



哈尔滨工业大学
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-停等协议的设计与实现， 可靠数据传输协议-GBN 协议的设计与实现					
姓名	余涛		院系	计算机科学与技术学院		
班级	1803202		学号	1180300829		
任课教师	刘亚维		指导教师	刘亚维		
实验地点	格物 207		实验时间	2020.11.07		
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						

计算学部

实验目的：

本次实验的主要目的。

实验二：理解可靠数据传输的基本原理；掌握停等协议的工作原理；掌握基于UDP 设计并实现一个停等协议的过程与技术。

实验三：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。

实验内容：

概述本次实验的主要内容，包含的实验项等。

实验二：

- 1) 基于UDP 实现的停等协议，可以不进行差错检测，可以利用UDP协议差错检测；
- 2) 为了验证所设计协议是否可以处理数据丢失，可以考虑在数据接收端或发送端引入数据丢失。
- 3) 在开发停等协议之前，需要先设计协议数据分组格式以及确认分组格式。
- 4) 计时器实现方法：对于阻塞的socket 可用int setsockopt(int socket,int level, int option_name, const void* option_value, size_t option_len)函数设置套接字发送与接收超时时间；对于非阻塞socket 可以使用累加sleep时间的方法判断socket 接受数据是否超时(当时间累加量超过一定数值时则认为套接字接受数据超时)。

实验三：

- 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，可以当堂完成或课下完成）
- 4) 将所设计的GBN 协议改进为SR 协议。（选作内容，加分项目，可以当堂完成或课下完成）

实验过程：

以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交。

由于实验三是在实验二的基础上完成的，所以只对实验三进行实验过程的描述。

(1) GBN协议数据分组格式、确认分组格式，各个域的作用

本实验的最终目的是要求实现双向传输，故数据分组和确认分组可以使用同样的分组格式。可以使用如下的分组格式：

分组类型	序号	数据
1 byte	8 bytes	0~1024 byte

各个域的作用如下：

- **分组类型**字段长度为1字节，作用是却别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（在报告中已经指明数据字段 ≤ 1024 个字节），数据长度可以为0

GBN和SR分组的定义在代码中实现如下：

```
class Data: # 定义GBN的分组格式
    def __init__(self, is_ack, seq, data):
```

```

self.is_ack = is_ack # 分组类型, 0 表示数据分组, 1 表示确认分组
self.seq = seq # 分组标记的序号
self.data = data # 数据字段

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

由于需要在GBN的基础上实现SR协议，且SR协议的每个分组都需要有一个计时器，所以在实现SR协议的时候，需要将Data封装在一个新类DataGram中。当在每轮循环中，如果没有收到分组没有收到ack回复，则将其计时器加一。

定义的新类DataGram如下，用于完成SR协议：

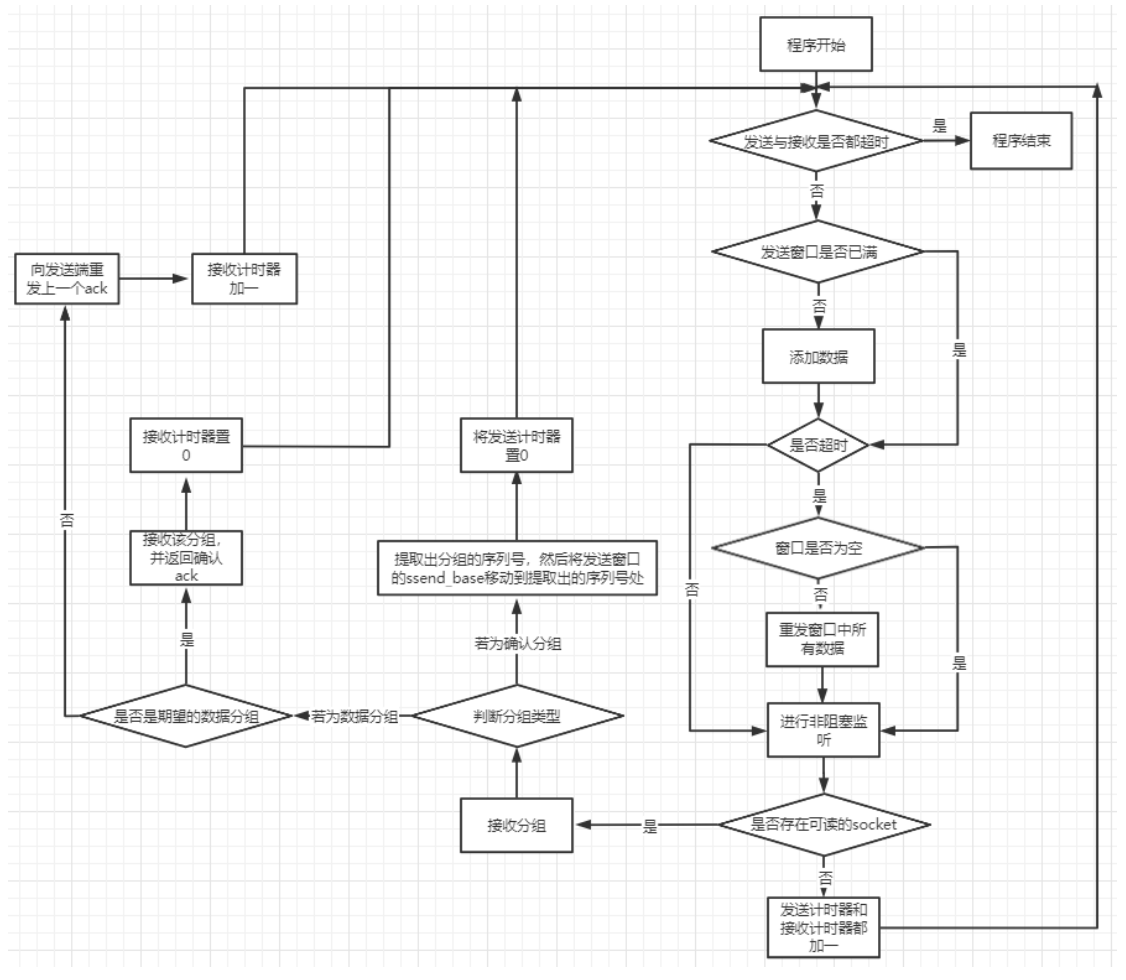
```

class DataGram: # 封装 Data 类
    def __init__(self, pkt, timer=0, is_acked=False):
        self.pkt = pkt
        self.timer = timer # 计时器
        self.is_acked = is_acked # 判断该组是否已经被 ack
    
```

(2) 协议两端程序流程图

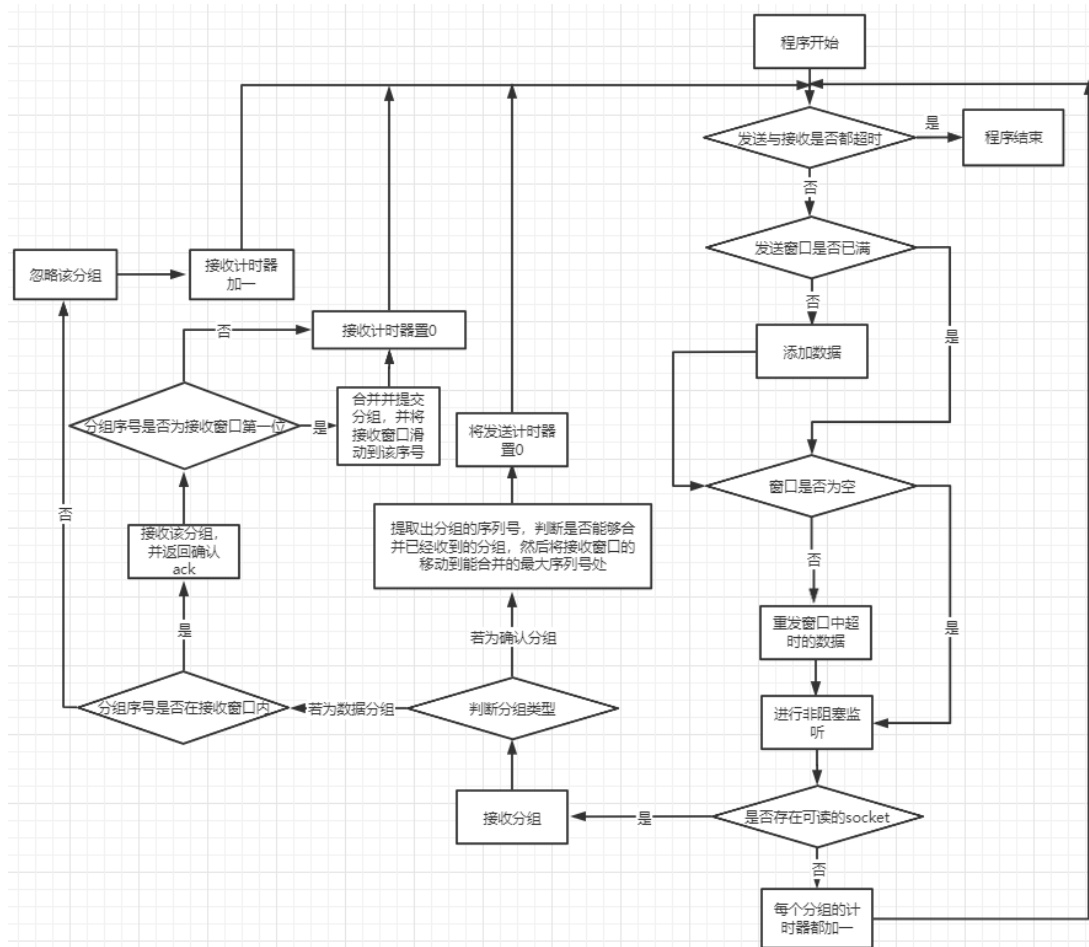
由于需要将单向运输扩展为双向运输，则发送方与接收方的功能应该相同，发送方也是接收方，接收方也是发送方，所以发送方与接收方的流程图应该相同：

GBN协议：



SR协议:

对于SR协议来说, 其与GBN的区别在于SR协议的接收方多维护了一个接收窗口, 当收到序号在发送窗口内的数据分组后, 就会正确接收该数据并返回该序列号的ack, 当收到按序到达的分组后且分组必须从接收窗口第一个分组开始, 然后再向后合并已接收到的分组, 提交给上层协议, 接收窗口向后滑动到没有连续已接收的分组的位置。



(3) 协议典型交互过程

GBN协议:

首先对于发送方来说, 发送方把需要发送的分组加入发送窗口, 然后向接收方发送窗口内的分组。对于接收方来说, 若接收到了期望收到的分组, 则向发送方返回一个该分组的ACK, 否则发送上一个正确接收的分组的ACK。对于发送方来说, 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序列号+1处, 然后进入下一轮循环。这样不断循环便能完成所有的文件发送。

SR协议:

SR协议与GBN协议的不同之处在于拥有一个接收窗口, 因此需要为每个分组都设置一个计时器, 当经过一次循环后, 将重发发送窗口中超时的那些分组。对于接收方来说, 如果收到乱序但序列号在接收窗口内的分组, 则接收方会把它缓存, 然后向发送方返回该分组的ACK。当收到按序到达的分组时, 接收方会检查接收窗口内是否有能合并的已接收的分组, 当这些分组能够从接收窗口的最左端合并时, 先将这些分组合并, 然后一起提交给上层协议, 并滑动接收窗口到未提交给上层协议的位置处。这样不断循环便能完成所有的文件的发送。

(4) 数据分组丢失验证模拟方法

由于很难实现真实的丢包, 所以需要模拟丢包, 具体模拟丢包的方法如下:

对于接收方来说，在每次循环时，设置非阻塞监听。若收到了数据，先调用recvfrom()将该数据包接收，然后调用random()函数，随机产生一个0到1之间的浮点数，通过比较该浮点数与设定的丢包率，就能判断是否丢包。举个例子，设定丢包率为10%，若产生的浮点数<0.1，则跳过此次循环，这样没有处理该数据，实现了模拟了该数据包的丢失。

具体实现如下：

```
while len(rs) > 0: # 若有可读的 socket
    rcv_pkt, address = self.socket.recvfrom(self.buffer_size) # 接收数据
    message = rcv_pkt.decode()
    ack_num = int(message[1:9])
    # 模拟丢包
    if random() < 0.1: # 随机产生一个 0 到 1 之间的浮点数，若小于 0.1，则说明丢包率小于 10%，则不处理该数据
        receive_timer += 1
        send_timer += 1
        print('client 丢包', str(ack_num))
        rs, ws, es = select.select([self.socket, ], [], [], 0.1)
        continue
```

在发生丢包时就会打印丢包信息。

GBN协议：

以下面GBN的单向传输为例：server发送数据包给client。在实验中设置窗口大小为5

```
C:\Users\Administrator\PycharmProjects\network_test2\venv\Scripts
client收到了正确分组，向发送方发送ack 0
client丢包 1
client收到的分组不是期望的分组，向发送方重发期望收到的分组的上一个ack 0
client收到的分组不是期望的分组，向发送方重发期望收到的分组的上一个ack 0
client收到的分组不是期望的分组，向发送方重发期望收到的分组的上一个ack 0
client收到的分组不是期望的分组，向发送方重发期望收到的分组的上一个ack 0
client收到了正确分组，向发送方发送ack 1
client收到了正确分组，向发送方发送ack 2
client收到了正确分组，向发送方发送ack 3
client收到了正确分组，向发送方发送ack 4
client收到了正确分组，向发送方发送ack 5
client收到了正确分组，向发送方发送ack 6
client收到了正确分组，向发送方发送ack 7
client收到了正确分组，向发送方发送ack 8
client收到了正确分组，向发送方发送ack 9
```

```
C:\Users\Administrator\PycharmPro
server发送了pkt0
server发送了pkt1
server发送了pkt2
server发送了pkt3
server发送了pkt4
Server收到了ack 0
此时窗口起点位置为: 1
Server收到了ack 0
此时窗口起点位置为: 1
Server收到了ack 0
此时窗口起点位置为: 1
Server收到了ack 0
此时窗口起点位置为: 1
server发送了pkt5
Server收到了ack 0
此时窗口起点位置为: 1
server发送超时,需要重传
server重传的数据包为 1
server重传的数据包为 2
server重传的数据包为 3
server重传的数据包为 4
server重传的数据包为 5
Server收到了ack 1
此时窗口起点位置为: 2
Server收到了ack 2
此时窗口起点位置为: 3
Server收到了ack 3
此时窗口起点位置为: 4
Server收到了ack 4
此时窗口起点位置为: 5
```

分析可以看出:

- server一开始正确发送了pkt0, pkt1, pkt2, pkt3, pkt4
- client收到了第一个包pkt0, 首先发送ack0给server, 然后client丢了第二个包, 所以对于后续收到的pkt1, pkt2, pkt3, pkt4, 都需要发送ack0(一共四个)给server
- server在收到第一个确认收到的ack0后, 窗口向后移动一位, 此时send_base为1, 然后server期望收到ack1, 但实际却收到四个ack0, 当计时器超时后, 选择重发窗口内所有的数据包pkt1, pkt2, pkt3, pkt4, pkt5

SR协议:

以下面SR的单向传输为例: server发送数据包给client。在实验中设置窗口大小为5

```
C:\Users\Administrator\PycharmProjects\network_test2\venv\Scripts\python.  
请输入命令: -testsr  
client收到了正确分组, 向发送方发送ack      0  
client 向上层提交数据: 0  
client收到了正确分组, 向发送方发送ack      1  
client 向上层提交数据: 1  
client收到了正确分组, 向发送方发送ack      2  
client 向上层提交数据: 2  
client丢包  3  
client收到的分组不是期望的分组, 但在接收窗口内, 向发送方发送该分组对应的ack      4  
client收到的分组不是期望的分组, 但在接收窗口内, 向发送方发送该分组对应的ack      5  
client收到的分组不是期望的分组, 但不在接收窗口内, 向发送方发送该分组对应的ack      1  
client收到了正确分组, 向发送方发送ack      3  
client 向上层提交数据: 3 到 5  
client收到了正确分组, 向发送方发送ack      6  
client 向上层提交数据: 6  
client收到了正确分组, 向发送方发送ack      7  
client 向上层提交数据: 7  
client收到了正确分组, 向发送方发送ack      8  
client 向上层提交数据: 8  
client收到了正确分组, 向发送方发送ack      9  
client 向上层提交数据: 9  
client收到了正确分组, 向发送方发送ack     10  
client 向上层提交数据: 10  
client收到了正确分组, 向发送方发送ack     11  
client 向上层提交数据: 11
```

```

C:\Users\Administrator\PycharmPro
server发送了pkt0
server发送了pkt1
server发送了pkt2
server发送了pkt3
server发送了pkt4
server收到ack0
此时窗口起点位置为: 1
server丢包 1
server收到ack2
server收到ack4
server发送了pkt5
server收到ack5
server重传的数据包为      1
server重传的数据包为      3
server收到ack1
此时窗口起点位置为: 3
server收到ack3
此时窗口起点位置为: 6
server发送了pkt6
server发送了pkt7
server发送了pkt8
server发送了pkt9
server发送了pkt10

```

分析可以看出:

- server一开始正确发送了pkt0, pkt1, pkt2, pkt3, pkt4
- client收到了pkt0, pkt1, pkt2, 并向上层提交数据, 然后向server返回ack0, ack1, ack2, 但是server丢包了ack1, 这样snd_base就只能往后移动一位, 然后client丢包pkt3, 但却收到了接收窗口存在的pkt4, pkt5, pkt3是接收窗口的第一位, 所以接收窗口不滑动。当server超时后, 就会重传pkt3, 这样接收窗口会一次性向后移动三位, 因为缺失的pkt3已经收到了。

(5) 程序实现的主要类(或函数)及其主要作用

主要类:

Data类:

```

class Data: # 定义GBN的分组格式
    def __init__(self, is_ack, seq, data):
        self.is_ack = is_ack # 分组类型, 0表示数据分组, 1表示确认分组
        self.seq = seq # 分组标记的序号
        self.data = data # 数据字段

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

```

主要作用:

用来定义GBN和SR的分组格式，分别储存了分组类型、分组标记的序列号和具体储存的数据字段，对于所有需要发送的数据，发送方都会将其封装在Data类中，并且将其添加到发送窗口然后发送。

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', 4321) # 发送方地址
        self.client_address = ('127.0.0.1', 1234) # 接收方地址
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# 新建一个 socket
        self.socket.bind(self.address) # 绑定到本地固定窗口
        self.send_window = [] # 发送窗口
        self.receive_buffer = [] # 接收缓冲区
        self.buffer_size = 1024 # 缓冲区大小
```

主要作用:

GBNServer类是用来模拟GBN协议的Server，通过初始化GBNServer会创建一个socket，并将其绑定到本地固定端口，接下来将通过该socket进行所有分组的发送和接收。GBNServer还包括窗口大小、发送超时时间、接收超时时间，发送方地址，接收方地址，发送窗口、接收缓冲区以及缓冲区大小等属性。由于实现的是双向传输，所有Server既充当发送方又充当接收方。

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', 1234) # 发送方地址
        self.server_address = ('127.0.0.1', 4321) # 接收方地址
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# 新建一个 socket
        self.socket.bind(self.address) # 绑定到本地固定窗口
        self.send_window = [] # 发送窗口
        self.receive_buffer = [] # 接收缓冲区
        self.buffer_size = 1024 # 缓冲区大小
```

主要作用:

功能与GBNServer作用相同，即充当发送方又充当接收方。

DataGram类:

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

```
self.timer = timer # 计时器
self.is_acked = is_acked # 判断该组是否已经被 ack
```

主要作用:

该类储存在SR协议中,目的是为了封装SR协议传输的Data类,因为在SR协议中需要为每个分组都设置一个计时器。所以创建DataGram类,其中的字段timer用作计时器, is_acked用来标记该分组是否已经被接收,只有超时并且已经is_acked==False的分组才能被重发

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', 8765) # 发送方地址
        self.client_address = ('127.0.0.1', 5678) # 接收方地址
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# 新建一个 socket
        self.socket.bind(self.address) # 绑定到本地固定窗口
        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 # 缓冲区大小
```

主要作用:

与GBNServer相似,但又相对GBNServer增加了接收窗口、接收窗口大小字段、用来模拟上层协议缓存的receive_data,需要将提交给上层的协议首先存在receive_data。由于实现的是双向传输,所有Server既充当发送方又充当接收方。

SRClient类:

```
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', 5678) # 发送方地址
        self.server_address = ('127.0.0.1', 8765) # 接收方地址
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# 新建一个 socket
        self.socket.bind(self.address) # 绑定到本地固定窗口
        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 # 缓冲区大小
```

主要作用:

功能与SRServer作用相同，即充当发送方又充当接收方。

主要函数：

`send_and_receive()`：(GBN协议中)

主要作用：

用来实现服务器端以及客户端的发送和接收，从而实现双向传输。

具体实现：

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。所有的模块如下所示：

将接收缓冲区的数据全部写入文件：

```
while True:
    if not self.send_window and receive_timer >
self.max_receive_time: # 如果没有超过接收超时时间
        with open('server_receive.txt', 'w') as f:
            for data in self.receive_buffer: # 将接收缓冲区的数据全部写入文件
                f.write(data)
            break
```

发送阶段：对于发送窗口正常发送数据和需要重传数据的情况进行处理（由详细注释给出处理过程）

```
# 当发送窗口未满足且有数据还需要发送时，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]) # 从缓冲区取出需要发送的数据
    self.socket.sendto(str(pkt).encode(), self.client_address) #
server 发送数据
    print('server 发送了 pkt ' + str(next_seq_num)) # 打印发送信息
    self.send_window.append(pkt) # 将数据加入发送窗口
    if send_base == next_seq_num: # 只要该数据的序列号不是起点就将序列号加一
        send_timer = 0
    next_seq_num = next_seq_num + 1

# 若发送的时间超过给定的发送超时时间，则重传发送窗口中的数据
if send_timer > self.max_send_time and self.send_window:
    print('server 发送超时, 需要重传')
    send_timer = 0 # 发送计时器重新置 0
    for pkt in self.send_window: # 对于发送窗口中所有 pkt 数据
        self.socket.sendto(str(pkt).encode(), self.client_address) #
重传数据
        print('server 重传的数据包为 ' + str(pkt.seq))
```

接收阶段：对于收到的分组是确认分组还是数据分组两种情况进行处理（由详细注释给出处理过程）

```

if message[0] == '1': # 如果收到的分组是确认分组
    ack_num = int(message[1:9]) # 得到确认序列号(8位)
    print('server 收到 ack: ' + str(ack_num))
    for i in range(len(self.send_window)): # 遍历发送窗口
        if ack_num == int(self.send_window[i].seq): # 如果发送窗口找到了
            这个确认序列号
                self.send_window = self.send_window[i + 1:] # 将窗口后移
                break
    send_base = ack_num + 1 # 窗口起点序列号更新
    print("此时窗口起点位置为: ", send_base)
    send_timer = 0 # 发送计时器置 0
elif message[0] == '0': # 如果收到的分组是数据分组
    rcv_seq_num = message[1:9] # 得到收到分组的序列号(8位)
    if int(rcv_seq_num) == expected_num: # 如果是期望收到的分组
        print('server 收到了正确分组, 向发送方发送 ack' + rcv_seq_num) # 打印信息
        self.receive_buffer.append(rcv_pkt.decode()[9:]) # 将分组中的数据放入接收缓冲区中(9位以后的数据)
        ack_pkt = Data(1, '%8d ' % expected_num, '') # 创建该 pkt 的 ack 分组
        self.socket.sendto(str(ack_pkt).encode(), self.client_address)
# 向发送方发送 ack 分组
        last_ack = expected_num # 储存最新发送的 ack 号
        expected_num += 1 # 下一次期望分组的序列号加 1
    else: # 如果不是期望收到的分组
        print('server 收到的分组不是期望的分组, 向发送方重发期望收到的分组的上一个 ack', last_ack) # 打印信息
        ack_pkt = Data(1, '%8d ' % last_ack, '') # 创建该 pkt 的 ack 分组
        self.socket.sendto(str(ack_pkt).encode(), self.client_address)
# 向发送方重新发送 ack 分组

```

send_and_receive(): (SR协议中)

与GBN协议有部分不同, 主要在接收阶段的处理。对于两种分组都需要进行已收到分组的合并, 只有收到窗口中第一个分组后, 才能将合并分组后的窗口后移(由详细注释给出处理过程)

```

if not self.send_window and receive_timer > self.max_receive_time: #
    如果没有超过接收超时时间
        with open('client_receive.txt', 'w') as f:
            for data in self.receive_data: # 将 receive_data 的数据全部写入文件
                f.write(data)
            break
# 发送阶段
# 当发送窗口未满足且有数据还需要发送时, 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]) # 从缓冲区取出需要发送的数据
    self.socket.sendto(str(pkt).encode(), self.server_address) # client 发送数据
    print('client 发送了 pkt' + str(next_seq_num)) # 打印发送信息
    self.send_window.append(DataGram(pkt)) # 给每个分组计时器初始化为 0, 把数据报加入发送窗口
    next_seq_num = next_seq_num + 1 # 将 next_seq_num 序列号加一

# 遍历所有已发送但未确认的分组, 如果有超时的分组, 则重发该分组
for dgram in self.send_window:
    if dgram.timer > self.max_send_time and not dgram.is_acked: # 如果已经超时且该组没有被接收
        self.socket.sendto(str(dgram.pkt).encode(), self.server_address) # 重发该分组
        print('server 重传的数据包为' + str(dgram.pkt.seq))

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

while len(rs) > 0: # 若有可读的 socket
    rcv_pkt, address = self.socket.recvfrom(self.buffer_size) # 接收数据
    message = rcv_pkt.decode()
    ack_num = int(message[1:9])
    # 模拟丢包
    if random() < 0.1: # 随机产生一个 0 到 1 之间的浮点数, 若小于 0.1, 则说明丢包率小于 10%, 则不处理该数据
        for dgram in self.send_window:
            dgram.timer += 1
        receive_timer += 1
        print('client 丢包', str(ack_num))
        rs, ws, es = select.select([self.socket, ], [], [], 0.01)
        continue
    receive_timer = 0 # 接收计时器重新置 0
    message = rcv_pkt.decode() # 将 rcv_pkt 解码为字符串编码

    if message[0] == '1': # 如果收到的分组是确认分组
        ack_num = int(message[1:9]) # 得到确认序列号(8 位)
        print('client 收到 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: # 如果找到了 ack_num
```

```

        dgram.timer = 0 # 计时器置0
        dgram.is_acked = True # 标记为已收到
        if self.send_window.index(dgram) == 0: # 如果ack的是发送窗口中的第一个分组
            idx = -1 # 用idx表示窗口中最后一个已确认的分组的下标
            for i in self.send_window: # 依次遍历发送窗口中已确认的分组
                if i.is_acked: # 记录从第一个分组开始连续分组的个数
                    idx += 1
                else: # 不连续则退出
                    break
            send_base = int(self.send_window[idx].pkt.seq) + 1
            print("此时窗口起点位置为: ", send_base)
            self.send_window = self.send_window[idx+1:] # 窗口滑动到最后一个连续已确认的分组序号之后
            break
        elif message[0] == '0': # 如果收到的分组是数据分组
            for dgram in self.send_window: # 发送窗口中所有分组计时器+1
                dgram.timer += 1
            rcv_seq_num = message[1:9] # 得到收到分组的序列号(8位)
            if int(rcv_seq_num) == expected_num: # 如果是期望收到的分组
                print('client 收到了正确分组, 向发送方发送ack' + str(rcv_seq_num)) # 打印信息
                ack_pkt = Data(1, '%8d' % expected_num, '') # 创建该pkt的ack分组
                self.socket.sendto(str(ack_pkt).encode(), self.server_address) # 向发送方发送ack分组
                # 再看它后面有没有能合并的分组
                self.receive_window[int(rcv_seq_num)] = rcv_pkt # 将最新收到的数据序列号加给接收窗口
                tmp = [(k, self.receive_window[k]) for k in sorted(self.receive_window.keys())] # 按照序列号对接收窗口进行排序
                idx = 0 # idx为接受窗口中能合并到的最后一个分组, 为连续数对的个数
                for i in range(len(tmp) - 1):
                    if tmp[i + 1][0] - tmp[i][0] == 1: # 如果两个序列号相差为1
                        idx += 1
                    else: # 相差不为1跳出(不连续)
                        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
                # 记录提交的分组的序号
                base = int(tmp[0][1].decode()[1:9]) # 写入receive_data的序列号起点

```

```

        end = int(tmp[idx][1].decode()[1:9]) # 写入 receive_data 的序
列号终点
        if base != end:
            print('client 向上层提交数据: ' + str(base) + ' 到 ' +
str(end))
        else:
            print('client 向上层提交数据: ' + str(base))
            expected_num = tmp[idx][0]+1 # 下一次期望分组的序列号加 1
            tmp = tmp[idx + 1:] # 接收窗口滑动
            self.receive_window = dict(tmp)
        else: # 如果不是期望收到的分组
            if expected_num <= int(rcv_seq_num) < expected_num +
self.receive_window_size - 1: # 如果 rcv_seq_num 在接收窗口内, 即若在
rcv_base~rcv_base + N -1 之内, 则加入接收窗口
                self.receive_window[int(rcv_seq_num)] = rcv_pkt # 加入接
收窗口
                ack_pkt = Data(1, '%8d ' % int(rcv_seq_num), '') # 创建
该 pkt 的 ack 分组
                print('client 收到的分组不是期望的分组, 但在接收窗口内, 向发送方
发送该分组对应的 ack' + str(rcv_seq_num)) # 打印信息
                self.socket.sendto(str(ack_pkt).encode(),
self.server_address) # 向发送方发送 ack 分组
                elif int(rcv_seq_num) < expected_num: # 如果收到的分组不在接收
窗口内 (窗口前)
                    ack_pkt = Data(1, '%8d ' % int(rcv_seq_num), '') # 创建
该 pkt 的 ack 分组
                    print('client 收到的分组不是期望的分组, 但不在接收窗口内, 向发送
方发送该分组对应的 ack' + str(rcv_seq_num))
                    self.socket.sendto(str(ack_pkt).encode(),
self.server_address) # 向发送方发送 ack 分组
                else:
                    print('丢弃')
                    pass
            else:
                pass
            rs, ws, es = select.select([self.socket, ], [], [], 0.01) # 进行非
阻塞监听
        else: # 非阻塞监听未收到
            receive_timer += 1
            for dgram in self.send_window:
                dgram.timer += 1

```

(6) 详细注释源程序 (在报告最后给出)

实验结果:

采用演示截图、文字说明等方式，给出本次实验的实验结果。

1.初始化: 有四个文本文件: client_receive.txt, client_send.txt, server_receive.txt, server_send.txt
在client_send.txt储存了client需要发送的文本文件:



在server_send.txt储存了client需要发送的文本文件:

server_send.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

《以父之名》

微凉的晨露沾湿黑礼服，石板路有雾父在低诉
无奈的觉悟只能更残酷，一切都为了通往圣堂的路
吹不散的雾隐没了意图，谁轻柔踱步停住
还来不及哭穿过的子弹就带走温度，我们每个人都有罪
犯着不同的罪，我能决定谁对
谁又该要沉睡，争论不能解决
在永无止境的夜，关掉你的嘴
唯一的恩惠，挡在前面的人都有罪
后悔也无路可退，以父之名判决
那感觉没有适合词汇，就像边笑边掉泪
凝视着完全的黑，阻挡悲剧蔓延的悲剧会让我沉醉
低头亲吻我的左手，换取被宽恕的承诺
老旧管风琴在角落，一直一直一直伴奏
黑色帘幕被风吹动，阳光无言地穿透
洒向那群被我驯服后的兽，沉默地喊叫沉默地喊叫
孤单开始发酵，不停对着我嘲笑
回忆逐渐燃烧，曾经纯真的画面
残忍地温柔出现，脆弱时间到
我们一起来祷告，仁慈的父我已坠入

第 1 行, 第 1 列

100%

Window

此时的client_receive.txt, server_receive.txt中为空:

client_receive.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

server_receive.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

2.输入-time显示当前时间:

```
C:\Users\Administrator\PycharmProjects\1180
请输入命令: -time
请输入命令: 2020-11-10 21:01:53.670650
```

3.输入-testgbn测试GBN双向传输:

```
C:\Users\Administrator\PycharmProjects\1180300829_network_lab2\venv\Scripts\python.exe C:/Users/Administ
请输入命令: -testgbn
client发送了pkt 0
server发送了pkt 0
client发送了pkt 1
server发送了pkt 1
client发送了pkt 2
server发送了pkt 2
server发送了pkt 3
client发送了pkt 3
server发送了pkt 4
client发送了pkt 4
server收到了正确分组, 向发送方发送ack      0client丢包
0
client收到的分组不是期望的分组, 向发送方重发期望收到的分组的上一个ack -1
server收到了正确分组, 向发送方发送ack      1
client收到的分组不是期望的分组, 向发送方重发期望收到的分组的上一个ack server收到了正确分组, 向发送方发送ack      2-1
client收到的分组不是期望的分组, 向发送方重发期望收到的分组的上一个ack -1
server收到了正确分组, 向发送方发送ack      3
client收到的分组不是期望的分组, 向发送方重发期望收到的分组的上一个ack -1server收到了正确分组, 向发送方发送ack      4
client丢包 server收到ack: -10
此时窗口和卡位置如左图所示
```


此时client_receive.txt, server_receive.txt分别收到了server_send.txt, client_send.txt传来的文件:

client_receive.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

《以父之名》

微凉的晨露沾湿黑礼服，石板路有雾父在低诉
无奈的觉悟只能更残酷，一切都为了通往圣堂的路
吹不散的雾隐没了意图，谁轻柔踱步停住
还来不及哭穿过的子弹就带走温度，我们每个人都有罪
犯着不同的罪，我能决定谁对
谁又该要沉睡，争论不能解决
在永无止境的夜，关掉你的嘴
唯一的恩惠，挡在前面的人都有罪
后悔也无路可退，以父之名判决
那感觉没有适合词汇，就像边笑边掉泪
凝视着完全的黑，阻挡悲剧蔓延的悲剧会让我沉醉
低头亲吻我的左手，换取被宽恕的承诺
老旧管风琴在角落，一直一直一直伴奏
黑色帘幕被风吹动，阳光无言地穿透
洒向那群被我驯服后的兽，沉默地喊叫沉默地喊叫
孤单开始发酵，不停对着我嘲笑
回忆逐渐燃烧，曾经纯真的画面
残忍地温柔出现，脆弱时间到
我们一起来祷告，仁慈的父我已坠入

 *server_receive.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

| 《听妈妈的话》

小朋友

你是否有很多问号

为什么

别人在那看漫画

我却在学画画对着钢琴说话

别人在玩游戏

我却靠在墙壁背我的ABC

我说我要一台大大的飞机

但却得到一台旧旧录音机

为什么要听妈妈的话

长大后你就会

开始懂了这段话

哼

长大后我开始明白

为什么我跑得比别人快

飞得比别人高

将来大家看的都是我

画的漫画

大家唱的都是我写的歌

4.输入-quit测试客户端退出:

C:\Users\Administrator\PycharmProjects\11803

请输入命令: `-quit`

请输入命令: Good bye!

5.输入-testsr测试SR双向传输:

```
C:\Users\Administrator\PycharmProjects\1180300829_network_lab2\venv\Scripts\python
请输入命令: -tester
client发送了pkt0
server发送了pkt0
client发送了pkt1
server发送了pkt1
client发送了pkt2
server发送了pkt2
server发送了pkt3
client发送了pkt3
server发送了pkt4
client发送了pkt4server收到了正确分组, 向发送方发送ack      0

server 向上层提交数据: 0
client收到了正确分组, 向发送方发送ack      0
client 向上层提交数据: 0
server收到了正确分组, 向发送方发送ack      1client收到了正确分组, 向发送方发送ack      1

client 向上层提交数据: 1
server 向上层提交数据: 1
client收到了正确分组, 向发送方发送ack      2
server收到了正确分组, 向发送方发送ack      2
client 向上层提交数据: 2
```

client_receive.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

《以父之名》

微凉的晨露沾湿黑礼服，石板路有雾父在低诉
 无奈的觉悟只能更残酷，一切都为了通往圣堂的路
 吹不散的雾隐没了意图，谁轻柔踱步停住
 还来不及哭穿过的子弹就带走温度，我们每个人都有罪
 犯着不同的罪，我能决定谁对
 谁又该要沉睡，争论不能解决
 在永无止境的夜，关掉你的嘴
 唯一的恩惠，挡在前面的人都有罪
 后悔也无路可退，以父之名判决
 那感觉没有适合词汇，就像边笑边掉泪
 凝视着完全的黑，阻挡悲剧蔓延的悲剧会让我沉醉
 低头亲吻我的左手，换取被宽恕的承诺
 老旧管风琴在角落，一直一直一直伴奏
 黑色帘幕被风吹动，阳光无言地穿透
 洒向那群被我驯服后的兽，沉默地喊叫沉默地喊叫
 孤单开始发酵，不停对着我嘲笑
 回忆逐渐燃烧，曾经纯真的画面
 残忍地温柔出现，脆弱时间到
 我们一起来祷告，仁慈的父我已坠入

*server_receive.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

| 《听妈妈的话》

小朋友

你是否有很多问号

为什么

别人在那看漫画

我却在学画画对着钢琴说话

别人在玩游戏

我却靠在墙壁背我的ABC

我说我要一台大大的飞机

但却得到一台旧旧录音机

为什么要听妈妈的话

长大后你就会

开始懂了这段话

哼

长大后我开始明白

为什么我跑得比别人快

飞得比别人高

将来大家看的都是我

画的漫画

大家唱的都是我写的歌

此时client_receive.txt, server_receive.txt分别收到了server_send.txt, client_send.txt传来的文件:

问题讨论:

对实验过程中的思考问题进行讨论或回答。

- 1.开始不知道怎么引入丢包机制，后来通过查阅相关资料学会了可以通过引入一个概率因子，然后不处理当前收到的数据包即可模拟丢包机制。
- 2.对于双向数据传输的实现，实际就是此时服务器也是客户机，客户机也是服务器，两个的作用相同，都能实现文件的发送和传输。
- 3.SR协议的重传机制相对GBN更加复杂，但却效率更高，因为不需要重传发送窗口的所有文件。

心得体会:

结合实验过程和结果给出实验的体会和收获。

通过此次实验我了解并掌握了GBN传输协议和SR传输协议完成文件传输的工作原理，掌握了丢包的重传机制，能够通过分析传输协议的输出来模拟实际各文件资源传输的具体过程，学会了基于UDP设计并实现一个GBN协议的过程和技术，也能够通过改进单向传输实现GBN协议和SR协议的双向传输。

注：详细注释源程序：

GBN_client.py

```
1. import socket
2. from random import random
```

```
3. import select
4.
5.
6. class Data: # 定义GBN的分组格式
7.     def __init__(self, is_ack, seq, data):
8.         self.is_ack = is_ack # 分组类型, 0表示数据分组, 1表示确认分组
9.         self.seq = seq # 分组标记的序号
10.        self.data = data # 数据字段
11.
12.    def __str__(self):
13.        return str(self.is_ack) + str(self.seq) + str(self.data)
14.
15.
16. class GBNCClient:
17.    def __init__(self):
18.        self.window_size = 5 # 窗口大小
19.        self.max_send_time = 3 # 发送超时时间
20.        self.max_receive_time = 10 # 接收超时时间
21.        self.address = ('127.0.0.1', 1234) # 发送方地址
22.        self.server_address = ('127.0.0.1', 4321) # 接收方地址
23.        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # 新建一个socket
24.        self.socket.bind(self.address) # 绑定到本地固定窗口
25.        self.send_window = [] # 发送窗口
26.        self.receive_buffer = [] # 接收缓冲区
27.        self.buffer_size = 1024 # 缓冲区大小
28.
29.        ....
30.        发送和接收功能
31.        ...
32.    def send_and_receive(self, buffer):
33.        send_timer = 0 # 发送计时器
34.        send_base = 0 # 窗口此时起点的序列号
35.        next_seq_num = send_base # 下一个待发送的序列号
36.        expected_num = 0 # 希望收到的序列号
37.        receive_timer = 0 # 接收计时器
38.        last_ack = -1 # 上一个ack序号
39.        total = len(buffer) # 缓冲区大小
40.        while True:
41.            if not self.send_window and receive_timer > self.max_receive_time:
42.                # 如果没有超过接收超时时间
43.                with open('client_receive.txt', 'w') as f:
44.                    for data in self.receive_buffer: # 将接收缓冲区的数据全部
45.                        写入文件
```

```
44.         f.write(data)
45.         break
46.
47.         # 发送阶段
48.         # 当发送窗口未满且有数据还需要发送时，发送数据
49.         while next_seq_num < send_base + self.window_size and next_seq_num < total:
50.             pkt = Data(0, '%8d' % next_seq_num, buffer[next_seq_num]) # 从缓冲区取出需要发送的数据
51.             self.socket.sendto(str(pkt).encode(), self.server_address)
52.             # client 发送数据
53.             print('client 发送了 pkt ' + str(next_seq_num)) # 打印发送信息
54.             self.send_window.append(pkt) # 将数据加入发送窗口
55.             if send_base == next_seq_num: # 只要该数据的序列号不是起点就将序列号加一
56.                 send_timer = 0
57.                 next_seq_num = next_seq_num + 1
58.             # 若发送的时间超过给定的发送超时时间，则重传发送窗口中的数据
59.             if send_timer > self.max_send_time and self.send_window:
60.                 print('client 发送超时,需要重传')
61.                 send_timer = 0 # 发送计时器重新置0
62.                 for pkt in self.send_window: # 对于发送窗口中所有 pkt 数据
63.                     self.socket.sendto(str(pkt).encode(), self.server_addresses) # 重传数据
64.                     print('server 重传的数据包为 ' + str(pkt.seq))
65.
66.         # 接收阶段
67.         rs, ws, es = select.select([self.socket, ], [], [], 0.1) # 进行非阻塞监听
68.
69.         while len(rs) > 0: # 若有可读的 socket
70.             rcv_pkt, address = self.socket.recvfrom(self.buffer_size) # 接收数据
71.             message = rcv_pkt.decode()
72.             ack_num = int(message[1:9])
73.             # 模拟丢包
74.             if random() < 0.1: # 随机产生一个 0 到 1 之间的浮点数，若小于 0.1，则说明丢包率小于 10%，则不处理该数据
75.                 receive_timer += 1
76.                 send_timer += 1
77.                 print('client 丢包', str(ack_num))
```



```
78.             rs, ws, es = select.select([self.socket, ], [], [], 0.1)
79.             continue
80.             message = rcv_pkt.decode() # 将 rcv_pkt 解码为字符串编码
81.             receive_timer = 0 # 接收计时器重新置 0
82.
83.             if message[0] == '1': # 如果收到的分组是确认分组
84.                 ack_num = int(message[1:9]) # 得到确认序列号(8 位)
85.                 print('Client 收到 ack: ' + str(ack_num))
86.                 for i in range(len(self.send_window)): # 遍历发送窗口
87.                     if ack_num == int(self.send_window[i].seq): # 如果发
                        送窗口找到了这个确认序列号
88.                         self.send_window = self.send_window[i + 1:] #
                        将窗口后移
89.                         break
90.                 send_base = ack_num + 1 # 窗口起点序列号更新
91.                 send_timer = 0 # 发送计时器置 0
92.                 elif message[0] == '0': # 如果收到的分组是数据分组
93.                     rcv_seq_num = message[1:9] # 得到收到分组的序列号(8
                        位)
94.                     if int(rcv_seq_num) == expected_num: # 如果是期望收到的分
                        组
95.                         print('client 收到了正确分组, 向发送方发送
                        ack ' + rcv_seq_num) # 打印信息
96.                         self.receive_buffer.append(rcv_pkt.decode()[9:]) #
                        将分组中的数据放入接收缓冲区中(9 位以后的数据)
97.                         ack_pkt = Data(1, '%8d ' % expected_num, '') # 创建
                        该 pkt 的 ack 分组
98.                         self.socket.sendto(str(ack_pkt).encode(), self.serve
                        r_address) # 向发送方发送 ack 分组
99.                         last_ack = expected_num # 储存最新发送的 ack 号
100.                        expected_num += 1 # 下一次期望分组的序列号加 1
101.                        else: # 如果不是期望收到的分组
102.                            print('client 收到的分组不是期望的分组, 向发送方重发期
                            望收到的分组的上一个 ack ', last_ack) # 打印信息
103.                            ack_pkt = Data(1, '%8d ' % last_ack, '') # 创建该
                            pkt 的 ack 分组
104.                            self.socket.sendto(str(ack_pkt).encode(), self.serv
                            er_address) # 向发送方重新发送 ack 分组
105.                        else:
106.                            pass
107.                        rs, ws, es = select.select([self.socket, ], [], [], 0.1) #
                        进行非阻塞监听
108.                        else: # 非阻塞监听未收到
```

```
109.         receive_timer += 1
110.         send_timer += 1
111.
112.
113. def start():
114.     client_socket = GBNClient()
115.     data = []
116.     with open('client_send.txt', 'r') as f: # 打开发送内容的文件
117.         while True:
118.             pkt = f.read(10)
119.             if len(pkt) > 0: # 读取文件并储存在 data 中, data 是需要发送的文件
                内容
120.                 data.append(pkt)
121.             else:
122.                 break
123.     while True:
124.         temp = input("请输入命令: ")
125.         if temp == '-time':
126.             client_socket.socket.sendto('-
time'.encode(), client_socket.server_address)
127.         if temp == '-testgbn':
128.             client_socket.socket.sendto('-
testgbn'.encode(), client_socket.server_address)
129.             client_socket.send_and_receive(data)
130.         if temp == '-quit':
131.             client_socket.socket.sendto('-
quit'.encode(), client_socket.server_address)
132.
133.
134. if __name__ == '__main__':
135.     start()
```

GBN_server.py

```
1. import socket
2. from random import random
3. import select
4. import datetime
5.
6.
7. class Data: # 定义 GBN 的分组格式
8.     def __init__(self, is_ack, seq, data):
9.         self.is_ack = is_ack # 分组类型, 0 表示数据分组, 1 表示确认分组
10.        self.seq = seq # 分组标记的序号
11.        self.data = data # 数据字段
```

```
12.
13.     def __str__(self):
14.         return str(self.is_ack) + str(self.seq) + str(self.data)
15.
16.
17. class GBNServer:
18.     def __init__(self):
19.         self.window_size = 5 # 窗口大小
20.         self.max_send_time = 3 # 发送超时时间
21.         self.max_receive_time = 10 # 接收超时时间
22.         self.address = ('127.0.0.1', 4321) # 发送方地址
23.         self.client_address = ('127.0.0.1', 1234) # 接收方地址
24.         self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # 新建一个 socket
25.         self.socket.bind(self.address) # 绑定到本地固定窗口
26.         self.send_window = [] # 发送窗口
27.         self.receive_buffer = [] # 接收缓冲区
28.         self.buffer_size = 1024 # 缓冲区大小
29.
30.         ....
31.         发送和接收功能
32.         '''
33.     def send_and_receive(self, buffer):
34.         send_timer = 0 # 发送计时器
35.         send_base = 0 # 窗口此时起点的序列号
36.         next_seq_num = send_base # 下一个待发送的序列号
37.         expected_num = 0 # 希望收到的序列号
38.         receive_timer = 0 # 接收计时器
39.         last_ack = -1 # 上一个 ack 序号
40.         total = len(buffer) # 缓冲区大小
41.         while True:
42.             if not self.send_window and receive_timer > self.max_receive_time: # 如果没有超过接收超时时间
43.                 with open('server_receive.txt', 'w') as f:
44.                     for data in self.receive_buffer: # 将接收缓冲区的数据全部
45.                         f.write(data)
46.                     break
47.
48.             # 发送阶段
49.             # 当发送窗口未满足且有数据还需要发送时，发送数据
50.             while next_seq_num < send_base + self.window_size and next_seq_num < total:
```

```
51.         pkt = Data(0, '%8d' % next_seq_num, buffer[next_seq_num]) #
           从缓冲区取出需要发送的数据
52.         self.socket.sendto(str(pkt).encode(), self.client_address)
           # server 发送数据
53.         print('server 发送了 pkt ' + str(next_seq_num)) # 打印发送信
           息
54.         self.send_window.append(pkt) # 将数据加入发送窗口
55.         if send_base == next_seq_num: # 只要该数据的序列号不是起点就将
           序列号加一
56.             send_timer = 0
57.             next_seq_num = next_seq_num + 1
58.
59.         # 若发送的时间超过给定的发送超时时间, 则重传发送窗口中的数据
60.         if send_timer > self.max_send_time and self.send_window:
61.             print('server 发送超时, 需要重传')
62.             send_timer = 0 # 发送计时器重新置 0
63.             for pkt in self.send_window: # 对于发送窗口中所有 pkt 数据
64.                 self.socket.sendto(str(pkt).encode(), self.client_addresses) # 重传数据
65.                 print('server 重传的数据包为 ' + str(pkt.seq))
66.
67.         # 接收阶段
68.         rs, ws, es = select.select([self.socket, ], [], [], 0.1) # 进行
           非阻塞监听
69.
70.         while len(rs) > 0: # 若有可读的 socket
71.             rcv_pkt, address = self.socket.recvfrom(self.buffer_size) #
           接收数据
72.             # 模拟丢包
73.             message = rcv_pkt.decode()
74.             ack_num = int(message[1:9])
75.             if random() < 0.1: # 随机产生一个 0 到 1 之间的浮点数, 若小于
           0.1, 则说明丢包率小于 10%, 则不处理该数据
76.                 send_timer += 1
77.                 receive_timer += 1
78.                 print('server 丢包'+str(ack_num))
79.                 rs, ws, es = select.select([self.socket, ], [], [], 0.1)
80.                 continue
81.             message = rcv_pkt.decode() # 将 rcv_pkt 解码为字符串编码
82.             receive_timer = 0 # 接收计时器重新置 0
83.
84.             if message[0] == '1': # 如果收到的分组是确认分组
85.                 ack_num = int(message[1:9]) # 得到确认序列号(8 位)
```

```
86.         print('server 收到 ack: ' + str(ack_num))
87.         for i in range(len(self.send_window)): # 遍历发送窗口
88.             if ack_num == int(self.send_window[i].seq): # 如果发
                送窗口找到了这个确认序列号
89.                 self.send_window = self.send_window[i + 1:] #
                将窗口后移
90.                 break
91.                 send_base = ack_num + 1 # 窗口起点序列号更新
92.                 print("此时窗口起点位置为: ", send_base)
93.                 send_timer = 0 # 发送计时器置 0
94.                 elif message[0] == '0': # 如果收到的分组是数据分组
95.                     rcv_seq_num = message[1:9] # 得到收到分组的序列号(8
                        位)
96.                     if int(rcv_seq_num) == expected_num: # 如果是期望收到的分
                        组
97.                         print('server 收到了正确分组, 向发送方发送
                            ack' + rcv_seq_num) # 打印信息
98.                         self.receive_buffer.append(rcv_pkt.decode()[9:]) #
                            将分组中的数据放入接收缓冲区中(9 位以后的数据)
99.                         ack_pkt = Data(1, '%8d ' % expected_num, '') # 创建
                            该 pkt 的 ack 分组
100.                        self.socket.sendto(str(ack_pkt).encode(), self.clie
                            nt_address) # 向发送方发送 ack 分组
101.                        last_ack = expected_num # 储存最新发送的 ack 号
102.                        expected_num += 1 # 下一次期望分组的序列号加 1
103.                    else: # 如果不是期望收到的分组
104.                        print('server 收到的分组不是期望的分组, 向发送方重发期
                            望收到的分组的上一个 ack', last_ack) # 打印信息
105.                        ack_pkt = Data(1, '%8d ' % last_ack, '') # 创建该
                            pkt 的 ack 分组
106.                        self.socket.sendto(str(ack_pkt).encode(), self.clie
                            nt_address) # 向发送方重新发送 ack 分组
107.                    else:
108.                        pass
109.                    rs, ws, es = select.select([self.socket, ], [], [], 0.1) #
                        进行非阻塞监听
110.                    else: # 非阻塞监听未收到
111.                        receive_timer += 1
112.                        send_timer += 1
113.
114.
115. def start():
116.     server_socket = GBNServer()
117.     data = []
```

```
118.     with open('server_send.txt', 'r') as f: # 打开发送内容的文件
119.         while True:
120.             pkt = f.read(10)
121.             if len(pkt) > 0: # 读取文件并储存在 data 中, data 是需要发送的文件
                内容
122.                 data.append(pkt)
123.             else:
124.                 break
125.     timer = 0 # 服务器计时器置 0
126.     while True:
127.         # 服务器计时器, 如果收不到客户端的请求则退出
128.         if timer > 20: # 超时则退出
129.             return
130.     rs, ws, es = select.select([server_socket.socket, ], [], [], 1) #
        进行非阻塞监听
131.     if len(rs) > 0: # 若有可读的 socket
132.         message, address = server_socket.socket.recvfrom(server_socket.
            buffer_size) # 服务器接收数据
133.         if message.decode() == '-time': # 如果信息为-testgbn, 则执行
134.             print(datetime.datetime.now())
135.         if message.decode() == '-testgbn': # 如果信息为-testgbn, 则执
            行
136.             server_socket.send_and_receive(data)
137.         if message.decode() == '-quit': # 如果信息为-testgbn, 则停止
138.             print('Good bye!')
139.             return
140.         timer += 1
141.
142.
143. if __name__ == '__main__':
144.     start()
```

GBN_one_way_Client.py

```
1. import socket
2. from random import random
3. import select
4. # 单向数据传输 GBN 协议
5.
6.
7. class ACK: # 定义 ACK
8.     def __init__(self, seq):
9.         self.seq = seq # 分组标记的序号
10.
11.     def __str__(self):
```

```
12.         return str(self.seq)
13.
14.
15. class GBNClient:
16.     def __init__(self):
17.         self.window_size = 5 # 窗口大小
18.         self.max_receive_time = 10 # 接收超时时间
19.         self.address = ('127.0.0.1', 1234) # 发送方地址
20.         self.server_address = ('127.0.0.1', 4321) # 接收方地址
21.         self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # 新建一个 socket
22.         self.socket.bind(self.address) # 绑定到本地固定窗口
23.         self.receive_buffer = [] # 接收缓冲区
24.         self.buffer_size = 1024 # 缓冲区大小
25.
26.     def receive(self):
27.         expected_num = 0 # 希望收到的序列号
28.         receive_timer = 0 # 接收计时器
29.         last_ack = -1 # 上一个 ack 序号
30.         while True:
31.             if receive_timer > self.max_receive_time: # 如果没有超过接收超
                时间
32.                 with open('client_receive.txt', 'w') as f:
33.                     for data in self.receive_buffer: # 将接收缓冲区的数据全部
                        写入文件
34.                         f.write(data)
35.                     break
36.
37.             rs, ws, es = select.select([self.socket, ], [], [], 0.1) # 进行
                非阻塞监听
38.
39.             while len(rs) > 0: # 若有可读的 socket
40.                 rcv_pkt, address = self.socket.recvfrom(self.buffer_size) #
                    接收数据
41.                 # 模拟丢包
42.                 message = rcv_pkt.decode()
43.                 ack_num = int(message[0:8])
44.                 if random() < 0.1:
45.                     receive_timer += 1
46.                     print('client 丢包', str(ack_num))
47.                     rs, ws, es = select.select([self.socket, ], [], [], 0.1)
48.
49.                     continue
50.                 message = rcv_pkt.decode() # 将 rcv_pkt 解码为字符串编码
```

```

50.         receive_timer = 0 # 接收计时器重新置 0
51.
52.         rcv_seq_num = message[0:8]
53.         if int(rcv_seq_num) == expected_num: # 得到收到分组的序号
            (8 位)
54.             print('client 收到了正确分组, 向发送方发送
ack ' + str(int(rcv_seq_num)) + ' ') # 打印信息
55.             self.receive_buffer.append(rcv_pkt.decode()[8:]) # 将分
组中的数据放入接收缓冲区中(8 位以后的数据)
56.             ack_pkt = ACK('%8d ' % expected_num) # 创建该 pkt 的 ack
分组
57.             self.socket.sendto(str(ack_pkt).encode(), self.server_ad
dress) # 向发送方发送 ack 分组
58.             last_ack = expected_num # 储存最新发送的 ack 号
59.             expected_num += 1 # 下一次期望分组的序号加 1
60.         else: # 如果不是期望收到的分组
61.             print('client 收到的分组不是期望的分组, 向发送方重发期望收到
的分组的上一个 ack ', str(int(last_ack)) + ' ') # 打印信息
62.             ack_pkt = ACK('%8d ' % last_ack) # 创建该 pkt 的 ack 分
组
63.             self.socket.sendto(str(ack_pkt).encode(), self.server_ad
dress) # 向发送方重新发送 ack 分组
64.
65.             rs, ws, es = select.select([self.socket, ], [], [], 0.1) #
进行非阻塞监听
66.         else: # 非阻塞监听未收到
67.             receive_timer += 1
68.
69.
70. def client_start():
71.     client_socket = GBNClient()
72.     client_socket.socket.sendto('-
testgbn'.encode(), client_socket.server_address)
73.     client_socket.receive()
74.
75.
76. if __name__ == '__main__':
77.     client_start()
    
```

GBN_one_way_Server.py

```

1. import datetime
2. import socket
3. from random import random
4. import select
    
```



```
5. # 单向数据传输 GBN 协议
6.
7.
8. class Data: # 定义 pck
9.     def __init__(self, seq, data):
10.         self.seq = seq # 分组标记的序号
11.         self.data = data # 数据字段
12.
13.     def __str__(self):
14.         return str(self.seq) + str(self.data)
15.
16.
17. class GBNServer:
18.
19.     def __init__(self):
20.         self.window_size = 5 # 窗口大小
21.         self.max_send_time = 3 # 发送超时时间
22.         self.address = ('127.0.0.1', 4321) # 发送方地址
23.         self.client_address = ('127.0.0.1', 1234) # 接收方地址
24.         self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # 新建一个 socket
25.         self.socket.bind(self.address) # 绑定到本地固定窗口
26.         self.send_window = [] # 发送窗口
27.         self.buffer_size = 1024
28.
29.     def send(self, buffer):
30.         send_timer = 0 # 发送计时器
31.         send_base = 0 # 窗口此时起点的序列号
32.         next_seq_num = send_base # 下一个待发送的序列号
33.         while next_seq_num <= len(buffer)-1:
34.             # 当发送窗口未满且有数据还需要发送时，发送数据
35.             while next_seq_num < send_base + self.window_size and next_seq_num < len(buffer):
36.                 pkt = Data('%8d' % next_seq_num, buffer[next_seq_num]) # 从缓冲区取出需要发送的数据
37.                 self.socket.sendto(str(pkt).encode(), self.client_address)
38.                 # server 发送数据
39.                 print('server 发送了 pkt' + str(next_seq_num) + ' ') # 打印发送信息
40.                 self.send_window.append(pkt) # 将数据加入发送窗口
41.                 if send_base == next_seq_num: # 只要该数据的序列号不是起点就将序列号加一
42.                     send_timer = 0
43.                     next_seq_num = next_seq_num + 1
```

```
43.
44.         # 若发送的时间超过给定的发送超时时间, 则重传发送窗口中的数据
45.         if send_timer > self.max_send_time and self.send_window:
46.             print('server 发送超时,需要重传')
47.             send_timer = 0 # 发送计时器重新置0
48.             for pkt in self.send_window: # 对于发送窗口中所有 pkt 数据
49.                 self.socket.sendto(str(pkt).encode(), self.client_addresses) # 重传数据
50.             print('server 重传的数据包
    为 ' + str(int(str(pkt.seq))) + ' ')
51.
52.         rs, ws, es = select.select([self.socket, ], [], [], 0.1) # 进行
    非阻塞监听
53.
54.         while len(rs) > 0: # 若有可读的 socket
55.             rcv_pkt, address = self.socket.recvfrom(self.buffer_size) #
    接收数据
56.             # 模拟丢包
57.             message = rcv_pkt.decode()
58.             ack_num = int(message[0:8])
59.             if random() < 0.1:
60.                 send_timer += 1
61.                 print('Server 丢包 ack', str(ack_num))
62.                 rs, ws, es = select.select([self.socket, ], [], [], 0.1)
63.                 continue
64.                 print('Server 收到了 ack', str(ack_num))
65.                 for i in range(len(self.send_window)): # 遍历发送窗口
66.                     if ack_num == int(self.send_window[i].seq): # 如果发送窗
    口找到了这个确认序列号
67.                         self.send_window = self.send_window[i + 1:] # 将窗口
    后移
68.                     break
69.                 send_base = ack_num + 1 # 窗口起点序列号更新
70.                 print("此时窗口起点位置为: ", send_base)
71.                 send_timer = 0 # 发送计时器置0
72.
73.                 rs, ws, es = select.select([self.socket, ], [], [], 0.1) #
    进行非阻塞监听
74.             else: # 非阻塞监听未收到
75.                 send_timer += 1
76.
77.
78. def server_start():
```

```

79.     server_socket = GBNServer()
80.     data = []
81.     with open('server_send.txt', "r") as f:
82.         while True:
83.             pkt = f.read(10)
84.             if len(pkt) > 0:
85.                 data.append(pkt)
86.             else:
87.                 break
88.     timer = 0
89.     while True:
90.         # 服务器计时器，如果收不到客户端的请求则退出
91.         if timer > 20:
92.             return
93.         rs, ws, es = select.select([server_socket.socket, ], [], [], 1)
94.         if len(rs) > 0:
95.             message, address = server_socket.socket.recvfrom(server_socket.b
uffer_size)
96.             if message.decode() == '-testgbn':
97.                 server_socket.send(data)
98.             timer += 1
99.
100.
101. if __name__ == '__main__':
102.     server_start()

```

GBN_test.py

```

1. import threading
2. from Lab2.GBN_client import start as gbn_client_start
3. from Lab2.GBN_server import start as gbn_server_start
4.
5.
6. def main():
7.     t1 = threading.Thread(target=gbn_server_start, args=())
8.     t2 = threading.Thread(target=gbn_client_start, args=())
9.     t1.start()
10.    t2.start()
11.
12.
13. if __name__ == '__main__':
14.     main()

```

SR_Client.py

```
1. import socket
2. from random import random
3. import select
4.
5.
6. class Data: # 定义 SR 的分组格式
7.     def __init__(self, is_ack, seq, data):
8.         self.is_ack = is_ack # 分组类型, 0 表示数据分组, 1 表示确认分组
9.         self.seq = seq # 分组标记的序号
10.        self.data = data # 数据字段
11.
12.    def __str__(self):
13.        return str(self.is_ack) + str(self.seq) + str(self.data)
14.
15.
16. class DataGram: # 封装 Data 类
17.    def __init__(self, pkt, timer=0, is_acked=False):
18.        self.pkt = pkt
19.        self.timer = timer # 计时器
20.        self.is_acked = is_acked # 判断该组是否已经被 ack
21.
22.
23. class SRClient:
24.    def __init__(self):
25.        self.send_window_size = 5 # 发送窗口大小
26.        self.receive_window_size = 5 # 接收窗口大小
27.        self.max_send_time = 5 # 发送超时时间
28.        self.max_receive_time = 15 # 接收超时时间
29.        self.address = ('127.0.0.1', 5678) # 发送方地址
30.        self.server_address = ('127.0.0.1', 8765) # 接收方地址
31.        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # 新建一个 socket
32.        self.socket.bind(self.address) # 绑定到本地固定窗口
33.        self.send_window = [] # 发送窗口
34.        self.receive_data = [] # 有序数据
35.        self.receive_window = {} # 接收窗口
36.        self.buffer_size = 1024 # 缓冲区大小
37.
38.        ....
39.        发送和接收功能
40.        ...
41.    def send_and_receive(self, buffer):
42.        send_base = 0 # 窗口此时起点的序列号
43.        next_seq_num = send_base # 下一个待发送的序列号
```

```
44.         expected_num = 0 # 希望收到的序列号
45.         receive_timer = 0 # 接收计时器
46.         total = len(buffer) # 缓冲区大小
47.         while True:
48.             if not self.send_window and receive_timer > self.max_receive_tim
e: # 如果没有超过接收超时时间
49.                 with open('client_receive.txt', 'w') as f:
50.                     for data in self.receive_data: # 将 receive_data 的数据全
部写入文件
51.                         f.write(data)
52.                     break
53.             # 发送阶段
54.             # 当发送窗口未满足且有数据还需要发送时，发送数据
55.             while next_seq_num < send_base + self.send_window_size and next_
seq_num < total:
56.                 pkt = Data(0, '%8d' % next_seq_num, buffer[next_seq_num]) #
从缓冲区取出需要发送的数据
57.                 self.socket.sendto(str(pkt).encode(), self.server_address)
# client 发送数据
58.                 print('client 发送了 pkt' + str(next_seq_num)) # 打印发送信
息
59.                 self.send_window.append(DataGram(pkt)) # 给每个分组计时器初始
化为 0，把数据报加入发送窗口
60.                 next_seq_num = next_seq_num + 1 # 将 next_seq_num 序列号加
一
61.
62.             # 遍历所有已发送但未确认的分组，如果有超时的分组，则重发该分组
63.             for dgram in self.send_window:
64.                 if dgram.timer > self.max_send_time and not dgram.is_acked:
# 如果已经超时且该组没有被接收
65.                     self.socket.sendto(str(dgram.pkt).encode(), self.server_
address) # 重发该分组
66.                     print('server 重传的数据包为' + str(dgram.pkt.seq))
67.
68.             # 非阻塞监听
69.             rs, ws, es = select.select([self.socket, ], [], [], 0.01) # 非阻
塞监听
70.
71.             while len(rs) > 0: # 若有可读的 socket
72.                 rcv_pkt, address = self.socket.recvfrom(self.buffer_size) #
接收数据
73.                 message = rcv_pkt.decode()
74.                 ack_num = int(message[1:9])
75.                 # 模拟丢包
```

```
76.         if random() < 0.1: # 随机产生一个0到1之间的浮点数，若小于
           0.1，则说明丢包率小于10%，则不处理该数据
77.             for dgram in self.send_window:
78.                 dgram.timer += 1
79.                 receive_timer += 1
80.                 print('client 丢包 ', str(ack_num))
81.                 rs, ws, es = select.select([self.socket, ], [], [], 0.01
           )
82.             continue
83.         receive_timer = 0 # 接收计时器重新置0
84.         message = rcv_pkt.decode() # 将rcv_pkt解码为字符串编码
85.
86.         if message[0] == '1': # 如果收到的分组是确认分组
87.             ack_num = int(message[1:9]) # 得到确认序列号(8位)
88.             print('client 收到 ack:' + str(ack_num))
89.             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
90.                 if int(dgram.pkt.seq) == ack_num: # 如果找到了
           ack_num
91.                     dgram.timer = 0 # 计时器置0
92.                     dgram.is_acked = True # 标记为已收到
93.                     if self.send_window.index(dgram) == 0: # 如果
           ack的是发送窗口中的第一个分组
94.                         idx = -1 # 用idx表示窗口中最后一个已确认的的
           分组的下标
95.                         for i in self.send_window: # 依次遍历发送窗口
           中已确认的分组的个数
96.                             if i.is_acked: # 记录从第一个分组开始连续
           分组的个数
97.                                 idx += 1
98.                             else: # 不连续则退出
99.                                 break
100.                        send_base = int(self.send_window[idx].pkt.s
           eq) + 1
101.                        print("此时窗口起点位置为: ", send_base)
102.                        self.send_window = self.send_window[idx+1:]
           # 窗口滑动到最后一个连续已确认的分组序号之后
103.                        break
104.         elif message[0] == '0': # 如果收到的分组是数据分组
105.             for dgram in self.send_window: # 发送窗口中所有分组计时
           器+1
106.                 dgram.timer += 1
107.             rcv_seq_num = message[1:9] # 得到收到分组的序列号(8
           位)
```

```
108.             if int(rcv_seq_num) == expected_num: # 如果是期望收到的
                分组
109.                 print('client 收到了正确分组，向发送方发送
                    ack' + str(rcv_seq_num)) # 打印信息
110.                 ack_pkt = Data(1, '%8d ' % expected_num, '') # 创
                    建该 pkt 的 ack 分组
111.                 self.socket.sendto(str(ack_pkt).encode(), self.serv
                    er_address) # 向发送方发送 ack 分组
112.                 # 再看它后面有没有能合并的分组
113.                 self.receive_window[int(rcv_seq_num)] = rcv_pkt #
                    将最新收到的数据序列号加给接收窗口
114.                 tmp = [(k, self.receive_window[k]) for k in sorted(
                    self.receive_window.keys())] # 按照序列号对接收窗口进行排序
115.                 idx = 0 # idx 为接受窗口中能合并到的最后一个分组，为
                    连续数对的个数
116.                 for i in range(len(tmp) - 1):
117.                     if tmp[i + 1][0] - tmp[i][0] == 1: # 如果两个序
                        列号相差为 1
118.                         idx += 1
119.                     else: # 相差不为 1 跳出（不连续）
120.                         break
121.                 for i in range(idx + 1):
122.                     self.receive_data.append(tmp[i][1].decode()[9:])
                        ) # 把接收窗口中的数据提交给 receive_data
123.                     # 记录提交的分组的序号
124.                     base = int(tmp[0][1].decode()[1:9]) # 写入
                        receive_data 的序列号起点
125.                     end = int(tmp[idx][1].decode()[1:9]) # 写入
                        receive_data 的序列号终点
126.                     if base != end:
127.                         print('client 向上层提交数
                            据: ' + str(base) + ' 到 ' + str(end))
128.                     else:
129.                         print('client 向上层提交数据: ' + str(base))
130.                     expected_num = tmp[idx][0] + 1 # 下一次期望分组的序列
                        号加 1
131.                     tmp = tmp[idx + 1:] # 接收窗口滑动
132.                     self.receive_window = dict(tmp)
133.                 else: # 如果不是期望收到的分组
134.                     if expected_num <= int(rcv_seq_num) < expected_num
                        + self.receive_window_size - 1: # 如果 rcv_seq_num 在接收窗口内，即若在
                            rcv_base~rcv_base + N - 1 之内，则加入接收窗口
135.                         self.receive_window[int(rcv_seq_num)] = rcv_pkt
                            # 加入接收窗口
```

```
136.         ack_pkt = Data(1, '%8d ' % int(rcv_seq_num), ''
    ) # 创建该 pkt 的 ack 分组
137.         print('client 收到的分组不是期望的分组，但在接收窗
    口内，向发送方发送该分组对应的 ack' + str(rcv_seq_num)) # 打印信息
138.         self.socket.sendto(str(ack_pkt).encode(), self.
    server_address) # 向发送方发送 ack 分组
139.         elif int(rcv_seq_num) < expected_num: # 如果收到的
    分组不在接收窗口内（窗口前）
140.         ack_pkt = Data(1, '%8d ' % int(rcv_seq_num), ''
    ) # 创建该 pkt 的 ack 分组
141.         print('client 收到的分组不是期望的分组，但不在接收
    窗口内，向发送方发送该分组对应的 ack' + str(rcv_seq_num))
142.         self.socket.sendto(str(ack_pkt).encode(), self.
    server_address) # 向发送方发送 ack 分组
143.         else:
144.             print('丢弃')
145.             pass
146.         else:
147.             pass
148.         rs, ws, es = select.select([self.socket, ], [], [], 0.01)
    # 进行非阻塞监听
149.         else: # 非阻塞监听未收到
150.             receive_timer += 1
151.             for dgram in self.send_window:
152.                 dgram.timer += 1
153.
154.
155. def start():
156.     client_socket = SRClient()
157.     data = []
158.     with open('client_send.txt', 'r') as f: # 打开发送内容的文件
159.         while True:
160.             pkt = f.read(10)
161.             if len(pkt) > 0:
162.                 data.append(pkt) # 读取文件并储存在 data 中，data 是需要发送的
    文件内容
163.             else:
164.                 break
165.         while True:
166.             temp = input("请输入命令: ")
167.             if temp == '-time':
168.                 client_socket.socket.sendto('-
    time'.encode(), client_socket.server_address)
169.             if temp == '-testsr':
```



```
170.         client_socket.socket.sendto('-  
        testsr'.encode(), client_socket.server_address)  
171.         client_socket.send_and_receive(data)  
172.         if temp == '-quit':  
173.             client_socket.socket.sendto('-  
             quit'.encode(), client_socket.server_address)  
174.  
175.  
176. if __name__ == '__main__':  
177.     start()
```

SR_Server.py

```
1. import socket  
2. from random import random  
3. import select  
4. import datetime  
5.  
6.  
7. class Data: # 定义 SR 的分组格式  
8.     def __init__(self, is_ack, seq, data):  
9.         self.is_ack = is_ack # 分组类型, 0 表示数据分组, 1 表示确认分组  
10.        self.seq = seq # 分组标记的序号  
11.        self.data = data # 数据字段  
12.  
13.    def __str__(self):  
14.        return str(self.is_ack) + str(self.seq) + str(self.data)  
15.  
16.  
17. class DataGram: # 封装 Data 类  
18.    def __init__(self, pkt, timer=0, is_acked=False):  
19.        self.pkt = pkt  
20.        self.timer = timer # 计时器  
21.        self.is_acked = is_acked # 判断该组是否已经被 ack  
22.  
23.  
24. class SRServer:  
25.    def __init__(self):  
26.        self.send_window_size = 5 # 发送窗口大小  
27.        self.receive_window_size = 5 # 接收窗口大小  
28.        self.max_send_time = 5 # 发送超时时间  
29.        self.max_receive_time = 15 # 接收超时时间  
30.        self.address = ('127.0.0.1', 8765) # 发送方地址  
31.        self.client_address = ('127.0.0.1', 5678) # 接收方地址
```

```
32.         self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # 新建一个 socket
33.         self.socket.bind(self.address) # 绑定到本地固定窗口
34.         self.send_window = [] # 发送窗口
35.         self.receive_data = [] # 有序数据
36.         self.receive_window = {} # 接收窗口
37.         self.buffer_size = 1024 # 缓冲区大小
38.
39.         ....
40.         发送和接收功能
41.         '''
42.         def send_and_receive(self, buffer):
43.             send_base = 0 # 窗口此时起点的序列号
44.             next_seq_num = send_base # 下一个待发送的序列号
45.             expected_num = 0 # 希望收到的序列号
46.             receive_timer = 0 # 接收计时器
47.             total = len(buffer) # 缓冲区大小
48.             while True:
49.                 if not self.send_window and receive_timer > self.max_receive_time: # 如果没有超过接收超时时间
50.                     with open('server_receive.txt', 'w') as f:
51.                         for data in self.receive_data: # 将 receive_data 的数据全部写入文件
52.                             f.write(data)
53.                         break
54.
55.                 # 发送阶段
56.                 # 当发送窗口未满足且有数据还需要发送时，发送数据
57.                 while next_seq_num < send_base + self.send_window_size and next_seq_num < total:
58.                     pkt = Data(0, '%8d' % next_seq_num, buffer[next_seq_num]) # 从缓冲区取出需要发送的数据
59.                     self.socket.sendto(str(pkt).encode(), self.client_address) # server 发送数据
60.                     print('server 发送了 pkt' + str(next_seq_num)) # 打印发送信息
61.                     self.send_window.append(DataGram(pkt)) # 给每个分组计时器初始化为 0，把数据报加入发送窗口
62.                     next_seq_num = next_seq_num + 1 # 将 next_seq_num 序列号加一
63.
64.                 # 遍历所有已发送但未确认的分组，如果有超时的分组，则重发该分组
65.                 for dgram in self.send_window:
```

```
66.         if dgram.timer > self.max_send_time and not dgram.is_acked:
67.             # 如果已经超时且该组没有被接收
68.                 self.socket.sendto(str(dgram.pkt).encode(), self.client_
69. address) # 重发该分组
70.                 print('server 重传的数据包为' + str(dgram.pkt.seq))
71.             # 接收阶段
72.             rs, ws, es = select.select([self.socket, ], [], [], 0.01) # 非阻
73.             塞监听
74.             while len(rs) > 0: # 若有可读的 socket
75.                 rcv_pkt, address = self.socket.recvfrom(self.buffer_size) #
76.                 接收数据
77.                 # 模拟丢包
78.                 message = rcv_pkt.decode()
79.                 ack_num = int(message[1:9])
80.                 if random() < 0.1: # 随机产生一个 0 到 1 之间的浮点数, 若小于
81.                 0.1, 则说明丢包率小于 10%, 则不处理该数据
82.                     for dgram in self.send_window: # 对所有分组计时器加 1
83.                         dgram.timer += 1
84.                         receive_timer += 1
85.                         print('server 丢包 ', str(ack_num))
86.                         rs, ws, es = select.select([self.socket, ], [], [], 0.01
87. )
88.                     continue
89.                     message = rcv_pkt.decode() # 将 rcv_pkt 解码为字符串编码
90.                     receive_timer = 0 # 接收计时器重新置 0
91.                     if message[0] == '1': # 如果收到的分组是确认分组
92.                         ack_num = int(message[1:9]) # 得到确认序列号(8 位)
93.                         print('server 收到 ack' + str(ack_num))
94.                         for dgram in self.send_window: # 在窗口中找序号为
95.                         ack_num 的分组, 并把 is_acked 设为 True
96.                             if int(dgram.pkt.seq) == ack_num: # 如果找到了
97.                                 ack_num
98.                                 dgram.timer = 0 # 计时器置 0
99.                                 dgram.is_acked = True # 标记为已收到
100.                                if self.send_window.index(dgram) == 0: # 如果
101.                                ack 的是发送窗口中的第一个分组
102.                                    idx = -1 # 用 idx 表示窗口中最后一个已确认的的
103.                                    分组的下标
104.                                    for i in self.send_window: # 依次遍历发送窗口
105.                                    中已确认的分组个数
106.                                        if i.is_acked: # 记录从第一个分组开始连续
107.                                        分组的个数
```

```
98.                 idx += 1
99.                 else: # 不连续则退出
100.                     break
101.
102.                 send_base = int(self.send_window[idx].pkt.seq) + 1
103.                 print("此时窗口起点位置为: ", send_base)
104.                 self.send_window = self.send_window[idx+1:]
105.                 # 窗口滑动到最后一个连续已确认的分组序号之后
106.                 break
107.                 elif message[0] == '0': # 如果收到的分组是数据分组
108.                     for dgram in self.send_window: # 发送窗口中所有分组计时器+1
109.                         dgram.timer += 1
110.                     rcv_seq_num = message[1:9] # 得到收到分组的序号(8位)
111.                     if int(rcv_seq_num) == expected_num: # 如果是期望收到的分组
112.                         print('server 收到了正确分组，向发送方发送 ack' + str(rcv_seq_num)) # 打印信息
113.                         ack_pkt = Data(1, '%8d ' % expected_num, '') # 创建该 pkt 的 ack 分组
114.                         self.socket.sendto(str(ack_pkt).encode(), self.client_address) # 向发送方发送 ack 分组
115.                         # 再看它后面有没有能合并的分组
116.                         self.receive_window[int(rcv_seq_num)] = rcv_pkt # 将最新收到的数据序号加给接收窗口
117.                         tmp = [(k, self.receive_window[k]) for k in sorted(self.receive_window.keys())] # 按照序号对接收窗口进行排序
118.                         idx = 0 # idx 为接受窗口中能合并到的最后一个分组，为连续数对的个数
119.                         for i in range(len(tmp) - 1):
120.                             if tmp[i + 1][0] - tmp[i][0] == 1: # 如果两个序号相差为 1
121.                                 idx += 1
122.                             else: # 相差不为 1 跳出（不连续）
123.                                 break
124.                         for i in range(idx + 1):
125.                             self.receive_data.append(tmp[i][1].decode()[9:]) # 把接收窗口中的数据提交给 receive_data
126.                             # 记录提交的分组的序号
127.                             base = int(tmp[0][1].decode()[1:9]) # 写入 receive_data 的序号起点
```

```
127.                 end = int(tmp[idx][1].decode())[1:9]) # 写入
receive_data 的序列号终点
128.                 if base != end:
129.                     print('server 向上层提交数
据: ' + str(base) + ' 到 ' + str(end))
130.                 else:
131.                     print('server 向上层提交数据: ' + str(base))
132.                 expected_num = tmp[idx][0]+1 # 下一次期望分组的序列
号加 1
133.                 tmp = tmp[idx + 1:] # 接收窗口滑动
134.                 self.receive_window = dict(tmp)
135.             else: # 如果不是期望收到的分组
136.                 if expected_num <= int(rcv_seq_num) < expected_num
+ self.receive_window_size - 1: # 如果 rcv_seq_num 在接收窗口内, 即若在
rcv_base~rcv_base + N -1 之内, 则加入接收窗口
137.                     self.receive_window[int(rcv_seq_num)] = rcv_pkt
# 加入接收窗口
138.                     ack_pkt = Data(1, '%8d ' % int(rcv_seq_num), ''
) # 创建该 pkt 的 ack 分组
139.                     print('server 收到的分组不是期望的分组, 但在接收窗
口内, 向发送方发送该分组对应的 ack' + str(rcv_seq_num)) # 打印信息
140.                     self.socket.sendto(str(ack_pkt).encode(), self.
client_address) # 向发送方发送 ack 分组
141.                 elif int(rcv_seq_num) < expected_num: # 如果收到的
分组不在接收窗口内 (窗口前)
142.                     ack_pkt = Data(1, '%8d ' % int(rcv_seq_num), ''
) # 创建该 pkt 的 ack 分组
143.                     print('server 收到的分组不是期望的分组, 但不在接收
窗口内, 向发送方发送该分组对应的 ack' + str(rcv_seq_num))
144.                     self.socket.sendto(str(ack_pkt).encode(), self.
client_address) # 发送 ack 分组
145.                 else:
146.                     pass
147.             else:
148.                 pass
149.             rs, ws, es = select.select([self.socket, ], [], [], 0.01)
# 进行非阻塞监听
150.         else: # 非阻塞监听未收到
151.             receive_timer += 1
152.         for dgram in self.send_window:
153.             dgram.timer += 1
154.
155.
156. def start():
```

```

157.     server_socket = SRServer()
158.     data = []
159.     with open('server_send.txt', 'r') as f: # 打开发送内容的文件
160.         while True:
161.             pkt = f.read(10)
162.             if len(pkt) > 0: # 读取文件并储存在 data 中, data 是需要发送的文件
                内容
163.                 data.append(pkt)
164.             else:
165.                 break
166.     timer = 0
167.     while True:
168.         # 服务器计时器, 如果收不到客户端的请求则退出
169.         if timer > 20: # 超时则退出
170.             return
171.         rs, ws, es = select.select([server_socket.socket, ], [], [], 1) #
            进行非阻塞监听
172.         if len(rs) > 0: # 若有可读的 socket
173.             message, address = server_socket.socket.recvfrom(server_socket.
                buffer_size) # 服务器接收数据
174.             if message.decode() == '-time': # 如果信息为-testgbn, 则执行
175.                 print(datetime.datetime.now())
176.             if message.decode() == '-testsr': # 如果信息为-testsr, 则执行
177.                 server_socket.send_and_receive(data)
178.             if message.decode() == '-quit': # 如果信息为-testgbn, 则停止
179.                 print('Good bye!')
180.                 return
181.             timer += 1
182.
183.
184. if __name__ == '__main__':
185.     start()
    
```

SR_test.py

```

1. import threading
2. from Lab2.SR_Client import start as sr_client_start
3. from Lab2.SR_Server import start as sr_server_start
4.
5.
6. def main():
7.
8.     t1 = threading.Thread(target=sr_server_start, args=())
9.     t2 = threading.Thread(target=sr_client_start, args=())
10.    t1.start()
    
```

```
11.     t2.start()
12.
13.
14. if __name__ == '__main__':
15.     main()
```

test_single_GBN.py

```
1. from Lab2.GBN_one_way_Server import server_start
2.
3.
4. def main():
5.     server_start()
6.
7.
8. if __name__ == '__main__':
9.     main()
```