哈爾濱工業大學
Harbin Institute of Technology

计算机网络
课程实验报告

<table>
<tr><td>实验名称</td><td colspan="5">GBN 协议的设计与实现</td></tr>
<tr><td>姓名</td><td colspan="2">梅智敏</td><td>院系</td><td colspan="2">计算学部软件工程</td></tr>
<tr><td>班级</td><td colspan="2">1837101</td><td>学号</td><td colspan="2">1183710118</td></tr>
<tr><td>任课教师</td><td colspan="2">李全龙</td><td>指导教师</td><td colspan="2">李全龙</td></tr>
<tr><td>实验地点</td><td colspan="2">格物 213</td><td>实验时间</td><td colspan="2">2020.11.7</td></tr>
<tr><td rowspan="2">实验课表现</td><td>出勤、表现得分(10)</td><td></td><td rowspan="2">实验报告
得分(40)</td><td rowspan="2"></td><td rowspan="2">实验总分</td></tr>
<tr><td>操作结果得分(50)</td><td></td></tr>
<tr><td colspan="6">教师评语</td></tr>
<tr><td colspan="6"></td></tr>
</table>

计算学部

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；
- 附加掌握基于 UDP 设计并实现一个 SR 协议的过程与技术。

实验内容：

- 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）
- 模拟引入数据包的丢失，验证所设计协议的有效性；
- 改进所设计的 GBN 协议，支持双向数据传输（选作， 加分项）；
- 将所设计的 GBN 协议改进为 SR 协议（选作， 加分项）。

实验过程：

注意：由于停等协议只需要将GBN协议的发送窗口大小设置为1便可实现，故本实验直接展示GBN和SR的实现过程，不再赘述停等协议。

一、 首先展示各个分组的格式

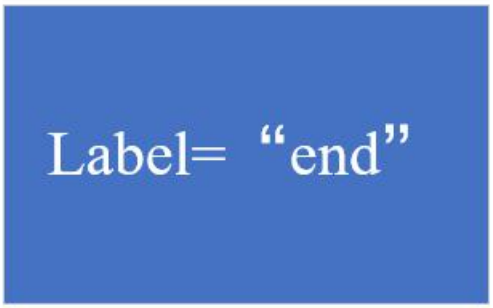
- 数据分组报文

1. 其中Label的值为“data”字符串，表示这是个数据报分组
2. “$”作为各个域的分隔符
3. Sequence是分组的序列号

- ACK分组报文

1. Label 的值为“ACK”字符串，表示这是个 ACK 报文分组
2. “$”和“Sequence”含义与数据报分组相同

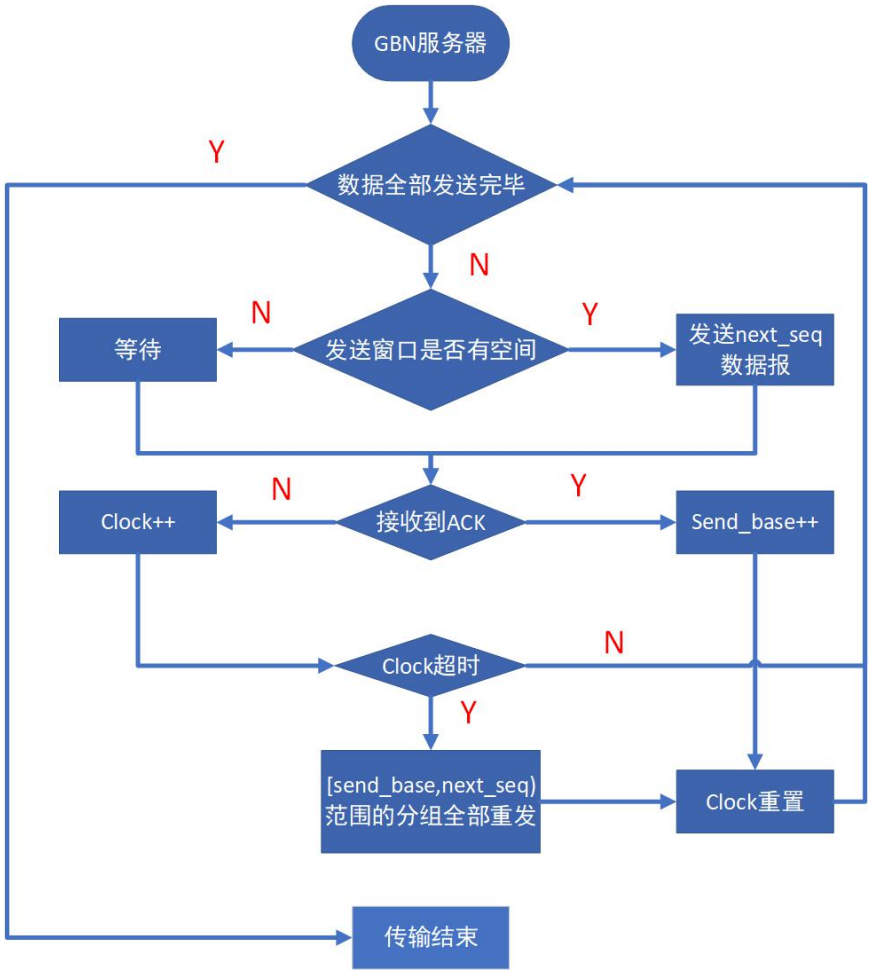
- 结束表示分组报文



1. Label的值为“end”字符串，表示这是个结束标志分组
2. 结束报文分组只在数据传输结束时起作用，用来告诉对方所有数据都已经传输完毕

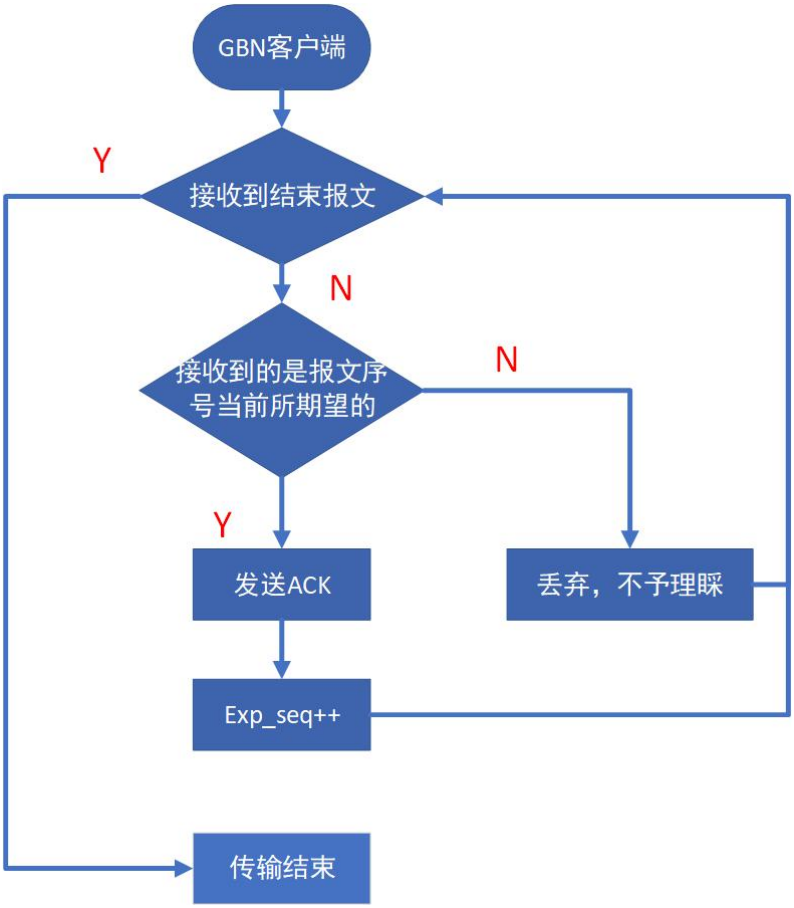
二、 接下来展示协议两端程序流程图

- GBN 的服务器



服务器主要有 2 个任务：发送数据报文和接收 ACK，并在内部设置一个计时器 Clock，用来记录 send_base 是否超时，若超时则需要重传所有空中分组。

- GBN 的客户端

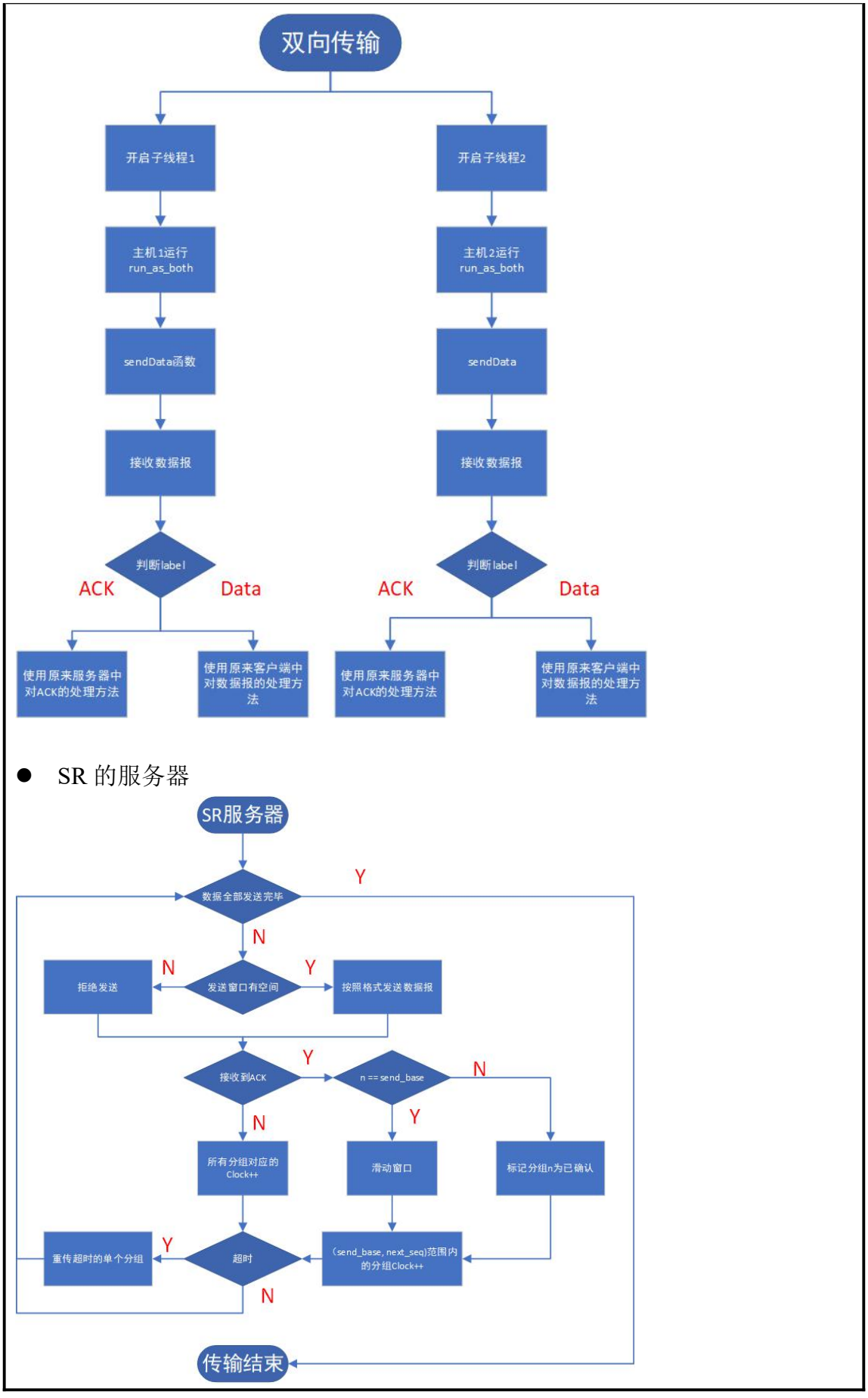


GBN 的客户端任务相对轻松，只需要判断接收到的数据报文的序列号是否和自己当前所期望的值“exp_seq”相同，若相同则构造 ACK 报文发送给服务器，否则直接丢弃。

- GBN 双向传输

为实现 GBN 的双向传输，我将客户端的功能集成到服务器，从而构建出 run_as_both 函数；再创建 2 个子线程，执行函数均为 run_as_both 函数即可实现双向通信。

下面展示示意图：

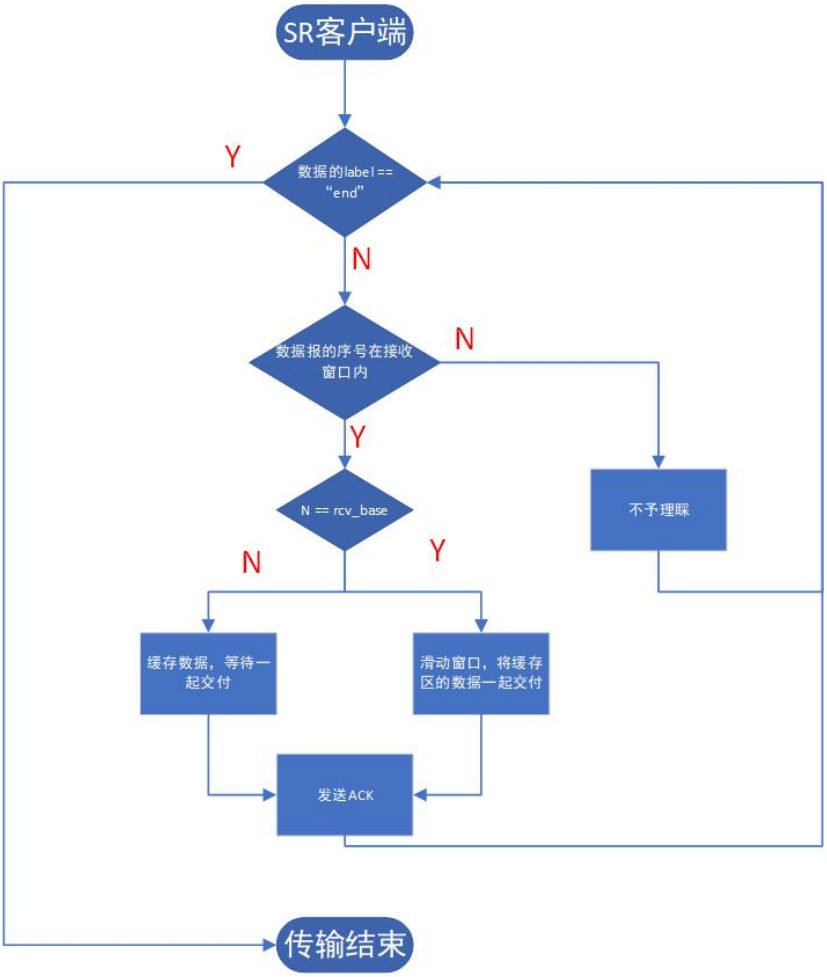

双向传输
开启子线程1
开启子线程2
主机1运行
run_as_both
主机2运行
run_as_both
sendData函数
sendData
接收数据报
接收数据报
判断label
判断label
ACK
Data
ACK
Data
使用原来服务器中对ACK的处理方法
使用原来客户端中对数据报的处理方法
使用原来服务器中对ACK的处理方法
使用原来客户端中对数据报的处理方法

- SR 的服务器

SR服务器
数据全部发送完毕
Y
N
拒绝发送
N
发送窗口有空间
Y
按照格式发送数据报
接收到ACK
Y
n == send_base
N
N
Y
所有分组对应的Clock++
滑动窗口
标记分组n为已确认
重传超时的单个分组
Y
超时
(send_base, next_seq)范围内的分组Clock++
N
传输结束

SR 服务器与 GBN 的主要区别在于：为每个分组单独设置一个 Clock，当接收到 ACK 时便将对应分组标记为已接收；当遇到超时事件时，单独重发一个分组，而不是将全部的空中分组全部重发。

- SR 的客户端



SR 客户端与 GBN 的主要区别在于：在客户端也设置一个滑动窗口，当到达的数据报不在窗口范围内便不予理睬。反之，判断 n 和 rcv_base 是否相同，若相同则滑动窗口并交付数据；若不同则将该数据报缓存起来，而不是像 GBN 那样直接丢弃，然后等待后面一起交付。

三、　数据包丢失模拟方法

本实验的模拟丢包是通过设置 2 个丢包阈值，再配合随机数生成函数来实现概率传输报文，从而模拟现实中丢包的情况。

- 模拟数据报文分组丢包

1. 首先在服务器的 class 中设置了数据报丢包阈值

```
# 设置数据包丢包率阈值
self.pkt_loss = 0.2
```

2. 结束 python 中内置的 random 函数产生（0,1）区间中的随机数，若该随机数

大于我们设定的丢包阈值，则构造数据报文分组进行发送；否则便不发送，这样便可以模拟出数据报文分组丢包的情况。

```
# 发送数据包方法
def sendData(self):
    # 若窗口仍有空间，则构造数据进行发送
    if self.next_seq < self.send_base + self.send_window:
        # 模拟丢包
        if random.random() > self.pkt_loss:
            # 构造data数据报进行发送
            self.socket.sendto(main.make_pkt(self.next_seq, self.send_buf[self.next_seq], 'data'), self.remoteHost)
        print((self.localHost, ':成功发送数据报' + str(self.next_seq)), file=self.infoFile)
        self.next_seq += 1
    else:
        print((self.localHost, ": 窗口空间已满，请等待！"), file=self.infoFile)
```

3. 当遇到 Clock 超时，需要重传空中的全部数据分组的时候，依然采用上面的方式来模拟数据包的丢失。

```
# 处理超时事件
def handle_timeOut(self, seq):
    print((self.localHost, ': 数据报' + str(seq) + '超时，下面开始重传所有空中分组'), file=self.infoFile)
    # 发送空中的所有分组
    for i in range(self.send_base, self.next_seq):
        # 模拟丢包
        if random.random() > self.pkt_loss:
            self.socket.sendto(main.make_pkt(i, self.send_buf[i], 'data'), self.remoteHost)
        print((self.localHost, ': 已重发数据报:' + str(i)), file=self.infoFile)
    # 超时计次重启
    self.clock = 0
```

- 模拟 ACK 报文分组丢包

1. 同样，在客户端的 Class 中设定一个 ACK 丢包阈值

```
# ACK丢包率阈值
self.ack_loss = 0.2
```

2. 当客户端需要发送 ACK 报文时，采用同前面一样的方式进行传输，以模拟 ACK 报文分组的丢失。

```
# 随机丢包发送ACK
if random.random() >= self.ack_loss:
    self.socket.sendto(main.make_pkt(self.exp_seq, 0, 'ACK'), self.remoteHost)
    # 若成功发送ACK，则更新exp_seq
    self.exp_seq += 1
```

四、 程序主要类、方法的作用

- GBN_Server类

1. getData方法 ：用于从txt文件中获取内容，存入send_buf中，以便于后续传输

注意：本实验利用文件读写来模拟服务器接收到上层应用发送来的数据，以及客户端将接受

到的数据传输给上层应用。

服务器所读入的文件如下：

```
Neil Armstrong on the moon next to an American flag. Laika the dog sitting in a space capsule, shortly before becoming the first an

尼尔·阿姆斯特朗在月球上与美国国旗的合影。第一只绕地球飞行的太空狗莱卡发射前坐在太空舱中的照片。巴兹·奥尔德林以蓝色地球弧线为背景的史上第一张太空自拍照。

These are just three of the 2,400 rare NASA photos now up for sale in an online auction hosted by the Christie's. The images captur

这些只是佳士得主办的在线拍卖会上展售的2400张美国宇航局罕见照片的其中三张。佳士得在新闻稿中指出，这套照片集涵盖了“太空探索的黄金时期”。

The collection of original photographs spans a number of historic missions, from the Mercury and Gemini spaceflight programs to the

这套原始照片集囊括水星计划、双子星计划以及阿波罗登月计划等一系列历史性任务，比如，阿波罗17号任务中，宇航员拍到的“蓝色弹珠”照片，现在都极具代表性意义。“蓝色弹

Others, however, were not released by NASA at the time they were taken, and are new to the general public.

不过，其他照片被拍下时美国宇航局没有发布出来，所以公众还是初次看到。

The most expensive item on offer is the only photo showing Armstrong on the Moon, taken by Aldrin during the Apollo 11 mission in 1

拍卖会上最昂贵的拍品是1969年阿波罗11号任务期间，宇航员奥尔德林拍下的唯一一张阿姆斯特朗在月球上的照片。据佳士得估计，这张照片价格应在3万至5万英镑（约合人民币
```

2. sendData方法 ：服务器首先判断窗口内是否有空间，若有，则依据我们规定的格式构造数据报文分组发送给客户端，随后将next_seq++

```
# 发送数据包方法
def sendData(self):
    # 若窗口仍有空间，则构造数据进行发送
    if self.next_seq < self.send_base + self.send_window:
        # 模拟丢包
        if random.random() > self.pkt_loss:
            # 构造data数据报进行发送
            self.socket.sendto(main.make_pkt(self.next_seq, self.send_buf[self.next_seq], 'data'), self.remoteHost)
        print((self.localHost, ':成功发送数据报' + str(self.next_seq)), file=self.infoFile)
        self.next_seq += 1
    else:
        print((self.localHost, ": 窗口空间已满，请等待！"), file=self.infoFile)
```

3. handle_timeOut方法 ：处理计时器超时事件，会将目前空中的全部分组进行重发，并将计时器重置。

```
# 发送空中的所有分组
for i in range(self.send_base, self.next_seq):
    # 模拟丢包
    if random.random() > self.pkt_loss:
        self.socket.sendto(main.make_pkt(i, self.send_buf[i], 'data'), self.remoteHost)
    print((self.localHost, ': 已重发数据报:' + str(i)), file=self.infoFile)
```

4. run_as_server方法 ：作为服务器的线程执行目标函数，既将前面的函数集成到一起，并不断检测是否接收到ACK，依据情况执行sendData方法和handle_timeOut方法

- GBN_Client类

1. writeDataToFile 方法 ：将接收到的数据写入文件，模拟将数据交付给上层应用的功能

```
def writeDataToFile(self, data):
    with open(self.write_path, 'a', encoding='utf-8') as f:
        f.write(data)
```

2. run_as_client 方法 ：作为客户端的线程执行目标函数，核心功能就是检测当前收到的数据分组是否是自己当前所期望的，再分情况决定发送 ACK 还是直接丢弃数据分组

```
# 接收到按序数据报
if int(seq) == self.exp_seq:
    print((self.localHost, ' : 收到期望序号分组数据报 ' + str(seq)), file=self.infoFile)
    # 将数据写入到本地文件中
    self.writeDataToFile(data)
    # 随机丢包发送ACK
    if random.random() >= self.ack_loss:
        self.socket.sendto(main.make_pkt(self.exp_seq, 0, 'ACK'), self.remoteHost)
        # 若成功发送ACK，则更新exp_seq
        self.exp_seq += 1
# 接收到不按序的数据报，直接丢弃
else:
    print((self.localHost, ' : 收到非期望数据，期望 : ' + str(self.exp_seq) + '实际 : ' + str(seq)),
          file=self.infoFile)
```

- SR类

1. slide_send_window方法：滑动服务器的发送窗口

```
# 滑动发送窗口，用于接收到最小的ack后调用
def slide_send_window(self):
    while self.ack_seqs.get(self.send_base):  # 一直滑动到未接收到ACK的分组序号处
        del self.ack_seqs[self.send_base]  # 从dict数据结构中删除此关键字
        del self.time_counts[self.send_base]  # 从dict数据结构中删除此关键字
        self.send_base = self.send_base + 1  # 滑动窗口
        print((self.local_address, ' :窗口滑动到' + str(self.send_base)), file=self.infoFIle1)
```

一直滑动到未被标记为已接收的分组序号处

2. slide_rcv_window方法：滑动客户端的接收窗口

```
# 滑动接收窗口:滑动rcv_base，向上层交付数据，并清除已交付数据的缓存
def slide_rcv_window(self):
    while self.rcv_data.get(self.rcv_base) is not None:  # 循环直到出现未接受的数据包
        self.write_data_to_file(self.rcv_data.get(self.rcv_base))  # 写入文件
        del self.rcv_data[self.rcv_base]  # 清除该缓存
        self.rcv_base = self.rcv_base + 1  # 滑动窗口
        print((self.local_address, ' :窗口滑动到' + str(self.rcv_base)), file=self.infoFIle2)
```

3. server_run方法 ：作为SR的服务器运行函数，与GBN的主要区别在于为每个分组单独设置Clock，且接收到ACK时会将对应的分组标记为已接收，以便于后续滑动窗口

```
        if len(readable) > 0:  # 接收ACK数据
            ack_seq = self.socket.recvfrom(self.ack_buf_size)[0].decode().split('$')[1]
            if self.send_base <= int(ack_seq) < self.next_seq:  # 收到ack，则标记为已确认且超时计数为0
                print((self.local_address, ' :收到有用ACK' + ack_seq), file=self.infoFIle1)
                self.ack_seqs[int(ack_seq)] = True  # 设为已接受
                if self.send_base == int(ack_seq):  # 收到的ack为最小的窗口序号
                    self.slide_send_window()  # 则滑动窗口
            else:
                print((self.local_address, ' :收到无用ACK' + ack_seq), file=self.infoFIle1)
        for seq in self.time_counts.keys():  # 每个未接收的分组的时长都加1
            if not self.ack_seqs[seq]:  # 若未收到ACK
                self.time_counts[seq] += 1  # 则计次+1
                if self.time_counts[seq] > self.time_out:  # 触发超时操作
                    self.handle_time_out(seq)  # 超时处理
```

4. client_run方法 ：作为SR的客户端运行函数，与GBN的主要不同在于设置了接收方的滑动窗口，若接收到的数据报不是rcv_base，则先缓存起来等待一起交付，而不是像GBN一样直接丢弃

```
if self.rcv_base <= int(rcv_seq) < self.rcv_base + self.rcv_window_size:  # 序号在窗口内
    self.rcv_data[int(rcv_seq)] = rcv_data  # 失序的数据到来:缓存+发送ack
    if int(rcv_seq) == self.rcv_base:  # 按序数据的到来:滑动窗口并交付数据(清除对应的缓冲区)
        self.slide_rcv_window()
if random.random() >= self.ack_loss:
    self.socket.sendto(main.make_pkt(int(rcv_seq), 0, 'ACK'), self.remote_address)
print((self.local_address, ' :发送ACK' + rcv_seq), file=self.infoFIle2)
```

- Main函数 ：程序入口，开启2个子线程，分别执行服务器的运行函数和客户端的运行函数，从而实现彼此通信。

```
# 开启2个子线程，分别作为客户端和服务器，从而实现彼此通信
t1 = threading.Thread(target=host_1.run_as_server)
t2 = threading.Thread(target=host_2.run_as_client)
t1.start()
t2.start()
```

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。

一、 演示GBN单向传输功能

- 没丢包的情况

服务器读入的文件，共70行

```
63
64  For decades, the unreleased photos were kept in the archives of the Manned Spacecraft Center in Houston, Texas, a
65
66  几十年来，这些未被公开的照片一直保存在得克萨斯州休斯敦载人航天中心档案室，只限官方认可的研究人员才能观看。这些照片由收藏家维克多·马丁-马
67
68  The online sale, named "Voyage To Another World: The Victor Martin-Malburet Photograph Collection," runs through
69
70  这场名为“前往另一个世界的航行：维克多·马丁-马尔雷特照片收藏集”的在线拍卖会将持续到11月19日。
```

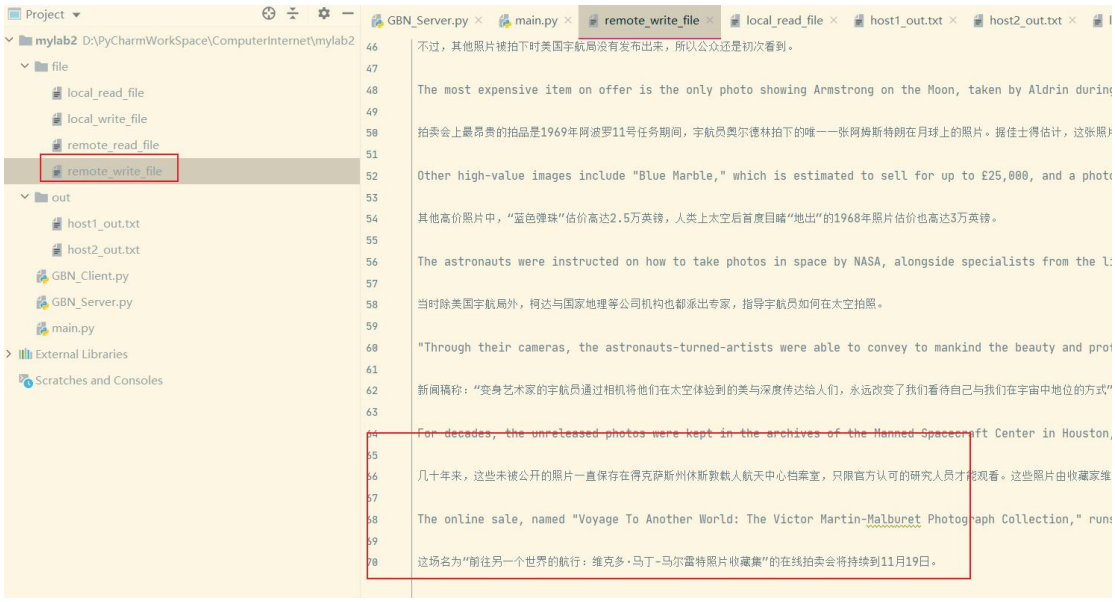
服务器的传输记录：

```
27  (('127.0.0.1', 8080), ':成功发送数据报13')
28  (('127.0.0.1', 8080), ': 收到ACK:13')
29  (('127.0.0.1', 8080), ':成功发送数据报14')
30  (('127.0.0.1', 8080), ': 收到ACK:14')
31  (('127.0.0.1', 8080), ':成功发送数据报15')
32  (('127.0.0.1', 8080), ': 收到ACK:15')
33  (('127.0.0.1', 8080), ':成功发送数据报16')
34  (('127.0.0.1', 8080), ': 收到ACK:16')
35  (('127.0.0.1', 8080), ':成功发送数据报17')
36  (('127.0.0.1', 8080), ': 收到ACK:17')
37  (('127.0.0.1', 8080), ':成功发送数据报18')
38  (('127.0.0.1', 8080), ': 收到ACK:18')
39  (('127.0.0.1', 8080), ' : 所有数据报都已发送结束且均接收到ACK，服务器关闭')
40
```

客户端的传输记录：

```
15  (('127.0.0.1', 8081), ' : 收到期望序号分组数据报 14')
16  (('127.0.0.1', 8081), ' : 收到期望序号分组数据报 15')
17  (('127.0.0.1', 8081), ' : 收到期望序号分组数据报 16')
18  (('127.0.0.1', 8081), ' : 收到期望序号分组数据报 17')
19  (('127.0.0.1', 8081), ' : 收到期望序号分组数据报 18')
20  (('127.0.0.1', 8081), ' : 所有数据报都已成功接收且均发欧发送ACK，客户端关闭')
```

客户端写入的文件内容：同样是70行，且对比后发现内容相同



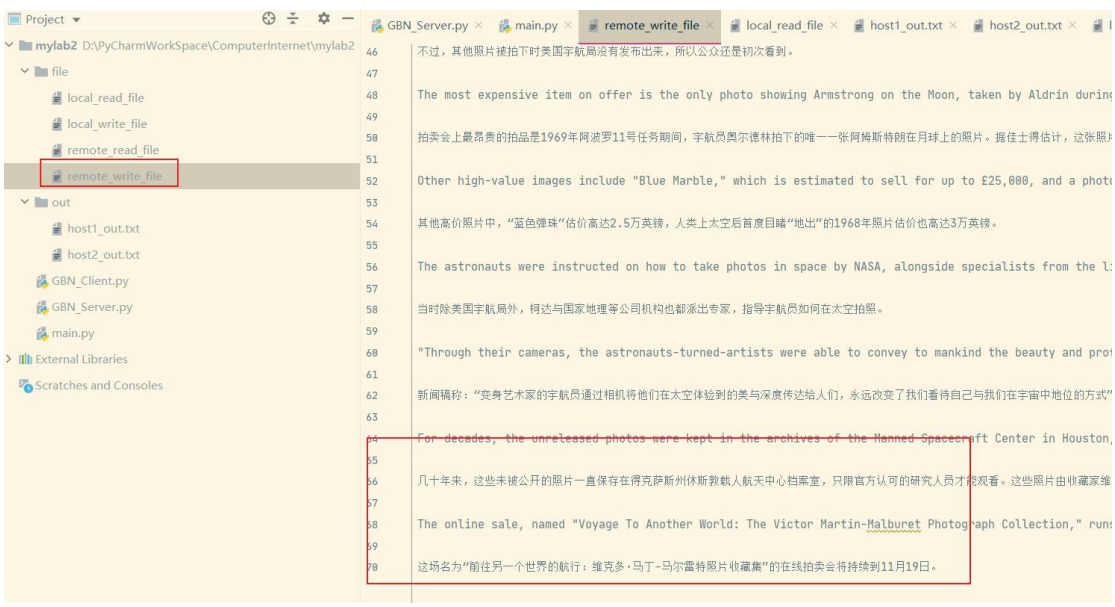
- 有丢包的情况：仍然使用一样的读入文件

服务器的传输记录：

```
15  (('127.0.0.1', 8080), ':成功发送数据报8')
16  (('127.0.0.1', 8080), ':成功发送数据报9')
17  (('127.0.0.1', 8080), ':成功发送数据报10')
18  (('127.0.0.1', 8080), ': 窗口空间已满，请等待！')
19  (('127.0.0.1', 8080), ': 数据报6超时，下面开始重传所有空中分组')
20  (('127.0.0.1', 8080), ': 已重发数据报:6')
21  (('127.0.0.1', 8080), ': 已重发数据报:7')
22  (('127.0.0.1', 8080), ': 已重发数据报:8')
23  (('127.0.0.1', 8080), ': 已重发数据报:9')
24  (('127.0.0.1', 8080), ': 已重发数据报:10')
25  (('127.0.0.1', 8080), ': 窗口空间已满，请等待！')
26  (('127.0.0.1', 8080), ': 收到ACK:6')
27  (('127.0.0.1', 8080), ':成功发送数据报11')
28  (('127.0.0.1', 8080), ': 收到ACK:7')
29  (('127.0.0.1', 8080), ':成功发送数据报12')
30  (('127.0.0.1', 8080), ': 窗口空间已满，请等待！')
```

可见当窗口已满时，会等待；且数据报6超时的时候，会重发空中的所有分组，即6、7、8、9、10

客户端的写入文件：



依然成功！

二、 演示GBN双向传输功能

1. 首先在main函数中开启2个子线程，分别让主机1和主机2运行run_as_both函数

```
def double():
    host_1 = GBN_Server.GBNServer(host1_address, host2_address, path1, path2, infoFile1)
    host_2 = GBN_Server.GBNServer(host2_address, host1_address, path3, path4, infoFile2)
    threading.Thread(target=host_1.run_as_both).start()
    threading.Thread(target=host_2.run_as_both).start()
```

2. 主机1的传输记录

```
55  (('127.0.0.1', 8080), '成功发送数据报17')
56  (('127.0.0.1', 8080), ': 收到期望序号分组数据报 14')
57  (('127.0.0.1', 8080), ': 窗口空间已满，请等待！')
58  (('127.0.0.1', 8080), ' : 收到期望序号分组数据报 15')
59  (('127.0.0.1', 8080), ': 窗口空间已满，请等待！')
60  (('127.0.0.1', 8080), ': 收到ACK:13')
61  (('127.0.0.1', 8080), ':成功发送数据报18')
62  (('127.0.0.1', 8080), ' : 收到期望序号分组数据报 16')
63  (('127.0.0.1', 8080), ': 窗口空间已满，请等待！')
64  (('127.0.0.1', 8080), ': 窗口空间已满，请等待！')
65  (('127.0.0.1', 8080), ': 窗口空间已满，请等待！')
66  (('127.0.0.1', 8080), ': 窗口空间已满，请等待！')
67  (('127.0.0.1', 8080), ': 窗口空间已满，请等待！')
68  (('127.0.0.1', 8080), ': 窗口空间已满，请等待！')
```

可见主机1既充当了服务器的角色，有充当类客户端的角色

3. 主机2的传输记录

```
7   (('127.0.0.1', 8081), ':成功发送数据报3')
8   (('127.0.0.1', 8081), ' : 收到期望序号分组数据报 2')
9   (('127.0.0.1', 8081), ':成功发送数据报4')
10  (('127.0.0.1', 8081), ': 收到ACK:1')
11  (('127.0.0.1', 8081), ':成功发送数据报5')
12  (('127.0.0.1', 8081), ' : 收到期望序号分组数据报 3')
13  (('127.0.0.1', 8081), ':成功发送数据报6')
14  (('127.0.0.1', 8081), ': 收到ACK:2')
15  (('127.0.0.1', 8081), ':成功发送数据报7')
16  (('127.0.0.1', 8081), ' : 收到期望序号分组数据报 4')
17  (('127.0.0.1', 8081), ': 窗口空间已满，请等待！')
18  (('127.0.0.1', 8081), ' : 收到期望序号分组数据报 5')
19  (('127.0.0.1', 8081), ': 窗口空间已满，请等待！')
20  (('127.0.0.1', 8081), ': 收到ACK:3')
21  (('127.0.0.1', 8081), ':成功发送数据报8')
```

可见主机2同样实现了服务器和客户端的功能，即实现了双向传输的功能

三、 演示SR传输功能

- 主机1的传输记录

```
(('127.0.0.1', 8080), ' :发送数据1')
(('127.0.0.1', 8080), ' :收到有用ACK1')
(('127.0.0.1', 8080), ' :窗口滑动到2')
(('127.0.0.1', 8080), ' :发送数据2')
(('127.0.0.1', 8080), ' :发送数据3')
(('127.0.0.1', 8080), ' :收到有用ACK3')
(('127.0.0.1', 8080), ' :发送数据4')
(('127.0.0.1', 8080), ' :收到有用ACK4')
(('127.0.0.1', 8080), ' :发送数据5')
(('127.0.0.1', 8080), ' :收到有用ACK5')
(('127.0.0.1', 8080), ' :窗口已满，暂不发送数据')
(('127.0.0.1', 8080), ' :窗口已满，暂不发送数据')
(('127.0.0.1', 8080), '超时重传数据报:2')
(('127.0.0.1', 8080), ' :窗口已满，暂不发送数据')
(('127.0.0.1', 8080), ' :收到有用ACK2')
(('127.0.0.1', 8080), ' :窗口滑动到3')
(('127.0.0.1', 8080), ' :窗口滑动到4')
(('127.0.0.1', 8080), ' :窗口滑动到5')
(('127.0.0.1', 8080), ' :窗口滑动到6')
(('127.0.0.1', 8080), ' :发送数据6')
(('127.0.0.1', 8080), ' :收到有用ACK6')
```

可见当超时事件发生的时候，我们重传的只是单个分组，而不是GBN那样重传所有空中分组

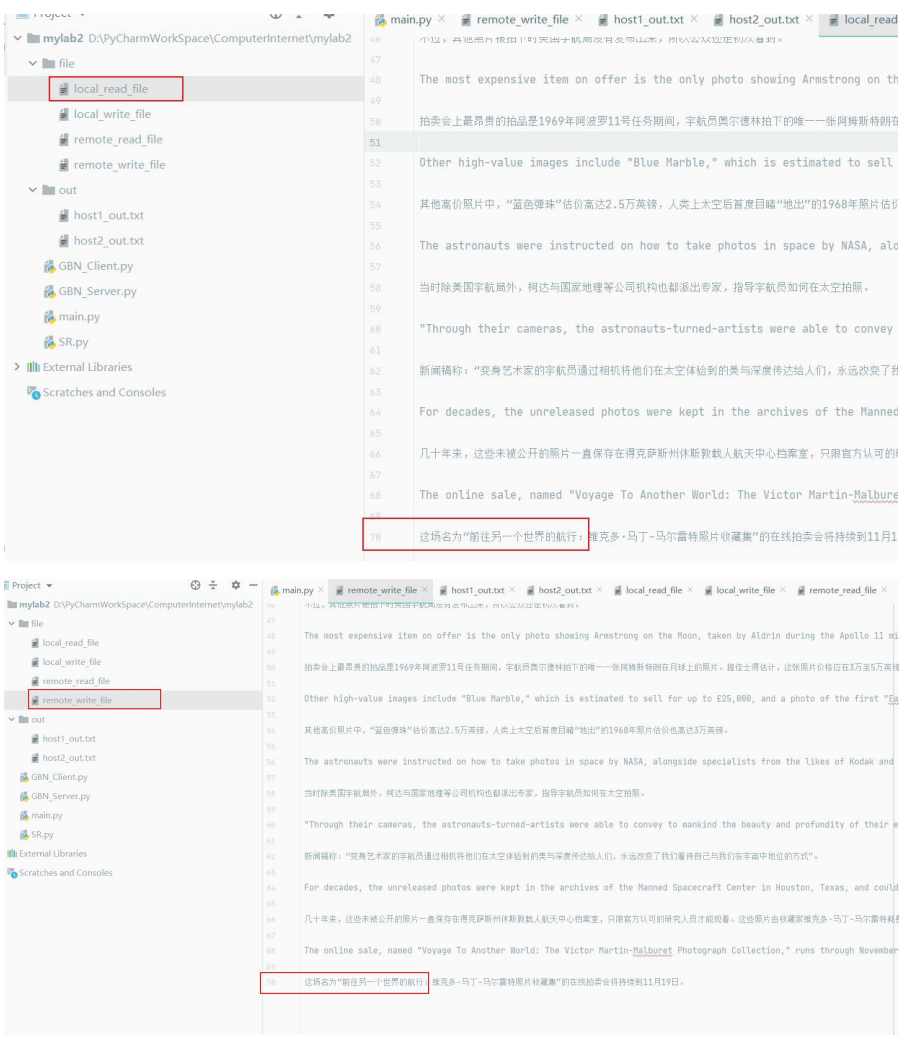
- 主机2的传输记录

```
(('127.0.0.1', 8081), ' :收到数据12')
(('127.0.0.1', 8081), ' :窗口滑动到13')
(('127.0.0.1', 8081), ' :发送ACK12')
(('127.0.0.1', 8081), ' :收到数据13')
(('127.0.0.1', 8081), ' :窗口滑动到14')
(('127.0.0.1', 8081), ' :发送ACK13')
(('127.0.0.1', 8081), ' :收到数据14')
(('127.0.0.1', 8081), ' :窗口滑动到15')
(('127.0.0.1', 8081), ' :发送ACK14')
(('127.0.0.1', 8081), ' :收到数据16')
(('127.0.0.1', 8081), ' :发送ACK16')
(('127.0.0.1', 8081), ' :收到数据17')
(('127.0.0.1', 8081), ' :发送ACK17')
(('127.0.0.1', 8081), ' :收到数据18')
(('127.0.0.1', 8081), ' :发送ACK18')
(('127.0.0.1', 8081), ' :收到数据15')
(('127.0.0.1', 8081), ' :窗口滑动到16')
(('127.0.0.1', 8081), ' :窗口滑动到17')
(('127.0.0.1', 8081), ' :窗口滑动到18')
(('127.0.0.1', 8081), ' :窗口滑动到19')
(('127.0.0.1', 8081), ' :发送ACK15')
(('127.0.0.1', 8081), ' :传输数据结束')
```

可见我们的客户端同样设置了滑动窗口，且当失序数据到达时先缓存，等待一起交付（伴随着窗口的连续滑动）

- 对比文件结果

通过对比服务器读入的文件内容以及客户端写入的文件内容，可以验证我们的SR协议是成功的！
结果如下



问题讨论：

- SR 服务器与 GBN 的主要区别在于：

为每个分组单独设置一个 Clock，当接收到 ACK 时便将对应分组标记为已接收；当遇到超时事件时，单独重发一个分组，而不是将全部的空中分组全部重发。

- SR 客户端与 GBN 的主要区别在于：

在客户端也设置一个滑动窗口，当到达的数据报不在窗口范围内便不予理睬。反之，判断 n 和 rcv_base 是否相同，若相同则滑动窗口并交付数据；若不同则将该数据报缓存起来，而不是像 GBN 那样直接丢弃，然后等待后面一起交付。

- 双向传输可以通过将服务器和客户端的功能集成到一起来实现

体会收获：

- 加深了对GBN协议的过程理解，以及双向传输的实现过程
- 理解了SR和GBN的主要区别在何处
- 掌握了使用python进行Socket编程的方法
- 熟悉了多线程编程方法，让我能够更容易地实现2端通信
- 掌握了基于UDP的通信过程，通过调用sendto(),recvfrom()等函数，我对UDP通信有了更深入的理解