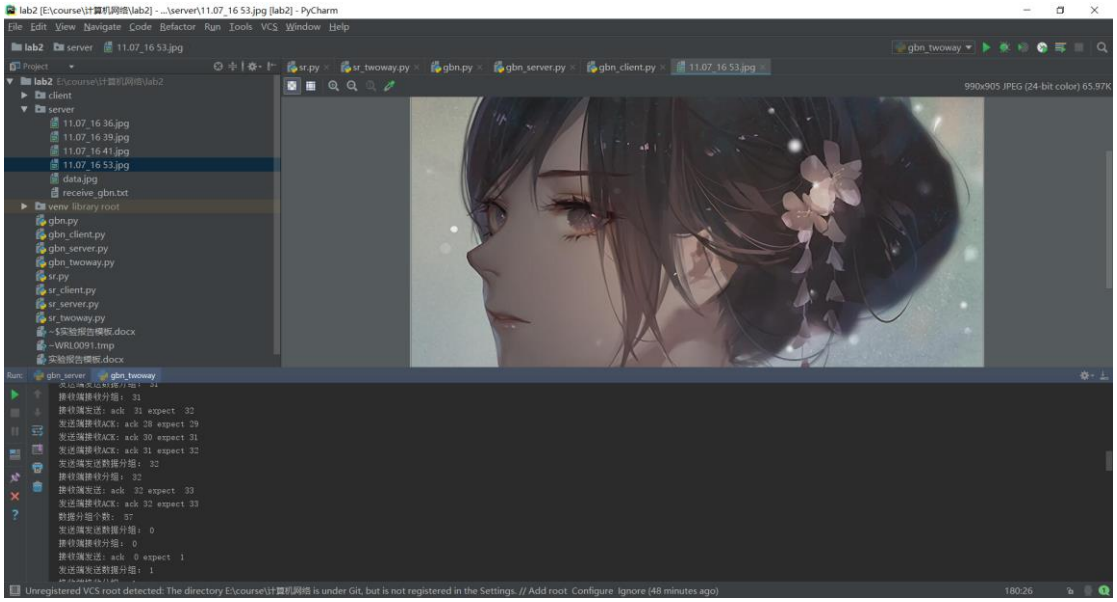


2） GBN协议双向数据传输测试

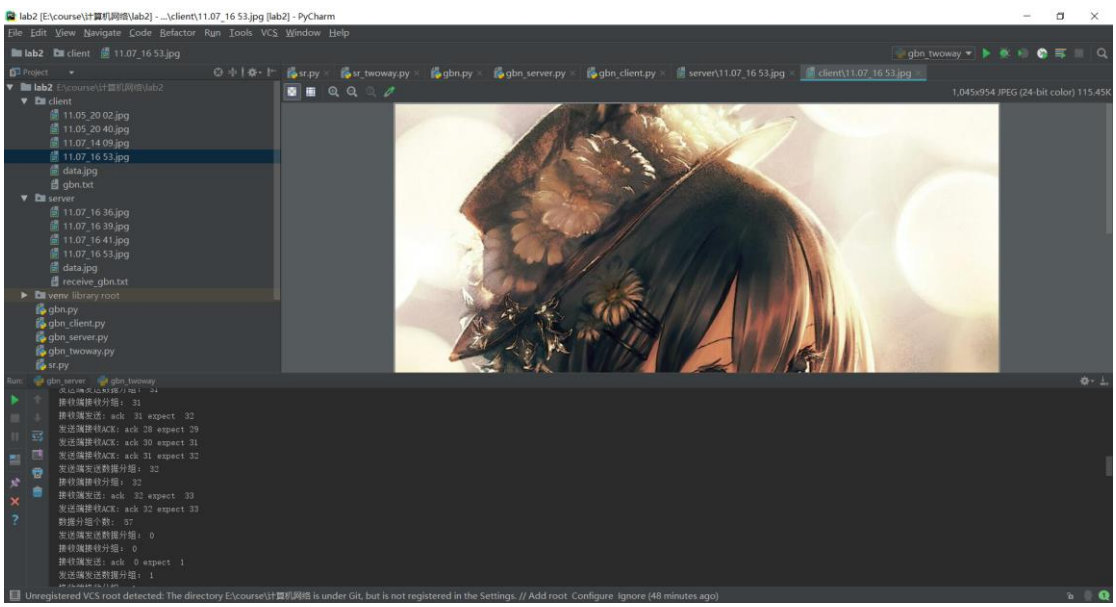
client向server发送/client/data.jgp（大小为33个数据分组），server向client发送/server/data.jgp（大小为57个数据分组），运行gbn_twoway.py

传输结果：分别位于/server文件夹以及/client文件夹中，都以传输时间.jgp命名

Client向server传输结果：



Server向client传输结果：



3） SR协议测试

将数据分组丢失概率设为0.1.

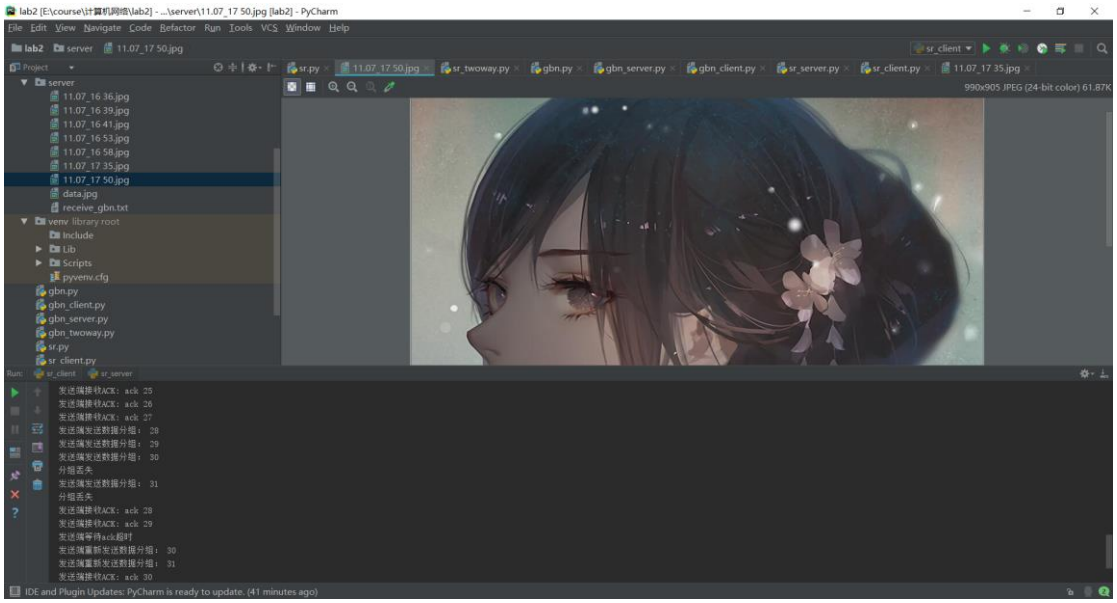
```
发送端发送数据分组： 4
发送端发送数据分组： 5
发送端发送数据分组： 6
分组丢失
发送端发送数据分组： 7
发送端接收ACK: ack 4
发送端接收ACK: ack 5
发送端接收ACK: ack 7
发送端等待ack超时
发送端重新发送数据分组： 6
发送端接收ACK: ack 6
发送端发送数据分组： 8
发送端发送数据分组： 9
发送端发送数据分组： 10
发送端发送数据分组： 11
```

发送方发送数据分组6失败，接收方收到了分组7（乱序），此时将其缓存起来而不是丢弃，并回复确认报文ack7。接收方接收确认报文ack6超时，此时接收方**只重发数据分组6。**

```
发送端发送数据分组： 20
发送端发送数据分组： 21
发送端发送数据分组： 22
发送端发送数据分组： 23
分组丢失
发送端接收ACK: ack 20
发送端接收ACK: ack 22
发送端等待ack超时
发送端重新发送数据分组： 21
发送端重新发送数据分组： 23
发送端接收ACK: ack 21
发送端接收ACK: ack 23
```

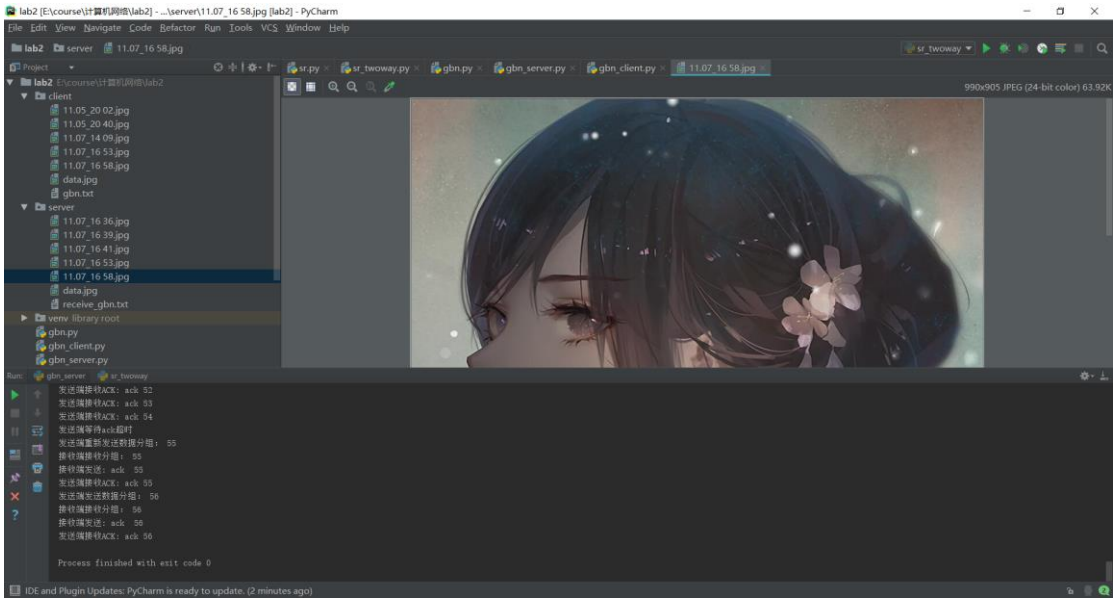
接收方回复确认报文ack21时丢失，导致接收方没有收到数据分组21的确认报文，由于SR协议**不再是停等协议，**即使发送方收到了ack22，也会重发数据分组21

传输结果：

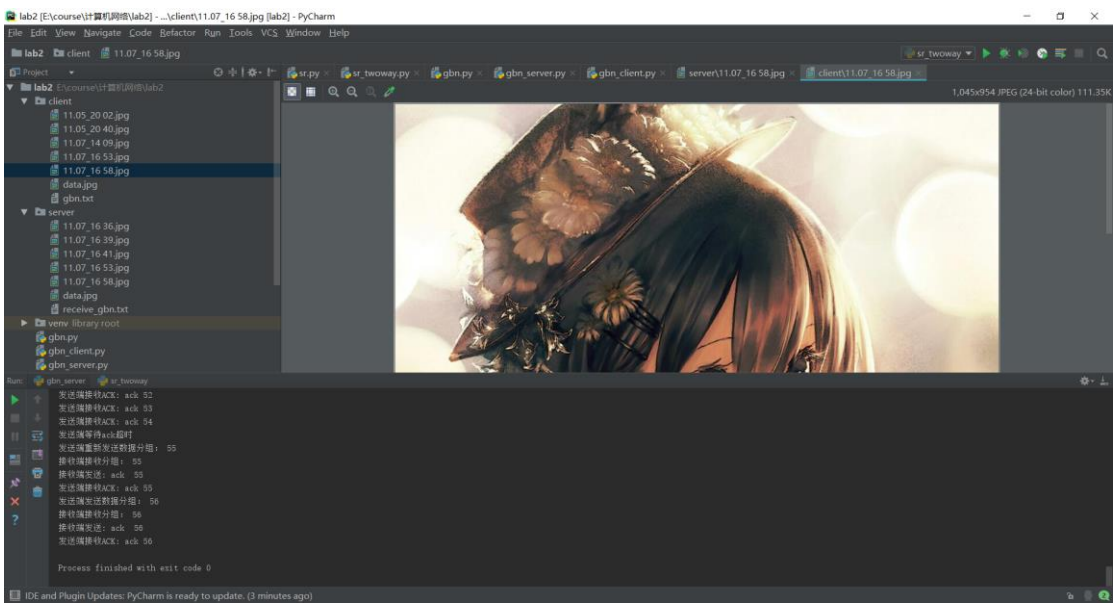


4）　SR协议双向数据传输测试

Client向server传输结果：



Server向client传输结果：



问题讨论：

GBN数据分组格式、确认分组格式、各个域的作用、典型交互过程、数据分组丢失验证模拟方法以及主要类等**在实验过程中**都已写明，不再重复。

心得体会：

1. 理解了滑动窗口协议的基本原理
2. 掌握了GBN的基本原理及简单实现
3. 掌握了SR协议的基本原理及简单实现
4. 掌握了基于UDP实现GBN以及SR协议的过程与技术