哈尔滨工业大学

计算机网络

实验报告

(2019 年度秋季学期)

姓名:	史纪元
学号:	1173300919
学院:	计算机科学与技术学院
教师:	刘亚维

实验二 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；

4）将所设计的 GBN 协议改进为 SR 协议。

三、实验过程及结果

1. GBN 协议分组格式及各个域的作用

由于本实验最终要求实现双向传输，故数据分组和确认分组可以使用同样的分组格式。

我采用的分组格式如下：

分组类型	序号	数据
1 byte	8 bytes	0~1024 bytes

其中

- **分组类型**字段长度为 1 字节，用来标识该分组是数据分组还是确认分组，数据分组的分组类型为 0，确认分组的分组类型为 1
- **序号**字段长度为 8 字节。若该分组是数据分组，则序号字段是发送方给该分组标记的序号。若该分组是确认分组，则序号字段为接收方已经正确收到的数据分组的序号。
- **数据**字段长度为 0~1024 字节，数据长度可以为 0

GBN 分组的定义如下：

```
class Data:
def __init__(self, is_ack, seq, data):
    self.is_ack = is_ack
    self.seq = seq
    self.data = data
```

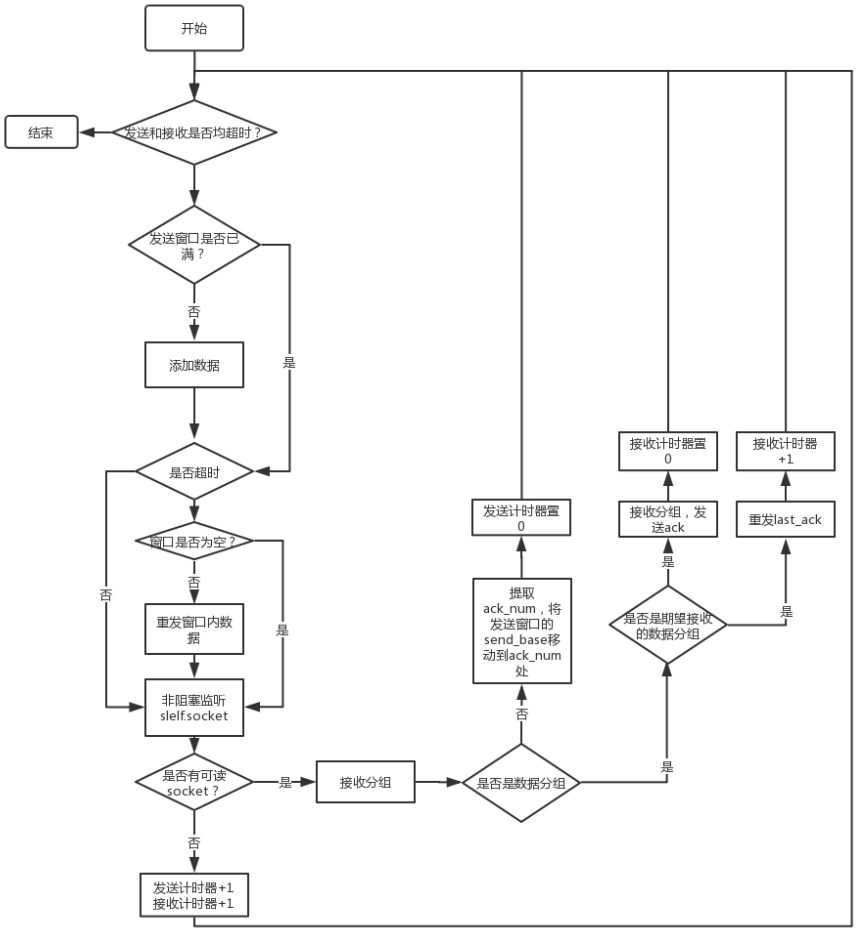
此外，在实现 SR 协议时，为了实现给每个分组都设置一个计时器，定义了一个新的类 DataGram，该类中有一个计时器属性，每轮循环中没有收到 ack 的分组的计时器会加一。

```
class DataGram:
def __init__(self, pkt, timer=0, is_acked=False):
    self.pkt = pkt
    self.timer = timer
    self.is_acked = is_acked
```

2. 协议两端程序流程图

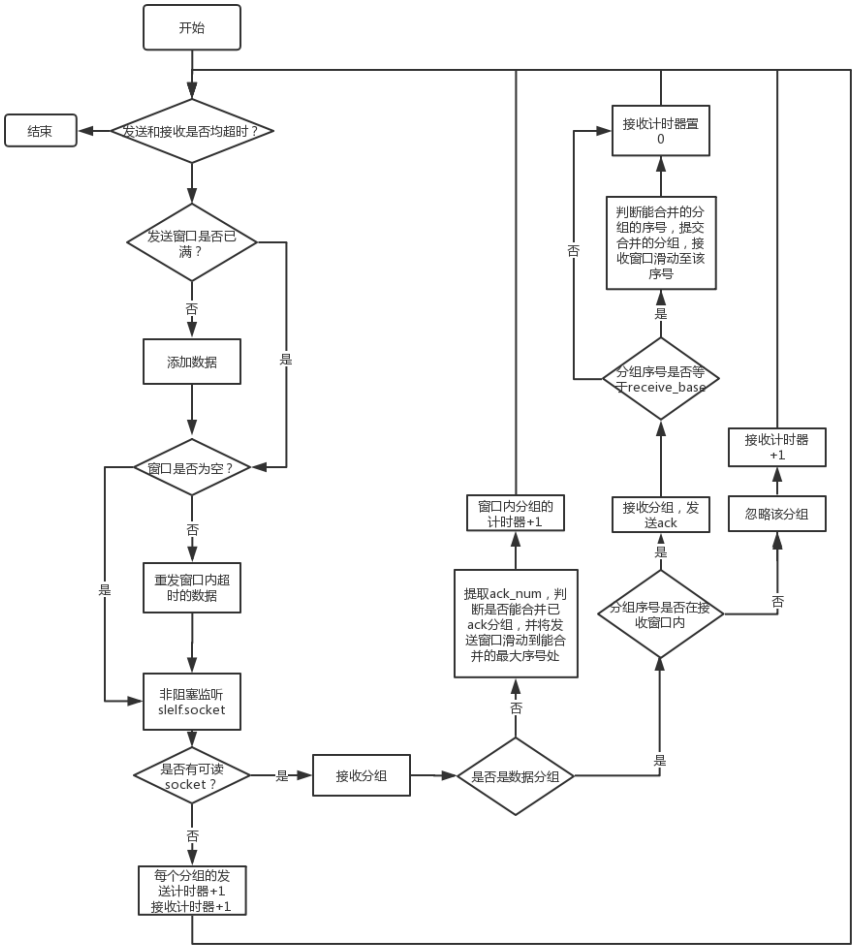
由于最后实现的是双向传输，故发送方和接收方的流程图相同

GBN 流程图：



SR 协议与 GBN 的区别在于 SR 协议的接收方维护一个接收窗口，只要收到序号在窗口内的数据分组都会正确接收并返回 ack，当收到按序到达的分组后再向后合并，提交给上层协议，接收窗口向后滑动。

SR 流程图：



3. 协议典型交互过程

GBN 协议：

发送方把需要发送的分组加入发送窗口，并向接收方发送窗口内的分组，若接收方接收到期望收到的分组，则向发送方发送 ACK+该分组的序号，否则发送 ACK+上一个正确接收的分组的序号。而发送方收到 ACK 分组后会将发送窗口的 send_base 移动到收到的 ACK 序号+1 处（如果窗口可以滑动的话），进入下一轮循环。

SR 协议：

SR 协议为每个分组都设置一个计时器，每次循环将重发窗口中超时的那些分组；另外 SR 协议的接收方维护一个接收窗口，若收到乱序但序号在接收窗口内的分组，则接收方会把它缓存，返回该分组的 ACK。当收到按序到达的分组时，会检查接收窗口内是否有能合并的已接收的分组，将这些分组合并后一起提交给上层协议，并滑动接收窗口。

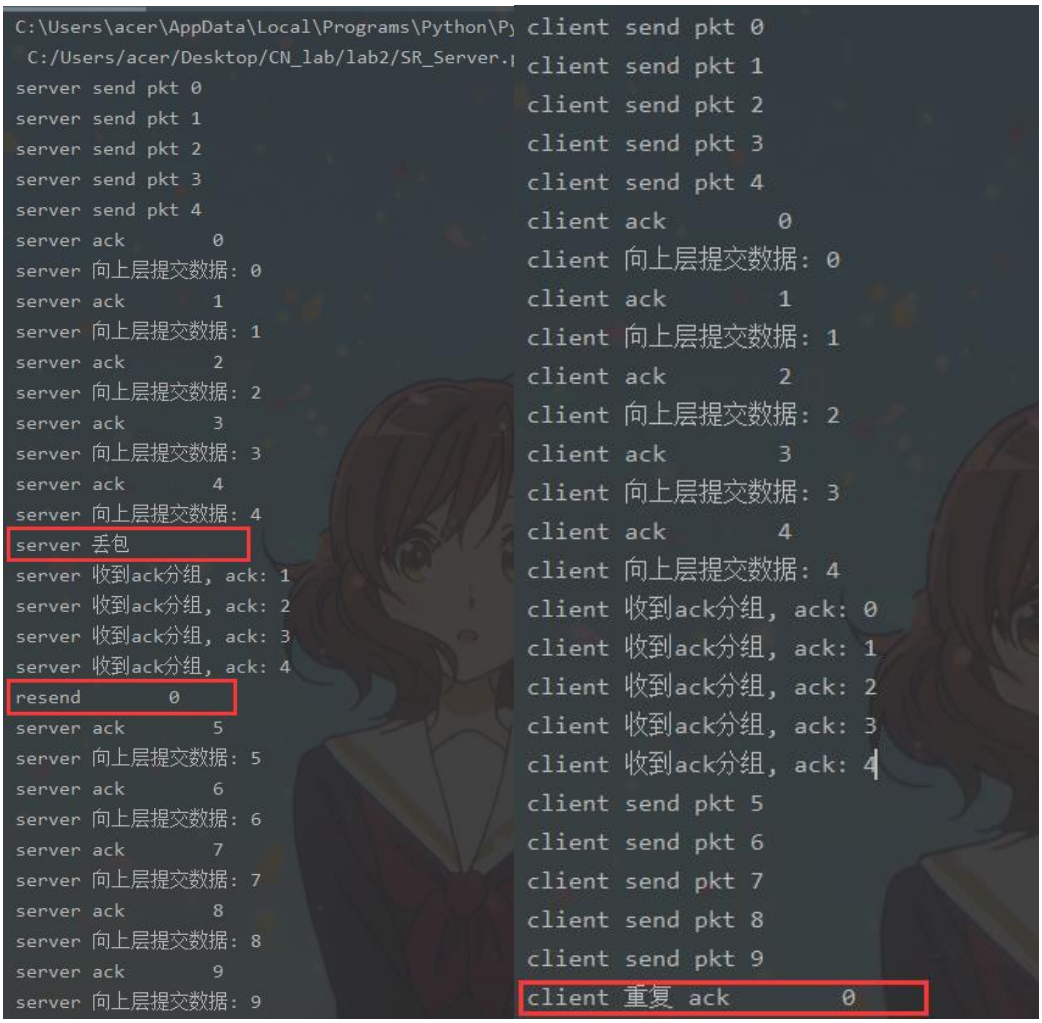
4. 数据分组丢失验证模拟方法

模拟丢包的方法：在接收数据时，设置非阻塞监听后，若收到数据，则调用 recvfrom()将该数据接收，并调用 random()函数，随机产生一个 0 到 1

之间的浮点数，若该浮点数<0.2（即丢包率为 20%），则不处理该数据，而是进行下一轮循环，这样就模拟了该数据包的丢失。

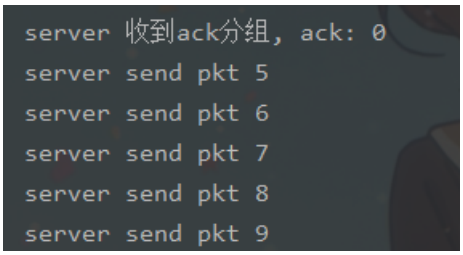
```
# 非阻塞监听
rs, ws, es = select.select([self.socket, ], [], [], 0.01)

while len(rs) > 0:
    rcv_pkt, address = self.socket.recvfrom(self.buffer_size)
    # 模拟丢包
    if random() < 0.2:
        for dgram in self.send_window:
            dgram.timer += 1
        receive_timer += 1
        print('server 丢包 ')
        rs, ws, es = select.select([self.socket, ], [], [], 0.01)
        continue
    ...
```

在有模拟分组丢失的情况下，运行程序（以 SR 协议为例）：



可以看到，客户端正确接收了服务器第一次发送的五个数据分组并都返回了 ack，但 ACK 0 在传输过程中发生了丢包，这样服务器端只收到了 ACK 1~4，故服务器的发送

窗口中仍有 0~5 五个分组，当 0 号分组的计时器超时后，服务器重发分组 0，此时 client 的接收窗口的 receive_base 已经滑动到了 5，故 client 收到分组 0 后仅向 server 重发 ack 0，这次 ack 0 没有丢包，被 server 成功接收（见下图），于是 server 的发送窗口可以向后滑动直到遇到第一个没有被 ack 的序号，即 5 号，并重新向窗口中添加分组并发送。



5. 程序实现的主要类、函数及其主要作用

Data 类：

```
class Data:
def __init__(self, is_ack, seq, data):
    self.is_ack = is_ack
    self.seq = seq
    self.data = data

def __str__(self):
    return str(self.is_ack) + str(self.seq) + str(self.data)
分组格式
+--------+-----+-----------+
| is_ack | seq |    data    |
+--------+-----+-----------+
0 表示是该分组数据分组
1 表示是分组是确认分组
is_ack 占 1 位
seq 占 8 位
```

Data 类用来模拟数据分组，对于需要发送的每个数据，发送方会将其封装到 Data 类，再进一步添加进发送窗口中并发送。

GBNServer 类：

```
class GBNServer:
def __init__(self):
    self.window_size = 5   # 窗口大小
    self.max_send_time = 3   # 发送超时时间
    self.max_receive_time = 10  # 接收超时时间
    self.address = ('127.0.0.1', 8888)  # 发送方地址
    self.client_address = ('127.0.0.1', 9999)   # 接收方地址
    self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.socket.bind(self.address)
    self.send_window = []   # 发送窗口
    self.receive_buffer = []
    self.buffer_size = 1024
```

GBNServer 类用来模拟 GBN 发送方，初始化 GBNServer 会新建一个 socket，并绑定到本地固定端口，接下来通过该 socket 进行分组的发送和接收。另外 GBNServer 还有窗口大小、超时时间、发送窗口、接收缓冲区以及缓冲区大小等属性。

GBNServer 类中有 send_and_receive()方法，该方法用来进行服务器端分组的**发送和接收**，其中发送和接收都通过 self.socket 这一个 socket 进行。先判断有无数据需要发送，然后进入非阻塞监听，若有可读 socket 则对该 socket 进行处理，当没有可读 socket 时，进入下一轮发送-接收循环。

具体的发送和接收的实现：

```
    # 当有窗口中有序号可用时，发送数据
while next_seq_num < send_base + self.window_size and next_seq_num < total:
    pkt = Data(0, '%8d' % next_seq_num, buffer[next_seq_num])
    self.socket.sendto(str(pkt).encode(), self.client_address)
    print('server send pkt ' + str(next_seq_num))
    self.send_window.append(pkt)
    if send_base == next_seq_num:
        send_timer = 0
    next_seq_num = next_seq_num + 1
```

当发送窗口未满且有数据需要发送时，将数据加入发送窗口，该数据的序号未 next_seq_num，然后将 next_seq_num 加一。

```
    # 超时，重传发送窗口中的数据
if send_timer > self.max_send_time and self.send_window:
    print('server send timeout, resend')
    send_timer = 0
    for pkt in self.send_window:
        self.socket.sendto(str(pkt).encode(), self.client_address)
        print('resend ' + str(pkt.seq))
```

若出现超时，则重发发送窗口中的数据分组。s

之后进入接收阶段，调用 select.select()函数进行非阻塞监听，若有可读 socket 则从另一端读取数据，先进行模拟丢包处理

```
    rs, ws, es = select.select([self.socket, ], [], [], 0.1)

while len(rs) > 0:
    rcv_pkt, address = self.socket.recvfrom(self.buffer_size)
    if random() < 0.1:
        send_timer += 1
        receive_timer += 1
        print('server 丢包 ')
        rs, ws, es = select.select([self.socket, ], [], [], 0.1)
        continue
    message = rcv_pkt.decode()
    receive_timer = 0
```

然后判断该分组是数据分组还是确认分组，若是数据分组，则判断其是否是期望收到的分组，只有当该数据分组是期望收到的分组时 GBN 才会正确接收

该分组并返回该分组的 ack 同时将计时器清零；其他任何情况都会丢弃该分组并返回上一个 ack 序号，计时器+1。

```
    if message[0] == '1':
        ack_num = int(message[1:9])
        for i in range(len(self.send_window)):
            if ack_num == int(self.send_window[i].seq):
                self.send_window = self.send_window[i + 1:]
                break
        send_base = ack_num + 1
        send_timer = 0
    elif message[0] == '0':
        rcv_seq_num = message[1:9]
        if int(rcv_seq_num) == expected_num:
            print('server ack ' + rcv_seq_num)
            self.receive_buffer.append(rcv_pkt.decode()[9:])
            ack_pkt = Data(1, '%8d ' % expected_num, '')
            self.socket.sendto(str(ack_pkt).encode(), self.client_address)
# 发送 ack 分组
            last_ack = expected_num
            expected_num += 1
        else:
            print('server 收到错误分组, 重发 ack ', last_ack)
            ack_pkt = Data(1, '%8d ' % last_ack, '')
            self.socket.sendto(str(ack_pkt).encode(), self.client_address)  #
发送 ack 分组
    else:
        pass
    rs, ws, es = select.select([self.socket, ], [], [], 0.1)
else:
    receive_timer += 1
    send_timer += 1
```

GBNClient 类：

```
    class GBNClient:
    def __init__(self):
        self.window_size = 5  # 窗口大小
        self.max_send_time = 3  # 发送超时时间
        self.max_receive_time = 10  # 接收超时时间
        self.address = ('127.0.0.1', 9999)  # 发送方地址
        self.server_address = ('127.0.0.1', 8888)  # 接收方地址
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.bind(self.address)
        self.send_window = []  # 发送窗口
```

```
        self.receive_buffer = []
        self.buffer_size = 1024
```

由于最后实现的是双向传输，故 GBNClient 类与 GBNServer 类相同，只是在协议中扮演不同角色。

DataGram 类：

```
    class DataGram:
    def __init__(self, pkt, timer=0, is_acked=False):
        self.pkt = pkt
        self.timer = timer
        self.is_acked = is_acked
```

由于 SR 协议中需要给每个分组都加一个计时器，故可以把 Data 封装进 DataGram 类，其中 DataGram 类有一个计时器和 is_acked 标记，分别用来计时和判断该分组是否已被 ack，只有超时并且 is_acked 为 False 的分组才会被重发。

SRServer 类：

```
    class SRServer:
    def __init__(self):
        self.send_window_size = 5   # 发送窗口大小
        self.receive_window_size = 5  # 接收窗口大小
        self.max_send_time = 5  # 发送超时时间
        self.max_receive_time = 15  # 接收超时时间
        self.address = ('127.0.0.1', 1111)  # 发送方地址
        self.client_address = ('127.0.0.1', 2222)   # 接收方地址
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.bind(self.address)
        self.send_window = []   # 发送窗口
        self.receive_data = []  # 有序数据
        self.receive_window = {}  # 接收窗口
        self.buffer_size = 1024
```

SRServer 类与 GBNServer 类相比多了接收窗口、接收窗口大小和用来模拟上层协议缓存的 receive_data（提交给上层的协议被存入 receive_data）。并且在 send_and_receive()方法中多了接收方接收窗口的操作以及给每个发送窗口中的分组设置计时器的功能。其他功能实现与 GBN 协议基本相同。

对于重发的处理：只重发分组计时器超时并未确认过的分组

```
    # 遍历已发送未确认分组，若有超时的分组，则重发
for dgram in self.send_window :
    if dgram.timer > self.max_send_time and not dgram.is_acked:
        self.socket.sendto(str(dgram.pkt).encode(), self.client_address)
        print('resend ' + str(dgram.pkt.seq))
```

接收时的处理：

若收到确认分组，会在发送窗口中检查确认的是否是 send_base 处的分组，若是，则判断其之后是否有连续的已确认的分组，发送窗口一直滑动到最后一个连续已确认的分组序号之后。若不是 send_base 分组，则只将所 ack 的分组的 is_acked 标记置为 True，不滑动窗口。

```
    for dgram in self.send_window:
    # 在窗口中找序号为 ack_num 的分组，并把 is_acked 设为 True
    if int(dgram.pkt.seq) == ack_num:
        dgram.timer = 0
        dgram.is_acked = True
        # 如果 ack 的是发送窗口中的第一个分组
        if self.send_window.index(dgram) == 0:
            idx = -1  # 用 idx 表示窗口中最后一个被 ack 的分组的下标
            for i in self.send_window:
                if i.is_acked:
                    idx += 1
                else:
                    break
            # 窗口滑动
            send_base = int(self.send_window[idx].pkt.seq) + 1
            self.send_window = self.send_window[idx+1:]
        break
```

若收到数据分组，则判断其序号是否是 receive_base，若是，则发送 ack，判断接收窗口内的分组是否可以向后合并，并将能合并的数据分组一并提交；若不是 receive_base 的分组，但位于接收窗口序号范围内，则接收该分组并存入接收窗口，发送 ack。若接收到的数据分组的序号小于 receive_base，则直接返回 ack 该分组的序号。对于其他情况则一律忽视。

```
    # 收到数据分组
elif message[0] == '0':
    # 发送窗口中所有分组计时器+1
    for dgram in self.send_window:
        dgram.timer += 1
    rcv_seq_num = message[1:9]
    # 收到了期望的分组
    if int(rcv_seq_num) == expected_num:
        # 先发送 ack
        print('server ack ' + rcv_seq_num)
        ack_pkt = Data(1, '%8d ' % expected_num, '')
        self.socket.sendto(str(ack_pkt).encode(), self.client_address)   # 发
送 ack 分组
        # 再看它后面有没有能合并的分组
        # 收到数据先加入接收窗口
        self.receive_window[int(rcv_seq_num)] = rcv_pkt
        tmp = [(k, self.receive_window[k]) for k in
sorted(self.receive_window.keys())]
        idx = 0   # idx 为接受窗口中能合并到的最后一个分组
        for i in range(len(tmp) - 1):
            if tmp[i + 1][0] - tmp[i][0] == 1:
                idx += 1
```

```
            else:
                break
        for i in range(idx + 1):
            self.receive_data.append(tmp[i][1].decode()[9:])   # 把接收窗口中
的数据提交给 receive_data
        base = int(tmp[0][1].decode()[1:9])
        end = int(tmp[idx][1].decode()[1:9])
        if base != end:
            print('server 向上层提交数据: ' + str(base) + ' ~ ' + str(end))
        else:
            print('server 向上层提交数据: ' + str(base))
        expected_num = tmp[idx][0]+1
        tmp = tmp[idx + 1:]   # 接收窗口滑动
        self.receive_window = dict(tmp)
    else:
        '''
        tmp 的格式为 [(序号, 数据分组)...]
        '''
        tmp = [(k, self.receive_window[k]) for k in
sorted(self.receive_window.keys())]
        # 若在 rcv_base~rcv_base + N -1 之内则加入接收窗口
        if expected_num <= int(rcv_seq_num) < expected_num +
self.receive_window_size - 1:
            self.receive_window[int(rcv_seq_num)] = rcv_pkt
            ack_pkt = Data(1, '%8d ' % int(rcv_seq_num), '')
            print('server ack ' + rcv_seq_num)
            self.socket.sendto(str(ack_pkt).encode(), self.client_address)  #
发送 ack 分组
        elif int(rcv_seq_num) < expected_num:
            ack_pkt = Data(1, '%8d ' % int(rcv_seq_num), '')
            print('server 重复 ack ' + rcv_seq_num)
            self.socket.sendto(str(ack_pkt).encode(), self.client_address)  #
发送 ack 分组
        else:
            pass
```

6. 实验结果

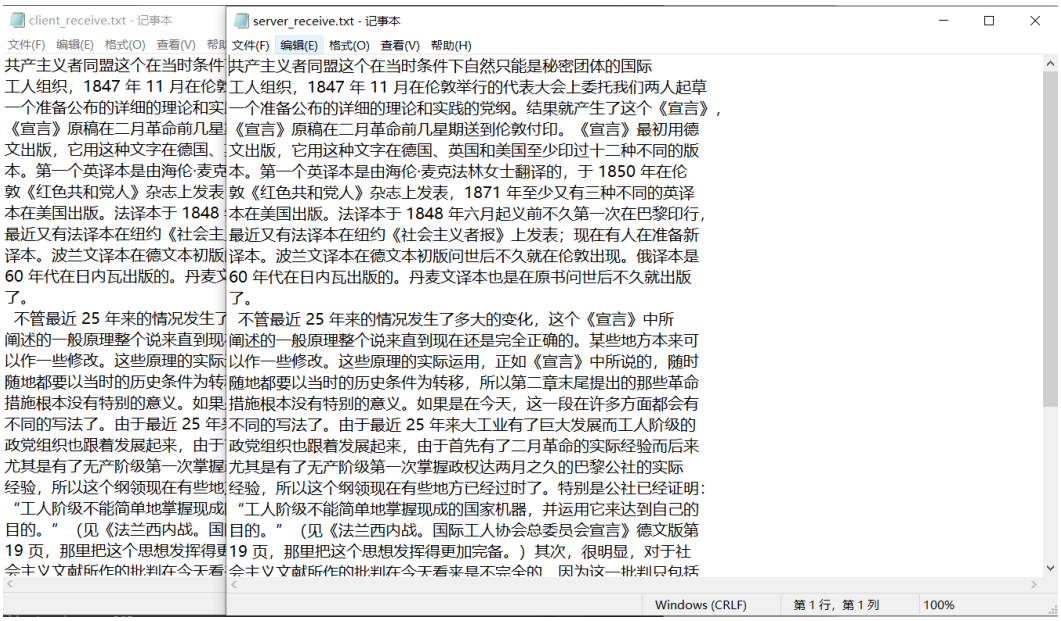
GBN 协议：

让 client 端发送共产党宣言第一篇序言，server 端发送两倍的该序言，文件如下：

运行程序：

```
client ack      132
server send pkt 133
client丢包
client收到错误分组，重发ack  132
server send pkt 134
server send pkt 135
server send pkt 136client丢包
client收到错误分组，重发ack  132

server send pkt 137
client收到错误分组，重发ack  132
server丢包
server send timeout，resend
resend      133
resend      134
client ack      133
resend      135client ack      134

resend      136
client ack      135
resend      137
client ack      136
client ack      137
server send pkt 138
server send pkt 139client ack      138

server send pkt 140
server send pkt 141
client丢包
server send pkt 195
client ack      193
client ack      194
client ack      195
server send pkt 196
server send pkt 197client ack      196

server send pkt 198
client ack      197
server send pkt 199
client ack      198
server send pkt 200
client ack      199
client ack      200
server send pkt 201
client ack      201
server send pkt 202
client ack      202server send pkt 203

client ack      203
finished

Process finished with exit code 0
```

最后在 client_receive.txt 和 server_receive.txt 接收到了对方发送的数据：

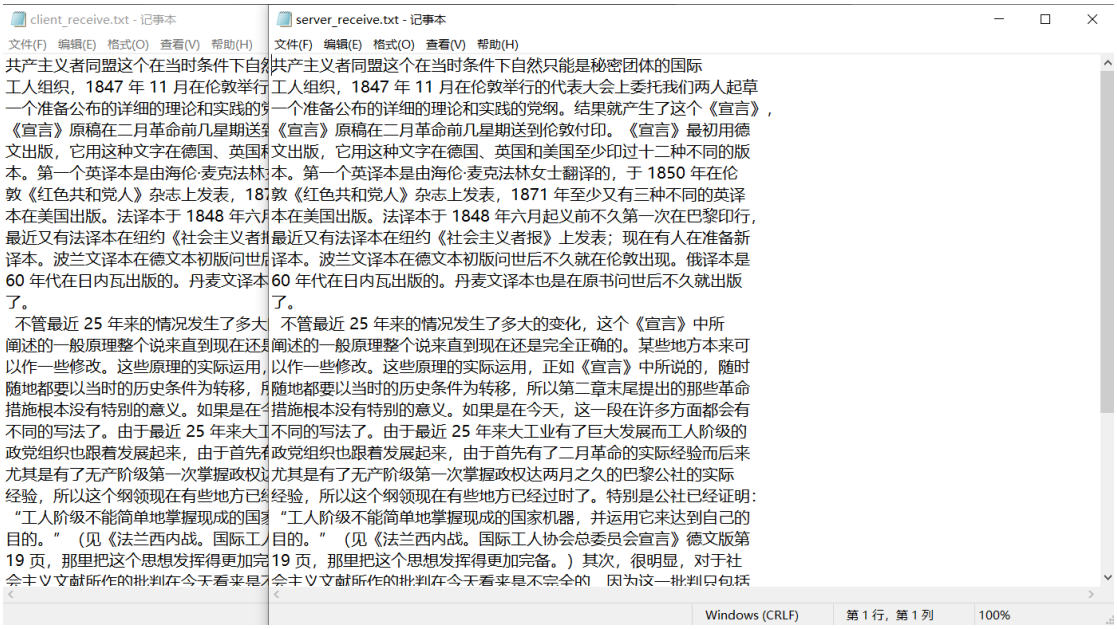


SR 协议：

协议双方发送同样的文本，运行测试程序：

```
server 收到ack分组, ack: 199
client 重复 ack      197
resend      197
resend      198
client 重复 ack      198
resend      200
resend      201client ack      200

client 向上层提交数据: 200
server 收到ack分组, ack: 197
server 收到ack分组, ack: 198
server 收到ack分组, ack: 200
client 丢包
server send pkt 202
client ack      202
server send pkt 203
resend      201
client ack      203
server 收到ack分组, ack: 202
client ack      201
client 向上层提交数据: 201 ~ 203
server 收到ack分组, ack: 203
server 收到ack分组, ack: 201
finished

Process finished with exit code 0
```

可以看到 SR 协议中的接收方有接收窗口。程序运行完成后，在 client_receive.txt 和 server_receive.txt 中接收到了对方发送的数据。



7. 源程序

GBN：

```
import socket
from random import random
import select


'''
分组格式
+--------+-----+-----------+
| is_ack | seq |    data   |
+--------+-----+-----------+
0 表示不是 ack
1 表示是 ack
is_ack 占 1 位
seq 占 8 位
'''


class Data:
    def __init__(self, is_ack, seq, data):
        self.is_ack = is_ack
        self.seq = seq
        self.data = data

    def __str__(self):
        return str(self.is_ack) + str(self.seq) + str(self.data)
```

```
class GBNServer:
    def __init__(self):
        self.window_size = 5   # 窗口大小
        self.max_send_time = 3   # 发送超时时间
        self.max_receive_time = 10  # 接收超时时间
        self.address = ('127.0.0.1', 8888)  # 发送方地址
        self.client_address = ('127.0.0.1', 9999)   # 接收方地址
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.bind(self.address)
        self.send_window = []   # 发送窗口
        self.receive_buffer = []
        self.buffer_size = 1024

    def send_and_receive(self, buffer):
        send_timer = 0
        send_base = 0
        next_seq_num = send_base
        expected_num = 0
        receive_timer = 0
        last_ack = -1
        total = len(buffer)
        while True:
            if not self.send_window and receive_timer > self.max_receive_time:
                with open('server_receive.txt', 'w') as f:
                    for data in self.receive_buffer:
                        f.write(data)
                break

            # 当有窗口中有序号可用时，发送数据
            while next_seq_num < send_base + self.window_size and next_seq_num
< total:
                pkt = Data(0, '%8d' % next_seq_num, buffer[next_seq_num])
                self.socket.sendto(str(pkt).encode(), self.client_address)
                print('server send pkt ' + str(next_seq_num))
                self.send_window.append(pkt)
                if send_base == next_seq_num:
                    send_timer = 0
                next_seq_num = next_seq_num + 1

            # 超时，重传发送窗口中的数据
            if send_timer > self.max_send_time and self.send_window:
                print('server send timeout, resend')
```

```
                send_timer = 0
                for pkt in self.send_window:
                    self.socket.sendto(str(pkt).encode(), self.client_address)
                    print('resend ' + str(pkt.seq))

            rs, ws, es = select.select([self.socket, ], [], [], 0.1)

            while len(rs) > 0:
                rcv_pkt, address = self.socket.recvfrom(self.buffer_size)
                if random() < 0.1:
                    send_timer += 1
                    receive_timer += 1
                    print('server 丢包 ')
                    rs, ws, es = select.select([self.socket, ], [], [], 0.1)
                    continue
                message = rcv_pkt.decode()
                receive_timer = 0

                if message[0] == '1':
                    ack_num = int(message[1:9])
                    for i in range(len(self.send_window)):
                        if ack_num == int(self.send_window[i].seq):
                            self.send_window = self.send_window[i + 1:]
                            break
                    send_base = ack_num + 1
                    send_timer = 0
                elif message[0] == '0':
                    rcv_seq_num = message[1:9]
                    if int(rcv_seq_num) == expected_num:
                        print('server ack ' + rcv_seq_num)
                        self.receive_buffer.append(rcv_pkt.decode()[9:])
                        ack_pkt = Data(1, '%8d ' % expected_num, '')
                        self.socket.sendto(str(ack_pkt).encode(),
self.client_address)   # 发送 ack 分组
                        last_ack = expected_num
                        expected_num += 1
                    else:
                        print('server 收到错误分组, 重发 ack ', last_ack)
                        ack_pkt = Data(1, '%8d ' % last_ack, '')
                        self.socket.sendto(str(ack_pkt).encode(),
self.client_address)  # 发送 ack 分组
                else:
                    pass
                rs, ws, es = select.select([self.socket, ], [], [], 0.1)
```

```
            else:
                receive_timer += 1
                send_timer += 1


def start():
    server_socket = GBNServer()
    data = []
    with open('server_send.txt', 'r') as f:
        while True:
            pkt = f.read(10)
            if len(pkt) > 0:
                data.append(pkt)
            else:
                break
    timer = 0
    while True:
        # 服务器计时器，如果收不到客户端的请求则退出
        if timer > 20:
            return

        rs, ws, es = select.select([server_socket.socket, ], [], [], 1)
        if len(rs) > 0:
            message, address =
server_socket.socket.recvfrom(server_socket.buffer_size)
            if message.decode() == '-testgbn':
                server_socket.send_and_receive(data)
            if message.decode() == '-finish':
                print('finished')
                return
        timer += 1


if __name__ == '__main__':
    start()

import socket
from random import random
import select

'''
分组格式
+--------+-----+-----------+
| is_ack | seq |    data   |
```

```
+--------+-----+-----------+
0 表示不是 ack
1 表示是 ack
is_ack 占 1 位
seq 占 8 位
'''


class Data:
    def __init__(self, is_ack, seq, data):
        self.is_ack = is_ack
        self.seq = seq
        self.data = data

    def __str__(self):
        return str(self.is_ack) + str(self.seq) + str(self.data)


class GBNClient:
    def __init__(self):
        self.window_size = 5  # 窗口大小
        self.max_send_time = 3  # 发送超时时间
        self.max_receive_time = 10  # 接收超时时间
        self.address = ('127.0.0.1', 9999)  # 发送方地址
        self.server_address = ('127.0.0.1', 8888)  # 接收方地址
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.bind(self.address)
        self.send_window = []  # 发送窗口
        self.receive_buffer = []
        self.buffer_size = 1024

    def send_and_receive(self, buffer):
        send_timer = 0
        send_base = 0
        next_seq_num = send_base
        expected_num = 0
        receive_timer = 0
        last_ack = -1
        total = len(buffer)
        while True:
            if not self.send_window and receive_timer > self.max_receive_time:
                with open('client_receive.txt', 'w') as f:
                    for data in self.receive_buffer:
                        f.write(data)
```

```
                break

            while next_seq_num < send_base + self.window_size and next_seq_num
< total:
                pkt = Data(0, '%8d' % next_seq_num, buffer[next_seq_num])
                self.socket.sendto(str(pkt).encode(), self.server_address)
                print('client send pkt ' + str(next_seq_num))
                self.send_window.append(pkt)
                if send_base == next_seq_num:
                    send_timer = 0
                next_seq_num = next_seq_num + 1

            # 超时，重传发送窗口中的数据
            if send_timer > self.max_send_time and self.send_window:
                print('client send timeout, resend')
                send_timer = 0
                for pkt in self.send_window:
                    self.socket.sendto(str(pkt).encode(),self.server_address)
                    print('resend ' + str(pkt.seq))

            rs, ws, es = select.select([self.socket, ], [], [], 0.1)

            while len(rs) > 0:
                rcv_pkt, address = self.socket.recvfrom(self.buffer_size)
                if random() < 0.1:
                    receive_timer += 1
                    send_timer += 1
                    print('client 丢包 ')
                    rs, ws, es = select.select([self.socket, ], [], [], 0.1)
                    continue
                message = rcv_pkt.decode()
                receive_timer = 0

                if message[0] == '1':
                    ack_num = int(message[1:9])
                    for i in range(len(self.send_window)):
                        if ack_num == int(self.send_window[i].seq):
                            self.send_window = self.send_window[i + 1:]
                            break
                    send_base = ack_num + 1
                    send_timer = 0
                elif message[0] == '0':
                    rcv_seq_num = message[1:9]
                    if int(rcv_seq_num) == expected_num:
```

```
                    print('client ack ' + rcv_seq_num)
                    self.receive_buffer.append(rcv_pkt.decode()[9:])
                    ack_pkt = Data(1, '%8d ' % expected_num, '')
                    self.socket.sendto(str(ack_pkt).encode(),
self.server_address)  # 发送 ack 分组
                    last_ack = expected_num
                    expected_num += 1
                else:
                    print('client 收到错误分组, 重发 ack ', last_ack)
                    ack_pkt = Data(1, '%8d ' % last_ack, '')
                    self.socket.sendto(str(ack_pkt).encode(),
self.server_address)  # 发送 ack 分组
            else:
                pass
            rs, ws, es = select.select([self.socket, ], [], [], 0.1)
        else:
            receive_timer += 1
            send_timer += 1


def start():
    client_socket = GBNClient()
    data = []
    with open('client_send.txt', 'r') as f:
        while True:
            pkt = f.read(10)
            if len(pkt) > 0:
                data.append(pkt)
            else:
                break
    client_socket.socket.sendto('-testgbn'.encode(),
client_socket.server_address)
    client_socket.send_and_receive(data)
    client_socket.socket.sendto('-finish'.encode(),
client_socket.server_address)


if __name__ == '__main__':
    start()
```

SR：

```
import socket
from random import random
import select
```

```
'''
分组格式
+--------+-----+-----------+
| is_ack | seq |    data   |
+--------+-----+-----------+
0 表示不是 ack
1 表示是 ack
is_ack 占 1 位
seq 占 8 位
'''


class Data:
    def __init__(self, is_ack, seq, data):
        self.is_ack = is_ack
        self.seq = seq
        self.data = data

    def __str__(self):
        return str(self.is_ack) + str(self.seq) + str(self.data)


class DataGram:
    def __init__(self, pkt, timer=0, is_acked=False):
        self.pkt = pkt
        self.timer = timer
        self.is_acked = is_acked


class SRServer:
    def __init__(self):
        self.send_window_size = 5   # 发送窗口大小
        self.receive_window_size = 5  # 接收窗口大小
        self.max_send_time = 5  # 发送超时时间
        self.max_receive_time = 15  # 接收超时时间
        self.address = ('127.0.0.1', 1111)  # 发送方地址
        self.client_address = ('127.0.0.1', 2222)   # 接收方地址
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.bind(self.address)
        self.send_window = []   # 发送窗口
        self.receive_data = []  # 有序数据
        self.receive_window = {}  # 接收窗口
```

```
        self.buffer_size = 1024

    def send_and_receive(self, buffer):
        send_base = 0
        next_seq_num = send_base
        expected_num = 0
        receive_timer = 0
        total = len(buffer)
        while True:
            # print(self.send_finished)
            # print(self.receive_finished)
            if not self.send_window and receive_timer > self.max_receive_time:
                with open('server_receive.txt', 'w') as f:
                    for data in self.receive_data:
                        f.write(data)
                break

            # 当有窗口中有序号可用时，发送数据
            while next_seq_num < send_base + self.send_window_size and
next_seq_num < total:
                pkt = Data(0, '%8d' % next_seq_num, buffer[next_seq_num])
                self.socket.sendto(str(pkt).encode(), self.client_address)
                print('server send pkt ' + str(next_seq_num))
                # 给每个分组计时器初始化为 0  把数据报加入发送窗口
                self.send_window.append(DataGram(pkt))
                next_seq_num = next_seq_num + 1

            # 发送窗口为空，将 send_finished 设为 True 反复发 finish 以防 finish 丢
失
            # if not self.send_window:
                # print('server finished sending')
                # self.socket.sendto('finish'.encode(), self.client_address)
                # self.send_finished = True

            # 遍历已发送未确认分组，若有超时的分组，则重发
            for dgram in self.send_window :
                if dgram.timer > self.max_send_time and not dgram.is_acked:
                    self.socket.sendto(str(dgram.pkt).encode(),
self.client_address)
                    print('resend ' + str(dgram.pkt.seq))

            '''
            select()的机制中提供一 fd_set 的数据结构，实际上是一 long 类型的数组， 每
一个数组元素都能与一打开的文件句柄
```

```
            （不管是 Socket 句柄，还是其他文件或命名管道或设备句柄）建立联系，建立联
系的工作由程序员完成，  当调用 select()
            时，由内核根据 IO 状态修改 fd_set 的内容，由此来通知执行了 select()的进程
哪一 Socket 或文件可读或可写。
            返回值：准备就绪的描述符数，若超时则返回 0，若出错则返回-1。
            '''

            # 非阻塞监听
            rs, ws, es = select.select([self.socket, ], [], [], 0.01)

            while len(rs) > 0:
                rcv_pkt, address = self.socket.recvfrom(self.buffer_size)
                # 模拟丢包
                if random() < 0.2:
                    for dgram in self.send_window:
                        dgram.timer += 1
                    receive_timer += 1
                    print('server 丢包 ')
                    rs, ws, es = select.select([self.socket, ], [], [], 0.01)
                    continue
                message = rcv_pkt.decode()
                receive_timer = 0
                # if message == 'finish':
                #     with open('server_receive.txt', 'w') as f:
                #         for data in self.receive_data:
                #             f.write(data)
                #     self.receive_finished = True

                # 收到的是 ACK 分组
                if message[0] == '1':
                    # 获取 ACK 的序号
                    ack_num = int(message[1:9])
                    print('server 收到 ack 分组, ack: ' + str(ack_num))
                    for dgram in self.send_window:
                        # 在窗口中找序号为 ack_num 的分组，并把 is_acked 设为 True
                        if int(dgram.pkt.seq) == ack_num:
                            dgram.timer = 0
                            dgram.is_acked = True
                            # 如果 ack 的是发送窗口中的第一个分组
                            if self.send_window.index(dgram) == 0:
                                idx = -1  # 用 idx 表示窗口中最后一个被 ack 的分组的
下标

                                for i in self.send_window:
                                    if i.is_acked:
```

```
                                    idx += 1
                                else:
                                    break
                            # 窗口滑动
                            send_base = int(self.send_window[idx].pkt.seq) +
1
                            self.send_window = self.send_window[idx+1:]
                        break
                # 收到数据分组
                elif message[0] == '0':
                    # 发送窗口中所有分组计时器+1
                    for dgram in self.send_window:
                        dgram.timer += 1
                    rcv_seq_num = message[1:9]
                    # 收到了期望的分组
                    if int(rcv_seq_num) == expected_num:
                        # 先发送 ack
                        print('server ack ' + rcv_seq_num)
                        ack_pkt = Data(1, '%8d ' % expected_num, '')
                        self.socket.sendto(str(ack_pkt).encode(),
self.client_address)   # 发送 ack 分组
                        # 再看它后面有没有能合并的分组
                        # 收到数据先加入接收窗口
                        self.receive_window[int(rcv_seq_num)] = rcv_pkt
                        tmp = [(k, self.receive_window[k]) for k in
sorted(self.receive_window.keys())]
                        idx = 0   # idx 为接受窗口中能合并到的最后一个分组
                        for i in range(len(tmp) - 1):
                            if tmp[i + 1][0] - tmp[i][0] == 1:
                                idx += 1
                            else:
                                break
                        for i in range(idx + 1):
                            self.receive_data.append(tmp[i][1].decode()[9:])
# 把接收窗口中的数据提交给 receive_data
                        base = int(tmp[0][1].decode()[1:9])
                        end = int(tmp[idx][1].decode()[1:9])
                        if base != end:
                            print('server 向上层提交数据: ' + str(base) + ' ~ ' +
str(end))
                        else:
                            print('server 向上层提交数据: ' + str(base))
                        expected_num = tmp[idx][0]+1
                        tmp = tmp[idx + 1:]   # 接收窗口滑动
```

```
                    self.receive_window = dict(tmp)
                else:
                    '''
                    tmp 的格式为 [(序号, 数据分组)...]
                    '''
                    tmp = [(k, self.receive_window[k]) for k in
sorted(self.receive_window.keys())]
                    # 若在 rcv_base~rcv_base + N -1 之内则加入接收窗口
                    if expected_num <= int(rcv_seq_num) < expected_num +
self.receive_window_size - 1:
                        self.receive_window[int(rcv_seq_num)] = rcv_pkt
                        ack_pkt = Data(1, '%8d ' % int(rcv_seq_num), '')
                        print('server ack ' + rcv_seq_num)
                        self.socket.sendto(str(ack_pkt).encode(),
self.client_address)  # 发送 ack 分组
                    elif int(rcv_seq_num) < expected_num:
                        ack_pkt = Data(1, '%8d ' % int(rcv_seq_num), '')
                        print('server 重复 ack ' + rcv_seq_num)
                        self.socket.sendto(str(ack_pkt).encode(),
self.client_address)  # 发送 ack 分组
                    else:
                        pass
            else:
                pass
            rs, ws, es = select.select([self.socket, ], [], [], 0.01)
        else:
            receive_timer += 1
            for dgram in self.send_window:
                dgram.timer += 1


def start():
    server_socket = SRServer()
    data = []
    with open('server_send.txt', 'r') as f:
        while True:
            pkt = f.read(10)
            if len(pkt) > 0:
                data.append(pkt)
            else:
                break
    timer = 0
    while True:
        # 服务器计时器，如果收不到客户端的请求则退出
```

```
        if timer > 20:
            return

        rs, ws, es = select.select([server_socket.socket, ], [], [], 1)
        if len(rs) > 0:
            message, address =
server_socket.socket.recvfrom(server_socket.buffer_size)
            if message.decode() == '-testsr':
                server_socket.send_and_receive(data)
            if message.decode() == '-finish':
                print('finished')
                return
        # print(timer)
        # print('here')
        timer += 1


if __name__ == '__main__':
    start()

import socket
from random import random
import select


'''
分组格式
+--------+-----+-----------+
| is_ack | seq |    data   |
+--------+-----+-----------+
0 表示不是 ack
1 表示是 ack
is_ack 占 1 位
seq 占 8 位
'''


class Data:
    def __init__(self, is_ack, seq, data):
        self.is_ack = is_ack
        self.seq = seq
        self.data = data

    def __str__(self):
```

```
        return str(self.is_ack) + str(self.seq) + str(self.data)


class DataGram:
    def __init__(self, pkt, timer=0, is_acked=False):
        self.pkt = pkt
        self.timer = timer
        self.is_acked = is_acked


class SRClient:
    def __init__(self):
        self.send_window_size = 5   # 发送窗口大小
        self.receive_window_size = 5  # 接收窗口大小
        self.max_send_time = 5   # 发送超时时间
        self.max_receive_time = 15  # 接收超时时间
        self.address = ('127.0.0.1', 2222)  # 发送方地址
        self.server_address = ('127.0.0.1', 1111)   # 接收方地址
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.bind(self.address)
        self.send_window = []   # 发送窗口
        self.receive_data = []  # 有序数据
        self.receive_window = {}  # 接收窗口
        self.buffer_size = 1024

    def send_and_receive(self, buffer):
        send_base = 0
        next_seq_num = send_base
        expected_num = 0
        receive_timer = 0
        total = len(buffer)
        while True:
            # print(self.send_finished)
            # print(self.receive_finished)
            # if self.send_finished and self.receive_finished:
            #     break
            if not self.send_window and receive_timer > self.max_receive_time:
                with open('client_receive.txt', 'w') as f:
                    for data in self.receive_data:
                        f.write(data)
                break

            # 当有窗口中有序号可用时，发送数据
            while next_seq_num < send_base + self.send_window_size and
```

```
next_seq_num < total:
                pkt = Data(0, '%8d' % next_seq_num, buffer[next_seq_num])
                self.socket.sendto(str(pkt).encode(), self.server_address)
                print('client send pkt ' + str(next_seq_num))
                # 给每个分组计时器初始化为 0  把数据报加入发送窗口
                self.send_window.append(DataGram(pkt))
                next_seq_num = next_seq_num + 1

            # 发送窗口为空，将 send_finished 设为 True 反复发 finish 以防 finish 丢
失
            # if not self.send_window:
            #     # print('server finished sending')
            #     self.socket.sendto('finish'.encode(), self.server_address)
            #     self.send_finished = True

            # 遍历已发送未确认分组，若有超时的分组，则重发
            for dgram in self.send_window:
                if dgram.timer > self.max_send_time and not dgram.is_acked:
                    self.socket.sendto(str(dgram.pkt).encode(),
self.server_address)
                    print('resend ' + str(dgram.pkt.seq))

            '''
            select()的机制中提供一 fd_set 的数据结构，实际上是一 long 类型的数组，每
一个数组元素都能与一打开的文件句柄
            （不管是 Socket 句柄，还是其他文件或命名管道或设备句柄）建立联系，建立联
系的工作由程序员完成，  当调用 select()
            时，由内核根据 IO 状态修改 fd_set 的内容，由此来通知执行了 select()的进程
哪一 Socket 或文件可读或可写。
            返回值：准备就绪的描述符数，若超时则返回 0，若出错则返回-1。
            '''

            # 非阻塞监听
            rs, ws, es = select.select([self.socket, ], [], [], 0.01)

            while len(rs) > 0:
                # print(rs)
                rcv_pkt, address = self.socket.recvfrom(self.buffer_size)
                # 模拟丢包
                if random() < 0.2:
                    for dgram in self.send_window:
                        dgram.timer += 1
                    receive_timer += 1
                    print('client 丢包 ')
```

```
                rs, ws, es = select.select([self.socket, ], [], [], 0.01)
                continue
            receive_timer = 0
            message = rcv_pkt.decode()
            # if message == 'finish':
            #     with open('client_receive.txt', 'w') as f:
            #         for data in self.receive_data:
            #             f.write(data)
            #     self.receive_finished = True
            #     break

            # 收到的是 ACK 分组
            if message[0] == '1':
                # 获取 ACK 的序号
                ack_num = int(message[1:9])
                print('client 收到 ack 分组, ack: ' + str(ack_num))
                for dgram in self.send_window:
                    # 在窗口中找序号为 ack_num 的分组, 并把 is_acked 设为 True
                    if int(dgram.pkt.seq) == ack_num:
                        dgram.timer = 0
                        dgram.is_acked = True
                        # 如果 ack 的是窗口中的第一个分组
                        if self.send_window.index(dgram) == 0:
                            idx = -1  # idx 为窗口中最后一个被 ack 的分组的下标
                            for i in self.send_window:
                                if i.is_acked:
                                    idx += 1
                                else:
                                    break
                            # 窗口滑动
                            send_base = int(self.send_window[idx].pkt.seq) +
1

                            self.send_window = self.send_window[idx+1:]
                        break
            # 收到数据分组
            elif message[0] == '0':
                # 发送窗口中所有分组计时器+1
                for dgram in self.send_window:
                    dgram.timer += 1
                rcv_seq_num = message[1:9]
                # 收到了期望的分组
                if int(rcv_seq_num) == expected_num:
                    # 先发送 ack
                    print('client ack ' + rcv_seq_num)
```

```
                    ack_pkt = Data(1, '%8d ' % expected_num, '')
                    self.socket.sendto(str(ack_pkt).encode(),
self.server_address)  # 发送 ack 分组
                    # 再看它后面有没有能合并的分组
                    # 收到数据先加入接收窗口
                    self.receive_window[int(rcv_seq_num)] = rcv_pkt
                    tmp = [(k, self.receive_window[k]) for k in
sorted(self.receive_window.keys())]
                    idx = 0   # idx 为接受窗口中能合并到的最后一个分组
                    for i in range(len(tmp) - 1):
                        if tmp[i + 1][0] - tmp[i][0] == 1:
                            idx += 1
                        else:
                            break
                    for i in range(idx + 1):
                        # print(tmp[i][1].decode())
                        self.receive_data.append(tmp[i][1].decode()[9:])
# 把接收窗口中的数据提交给 receive_data
                    # 记录提交的分组的序号
                    base = int(tmp[0][1].decode()[1:9])
                    end = int(tmp[idx][1].decode()[1:9])
                    if base != end:
                        print('client 向上层提交数据: ' + str(base) + ' ~ ' +
str(end))
                    else:
                        print('client 向上层提交数据: ' + str(base))
                    expected_num = tmp[idx][0]+1
                    tmp = tmp[idx + 1:]   # 接收窗口滑动
                    self.receive_window = dict(tmp)
                else:
                    '''
                    tmp 的格式为  [(序号, 数据分组)...]
                    '''
                    tmp = [(k, self.receive_window[k]) for k in
sorted(self.receive_window.keys())]
                    # 若在 rcv_base~rcv_base + N -1 之内则加入接收窗口
                    if expected_num <= int(rcv_seq_num) < expected_num +
self.receive_window_size - 1:
                        self.receive_window[int(rcv_seq_num)] = rcv_pkt
                        ack_pkt = Data(1, '%8d ' % int(rcv_seq_num), '')
                        print('client ack ' + rcv_seq_num)
                        self.socket.sendto(str(ack_pkt).encode(),
self.server_address)  # 发送 ack 分组
                    elif int(rcv_seq_num) < expected_num:
```

```
                        ack_pkt = Data(1, '%8d ' % int(rcv_seq_num), '')
                        print('client 重复 ack ' + rcv_seq_num)
                        self.socket.sendto(str(ack_pkt).encode(),
self.server_address)  # 发送 ack 分组
                    else:
                        print('丢弃')
                        pass
            else:
                pass
            rs, ws, es = select.select([self.socket, ], [], [], 0.01)
        else:
            receive_timer += 1
            for dgram in self.send_window:
                dgram.timer += 1


def start():
    client_socket = SRClient()
    data = []
    with open('client_send.txt', 'r') as f:
        while True:
            pkt = f.read(10)
            if len(pkt) > 0:
                data.append(pkt)
            else:
                break

    client_socket.socket.sendto('-testsr'.encode(),
client_socket.server_address)
    client_socket.send_and_receive(data)
    client_socket.socket.sendto('-finish'.encode(),
client_socket.server_address)


if __name__ == '__main__':
    start()
```

测试程序：

```
import threading
from GBN_client import start as gbn_client_start
from GBN_server import start as gbn_server_start
from SR_Client import start as sr_client_start
from SR_Server import start as sr_server_start
```

```
def main():
    # t1 = threading.Thread(target=gbn_server_start, args=())
    # t2 = threading.Thread(target=gbn_client_start, args=())
    t1 = threading.Thread(target=sr_server_start, args=())
    t2 = threading.Thread(target=sr_client_start, args=())
    t1.start()
    t2.start()


if __name__ == '__main__':
    main()
```

四、实验心得

通过本次实验，我对 GBN 和 SR 协议有了更清晰的认识，并且模拟丢包使我更直观地理解了 GBN 和 SR 协议使如何实现可靠传输的，以及 SR 相较于 GBN 的优点。本次实验还让我学到了有非阻塞监听功能的 select 函数的用法。对于双向传输的处理，一开始我尝试每轮循环双方只监听一次可读 socket，但后来发现这样会大大降低传输的速率，于是我把接收时监听 socket 也改为了循环监听，这样就提高了数据传输的速率。另外在丢包率的设置上也让我体会到了一个可靠的底层信道对数据的传输也十分重要。