哈爾濱工業大學
Harbin Institute of Technology

计算机网络
课程实验报告

实验名称	可靠数据传输协议-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	瞿久尧		院系	计算学部	
班级	2037101		学号	120L022314	
任课教师	李全龙		指导教师	李全龙	
实验地点	格物 207		实验时间	2022.10.14	
实验课表现	出勤、表现得分(10)		实验报告得分(40)		实验总分	
	操作结果得分(50)				
教师评语					

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

实验目的：

可靠数据传输协议-停等协议的设计与实现：
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。

可靠数据传输协议-GBN 协议的设计与实现：
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：

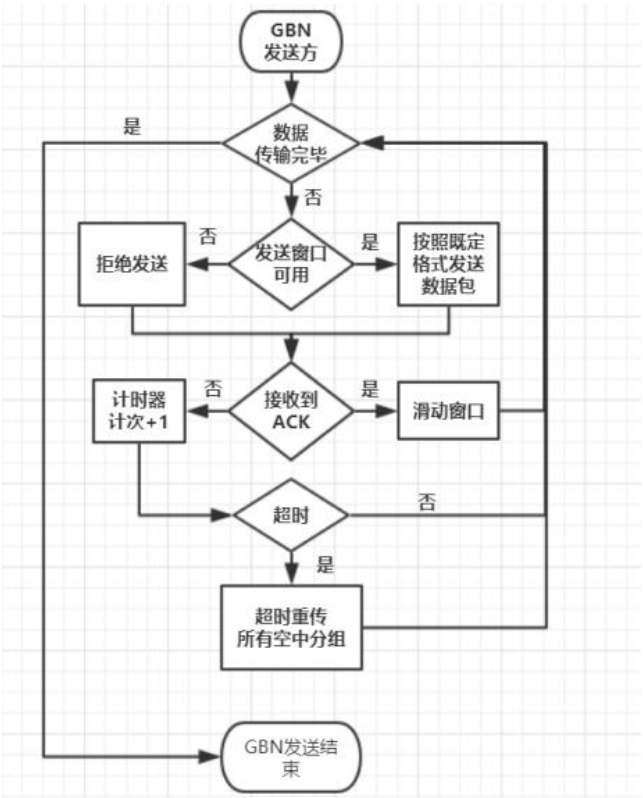
可靠数据传输协议-停等协议的设计与实现：
1) 基于 UDP 设计一个简单的停等协议，实现单向可靠数据传输（服务器到客户的数据传输）。
2) 模拟引入数据包的丢失，验证所设计协议的有效性。
3) 改进所设计的停等协议，支持双向数据传输；
4）基于所设计的停等协议，实现一个 C/S 结构的文件传输应用。

可靠数据传输协议-GBN 协议的设计与实现：
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。

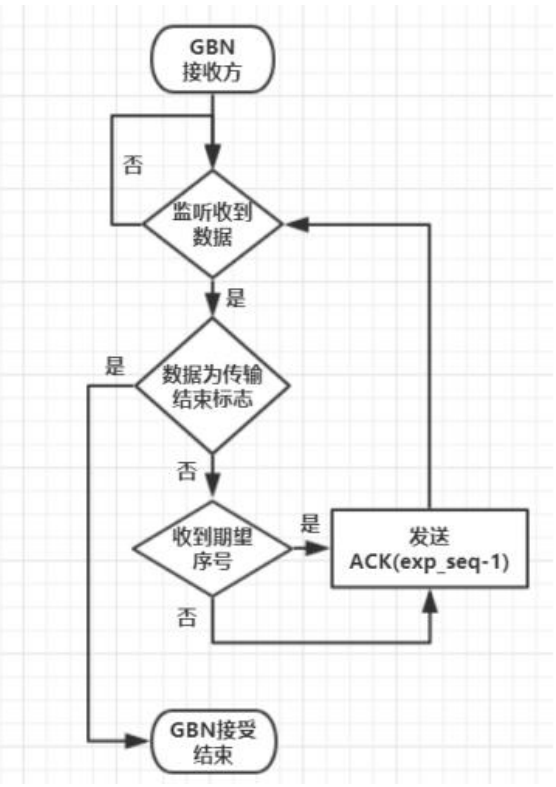
实验过程：

注：停等协议实际上为窗口尺寸为1的GBN，故只验收GBN的流程。
1.GBN协议
在程序实现中，双方并行运行，在套接字存活期间，周期地进入主循环，非阻塞地对对方传过来的数据报进行处理，并对无动作循环（即没有接收到ACK的一个循环）进行计数，进行超时判断。

发送方流程图：


GBN
发送方
数据
传输完毕
是
否
发送窗口
可用
否
是
拒绝发送
按照既定
格式发送
数据包
计时器
计次+1
否
接收到
ACK
是
滑动窗口
超时
否
是
超时重传
所有空中分组
GBN发送结
束

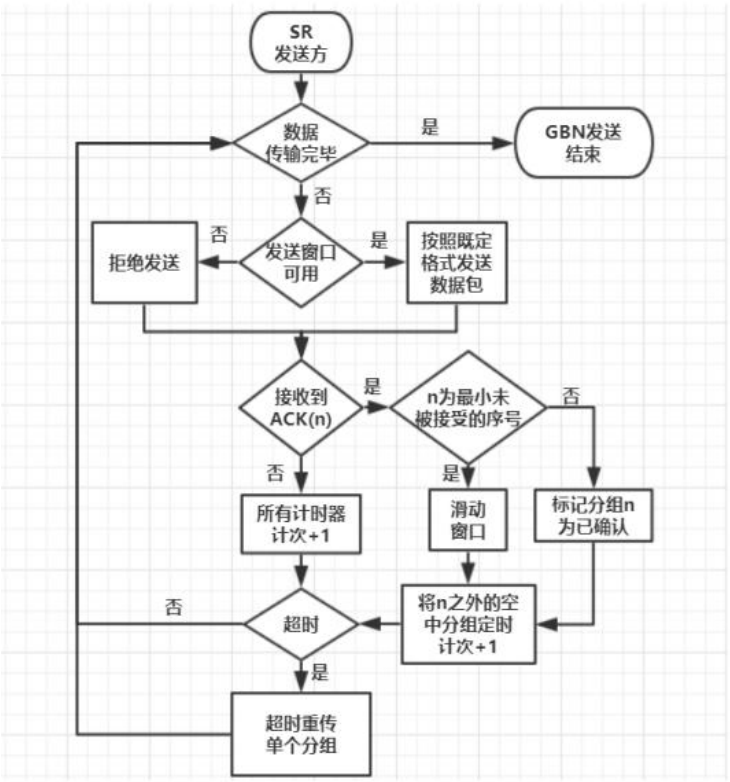
接收方流程图：


GBN
接收方
否
监听收到
数据
是
是
数据为传输
结束标志
否
收到期望
序号
是
发送
ACK(exp_seq-1)
否
GBN接受
结束

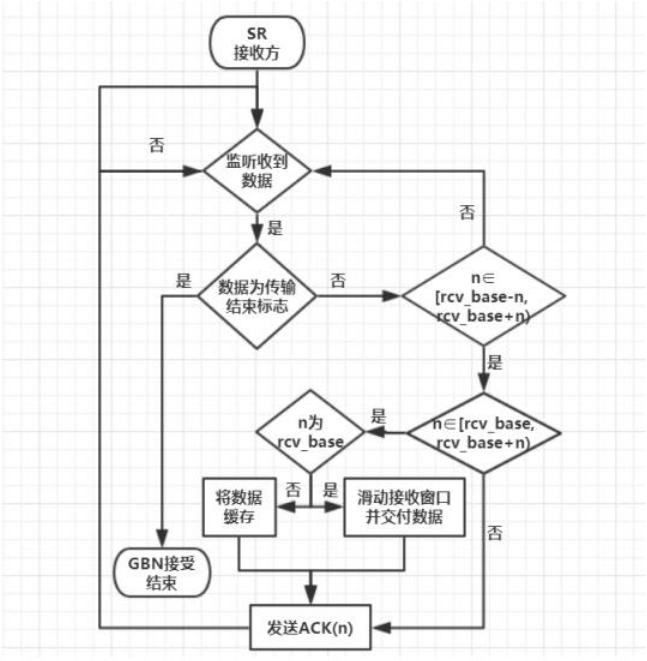
2.SN协议

SR实现时，需要对单个分组记超时时间，并在超时后重传该单个分组，因此可以对单个窗口置计数器，对循环内没有发生动作的窗口均计数加一。

发送方流程图：



接收方流程图：



3.报文结构规约

发送方和接收方对于报文结构作出了约定，规定报文结构为形似:
[seq_num,data]的二进制字节流，seq_num用于标记发送的报文序号。
seq_num为0的字节流[0,ack_num(exp_num-1)]，用于接收方的ACK报文（全双工设计）。
[0 0]报文用于发送方告知数据发送结束，接收方告知已知晓发送结束，双方确认结束后,会结束程序运行。

4.超时计算方法
超时逻辑仅与发送方有关。
GBN：
在一次主循环内，首先执行发送逻辑，判断当前是否有可用窗口，然后非阻塞地接收用户传回的ACK（select方法）：

```
readable = select.select([self.socket], [], [], 1)[0]
```

解析ACK内容，如果在该次循环没有收到ACK，则将超时标记加一。

```
if len(readable) > 0:  # 接收ACK数据
    rcv_ack = self.socket.recvfrom(self.ack_buf_size)[0].decode('gb2312').split()[0]
    print('收到客户端ACK:' + rcv_ack+'',end='\n')
    self.send_base = int(rcv_ack) + 1  # 滑动窗口的起始序号
    self.time_count = 0  # 计时器计次清0
else:  # 未收到ACK包
    self.time_count += 1  # 超时计次+1
    if self.time_count > self.time_out:  # 触发超时重传操作
        self.handle_time_out()
```

超时标记超过限制时，触发超时重传，重传所有空中分组.
SR：
不同处在于，SR将全局超时标记改为每个窗口拥有一个单独的超时标记，如果一个已发送窗口在单次循环中没有收到其指定的ACK，则其超时标记加一，当某个窗口超过限制时，重传该窗口的分组。

5.模拟包丢失
事实上，由于在本次实验中采用了本机对本机的socket接口来模拟数据传输，基本不存在自然的丢包行为。
我们可以采用概率发送的方式，来模拟丢包的行为，在每次发送包之前，调用一次Random方法，当取得的数大于n时 $(n \in [0,1])$，才对包进行发送，则可以模拟发生几率为n的包丢失。

```
if random.random() > self.pkt_loss:
    self.socket.sendto(gbn.util.mk_pkt(self.next_seq, self.data[self.next_seq]),
                       self.remote_address)
```

6.双向数据传输
本次实验实现了半双工的C/S模式,在一次传输中，双方持有固定的客户端或服务器身份，在一次文件传输结束后，可以重新确定身份，传输下一个文件。

实验结果：

可靠数据传输协议-GBN 协议的设计与实现：

窗口大小设为4。

发送成功：

```
import time
def run_gbn(local: tuple([str,int]),
            remote: tuple([str,int]),
            window_num: int = 4,
            window_size: int = 1024):
    threadLock = threading.Lock()
    host_1 = gbn.GBN(local, remote,window_num,window_size)
    host_2 = gbn.GBN(remote, local,window_num,window_size)
    client = threading.Thread(target=host_1.client_run,args=())
    server = threading.Thread(target=host_2.server_run,args=())
    server.start()
    client.start()
    while(host_1.isHostAlive() or host_2.isHostAlive()):
        sleep(0.2)
    host_1.shut_socket()
    host_2.shut_socket()
```


2read.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
窗外的麻雀在电线杆上多嘴
你说这一句很有夏天的感觉
手中的铅笔在纸上来来回回
我用几行字形容你是我的谁
秋刀鱼的滋味猫跟你都想了解
初恋的香味就这样被我们寻回
那温暖的阳光像刚摘的鲜艳草莓
你说你舍不得吃掉这一种感觉
雨下整夜我的爱溢出就像雨水
院子落叶跟我的思念厚厚一叠
几句是非也无法将我的热情冷却
你出现在我诗的每一页
雨下整夜我的爱溢出就像雨水
窗台蝴蝶像诗里纷飞的美丽章节
我接着写
把永远爱你写进诗的结尾
你是我唯一想要的了解
雨下整夜我的爱溢出就像雨水
院子落叶跟我的思念厚厚一叠
几句是非也无法将我的热情冷却
你出现在我诗的每一页
2save.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
窗外的麻雀在电线杆上多嘴
你说这一句很有夏天的感觉
手中的铅笔在纸上来来回回
我用几行字形容你是我的谁
秋刀鱼的滋味猫跟你都想了解
初恋的香味就这样被我们寻回
那温暖的阳光像刚摘的鲜艳草莓
你说你舍不得吃掉这一种感觉
雨下整夜我的爱溢出就像雨水
院子落叶跟我的思念厚厚一叠
几句是非也无法将我的热情冷却
你出现在我诗的每一页
雨下整夜我的爱溢出就像雨水
窗台蝴蝶像诗里纷飞的美丽章节
我接着写
把永远爱你写进诗的结尾
你是我唯一想要的了解
雨下整夜我的爱溢出就像雨水
院子落叶跟我的思念厚厚一叠
几句是非也无法将我的热情冷却
你出现在我诗的每一页

模拟丢包：

```
服务器:成功发送数据0
客户端:收到期望序号数据:0
客户端: 发送ACK0
客户端:收到期望序号数据:1服务器:成功发送数据1

客户端: 发送ACK1
收到客户端ACK:1
服务器:成功发送数据2
客户端:收到期望序号数据:2
客户端: 发送ACK2
收到客户端ACK:2
服务器:成功发送数据3客户端:收到期望序号数据:3

客户端: 发送ACK3
收到客户端ACK:3
服务器:成功发送数据4
客户端:收到期望序号数据:4
客户端: 发送ACK4
客户端:收到期望序号数据:5服务器:成功发送数据5

客户端: 发送ACK5
客户端:收到期望序号数据:6
服务器:成功发送数据6
客户端: 发送ACK6
收到客户端ACK:6
服务器:成功发送数据7
客户端:收到期望序号数据:7
客户端: 发送ACK7
收到客户端ACK:7
服务器:成功发送数据8
客户端:收到期望序号数据:8
客户端: 发送ACK8
收到客户端ACK:8
服务器:成功发送数据9
客户端:收到期望序号数据:9
客户端: 发送ACK9
收到客户端ACK:9
服务器:成功发送数据10客户端:收到期望序号数据:10
```

双向：

```python
def run_gbn(local: tuple([str,int]),
            remote: tuple([str,int]),
            window_num: int = 4,
            window_size: int = 1024):
    threadLock = threading.Lock()
    host_1 = gbn.GBN(local, remote,window_num,window_size,'h1')
    host_2 = gbn.GBN(remote, local,window_num,window_size,'h2')
    client = threading.Thread(target=host_1.server_run,args=())
    server = threading.Thread(target=host_2.client_run,args=())
    server.start()
    client.start()
    while(host_1.isHostAlive() or host_2.isHostAlive()):
        sleep(0.2)
    threading.Thread(target=host_1.client_run).start()
    threading.Thread(target=host_2.server_run).start()
    while(host_1.isHostAlive() or host_2.isHostAlive()):
        sleep(0.2)
    host_1.shut_socket()
    host_2.shut_socket()
```


此电脑 › Work (D:) › Rainbow › Study › 大三秋 › 计算机网络 › 实验 › lab2 › 120L022314-瞿久尧-实验2 › 双向 › GBNnSR › file › gbn
名称 修改日期 类型 大小
h12read.txt 2022/10/14 14:11 文本文档 1 KB
h12save.txt 2022/10/14 15:19 文本文档 1 KB
h22read.txt 2022/10/14 14:11 文本文档 1 KB
h22save.txt 2022/10/14 15:19 文本文档 1 KB
h12read.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
窗外的麻雀在电线杆上多嘴
你说这一句很有夏天的感觉
手中的铅笔在纸上来来回回
我用几行字形容你是我的谁
秋刀鱼的滋味猫跟你都想了解
初恋的香味就这样被我们寻回
那温暖的阳光像刚摘的鲜艳草莓
你说你舍不得吃掉这一种感觉
雨下整夜我的爱溢出就像雨水
院子落叶跟我的思念厚厚一叠
几句是非也无法将我的热情冷却
你出现在我诗的每一页
雨下整夜我的爱溢出就像雨水
窗台蝴蝶像诗里纷飞的美丽章节
我接着写
把永远爱你写进诗的结尾
你是我唯一想要的了解
雨下整夜我的爱溢出就像雨水
院子落叶跟我的思念厚厚一叠
几句是非也无法将我的热情冷却
你出现在我诗的每一页
h12save.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
窗外的麻雀在电线杆上多嘴
你说这一句很有夏天的感觉
手中的铅笔在纸上来来回回
我用几行字形容你是我的谁
秋刀鱼的滋味猫跟你都想了解
初恋的香味就这样被我们寻回
那温暖的阳光像刚摘的鲜艳草莓
你说你舍不得吃掉这一种感觉
雨下整夜我的爱溢出就像雨水
院子落叶跟我的思念厚厚一叠
几句是非也无法将我的热情冷却
你出现在我诗的每一页
雨下整夜我的爱溢出就像雨水
窗台蝴蝶像诗里纷飞的美丽章节
我接着写
把永远爱你写进诗的结尾
你是我唯一想要的了解
雨下整夜我的爱溢出就像雨水
院子落叶跟我的思念厚厚一叠
几句是非也无法将我的热情冷却
你出现在我诗的每一页

h22read.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

窗外的麻雀在电线杆上多嘴
你说这一句很有夏天的感觉
手中的铅笔在纸上来来回回
我用几行字形容你是我的谁
秋刀鱼的滋味猫跟你都想了解
初恋的香味就这样被我们寻回
那温暖的阳光像刚摘的鲜艳草莓
你说你舍不得吃掉这一种感觉
雨下整夜我的爱溢出就像雨水
院子落叶跟我的思念厚厚一叠
几句是非也无法将我的热情冷却
你出现在我诗的每一页
雨下整夜我的爱溢出就像雨水
窗台蝴蝶像诗里纷飞的美丽章节
我接着写
把永远爱你写进诗的结尾
你是我唯一想要的了解
雨下整夜我的爱溢出就像雨水
院子落叶跟我的思念厚厚一叠
几句是非也无法将我的热情冷却
你出现在我诗的每一页

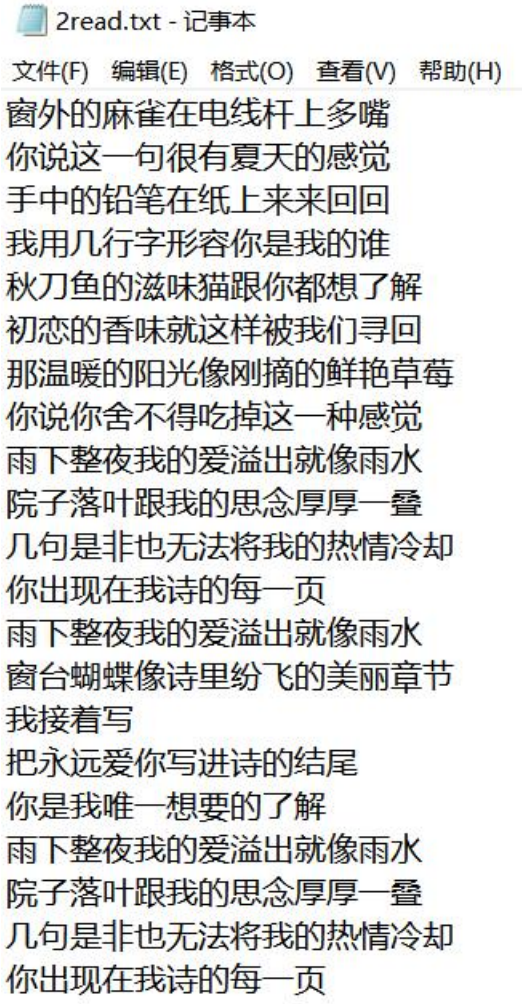
h22save.txt - 记事本

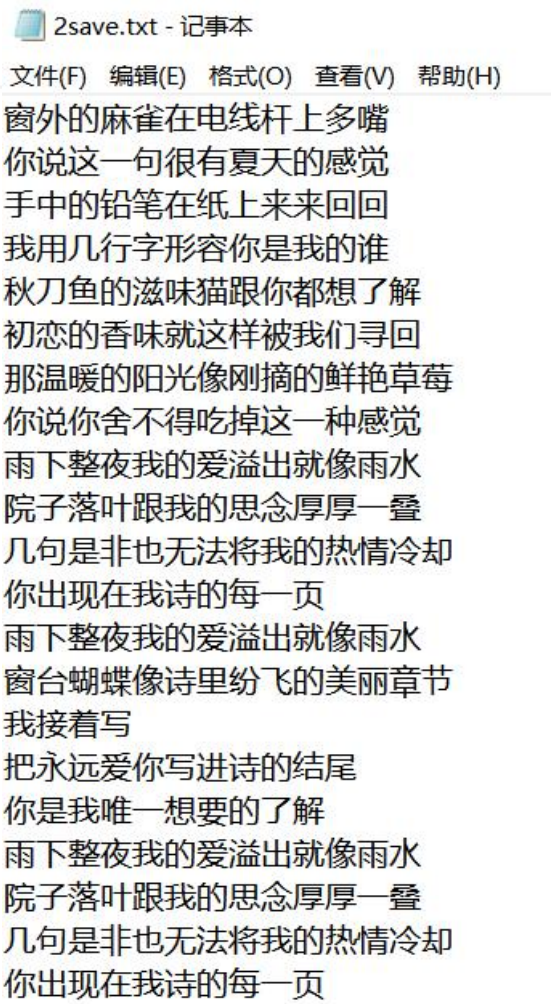
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

窗外的麻雀在电线杆上多嘴
你说这一句很有夏天的感觉
手中的铅笔在纸上来来回回
我用几行字形容你是我的谁
秋刀鱼的滋味猫跟你都想了解
初恋的香味就这样被我们寻回
那温暖的阳光像刚摘的鲜艳草莓
你说你舍不得吃掉这一种感觉
雨下整夜我的爱溢出就像雨水
院子落叶跟我的思念厚厚一叠
几句是非也无法将我的热情冷却
你出现在我诗的每一页
雨下整夜我的爱溢出就像雨水
窗台蝴蝶像诗里纷飞的美丽章节
我接着写
把永远爱你写进诗的结尾
你是我唯一想要的了解
雨下整夜我的爱溢出就像雨水
院子落叶跟我的思念厚厚一叠
几句是非也无法将我的热情冷却
你出现在我诗的每一页

SR：

```
def run_sr( local: tuple([str,int]),
            remote: tuple([str,int]),
            window_num: int = 4,
            window_size: int = 1024):
    host_1 = sr.SR(local, remote,window_num,window_size)
    host_2 = sr.SR(remote, local,window_num,window_size)
    threading.Thread(target=host_1.server_run).start()
    threading.Thread(target=host_2.client_run).start()
    while(host_1.isHostAlive() or host_2.isHostAlive()):
        sleep(0.2)
    host_1.shut_socket()
    host_2.shut_socket()
```





问题讨论：

对实验过程中的思考问题进行讨论或回答。

无。

心得体会：

1. 本次实验原定使用全双工的双向数据传输，但有如下的两个问题：

   1) 在本地传输的前提下，全双工在一次循环中会浪费两倍于半双工的计算资源，并且由于本地充当客户端和服务器端会出现重复调用同一 PORT 的错误，双方需要分时地访问 PORT，因此需要引入全局的分时，而这实际极大减缓了程序运行(在半双工中由于客户端即时地接收和处理，基本可以忽略其运行时间)
   2) 为了防止接口资源流失，PORT 应该在调用后及时关闭，然而在全双工中，由于无法及时判定当前 PORT 是否处于使用状态(报文不是连续不断到达的)，因此不能及时关闭 Socket 接口，而半双工则可以在一次文件传输结束后立即关闭 PORT，直到下一次需要时再打开。

2. 采用超时机制，对于丢包的反应是缓慢的(依靠服务器端的超时)，而且超时时间需要根据链路状态不断地更新，容易因为过早的超时而浪费资源，而且一次丢包就需要进行大量的重发，而且对链路情况不知情的服务器端进行大量重发，很容易又会加重链路的负担。TCP的“快速重传”机制兼顾了二者，是很优秀的发明。

附录：程序源代码

单向:gbn.py

```
import random
import select
import socket


class util:
    @staticmethod
    def mk_pkt(seqSen, seqRec, msg):
        return (str(seqSen) + ' ' + str(seqRec) + ' ' + str(msg)).encode('gb2312', 'ignore')

    @staticmethod
    def mk_pkt1(seq, msg):
        return (str(seq) + ' ' + str(msg)).encode('gb2312', 'ignore')


class GBN:

    def __init__(self,
                 local_address: tuple[[str, int]],
                 remote_address: tuple[[str, int]],
                 inputWindow_size: int = 4,
                 inputPkg_size: int = 1024,
                 hostNameInput: str = ''):

        self.hostname = hostNameInput
        self.fin_flag = 0
```

```
        self.pkg_size = inputPkg_size  # 单格窗口大小
        self.window_size = inputWindow_size  # 窗口尺寸
        self.send_base = 0  # 发送窗口的最左序号
        self.next_seq = 0  # 当前未被利用的序号
        self.time_count = 0  # 记录当前传输时间
        self.time_out = 5  # 设置超时时间
        self.local_address = local_address  # 设置本地 socket 地址
        self.remote_address = remote_address  # 设置远程 socket 地址
        self.data = []  # 缓存发送数据
        self.read_path = 'file/gbn/' + self.hostname + '2read.txt'  # 需要发送的源文件数据
        self.ack_buf_size = 1678
        self.get_data_from_file()

        self.data_buf_size = 1678  # 作为客户端接收数据缓存
        self.exp_seq = 0  # 当前期望收到该序号的数据
        self.save_path = 'file/gbn/' + self.hostname + '2save.txt'  # 接收数据时，保存数据的地址
        self.write_data_to_file('', mode='w')

        self.pkt_loss = 0.1  # 发送数据丢包率
        self.ack_loss = 0.1  # 返回的 ack 丢包率
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.bind(self.local_address)
        self.state_flag = 0

    def send_data(self):
        if self.send_base == len(self.data):  # data 数据已全部发送并收到 ack
            self.socket.sendto(util.mk_pkt(0, self.exp_seq - 1, 0), self.remote_address)
            print('服务器: ' + self.hostname + '\t' + '发送完毕!')
            return
        if self.next_seq == len(self.data):  # data 数据已全部发送，无需发送
            return
        if self.next_seq - self.send_base < self.window_size:  # 窗口中仍有可用空间
            if random.random() > self.pkt_loss:  # 随机产生丢包行为
                self.socket.sendto(util.mk_pkt(self.next_seq, self.exp_seq - 1, self.data[self.next_seq]),
                                   self.remote_address)
                isLose = 0
            else:
                isLose = 1
            print('服务器: ' + self.hostname + '\t' + '成功发送数据: ' + str(self.next_seq)
                  + ' ack=' + str(self.exp_seq - 1), end='\n')
            if isLose == 1:
                print('服务器: ' + self.hostname + '\t' + '丢失数据包: ' + str(self.next_seq))
            self.next_seq = self.next_seq + 1
        else:  # 窗口中无可用空间
```

```
            print('服务器: ' + self.hostname + '\t' + '窗口已满，暂不发送数据', end='\n')

    # 超时处理函数：计时器置 0
    def handle_time_out(self):
        print('服务器: ' + self.hostname + '\t' + '数据包: ' + str(self.send_base) + '超时，开始重传!', end='\n')
        self.time_count = 0  # 超时计次重启
        for i in range(self.send_base, self.next_seq):  # 发送空中的所有分组
            if random.random() > self.pkt_loss:  # 概率性重传
                self.socket.sendto(util.mk_pkt(i, self.exp_seq - 1, self.data[i]), self.remote_address)
                isLose = 0
            else:
                isLose = 1
            print('服务器: ' + self.hostname + '\t' + '数据: ' + str(i)
                  + ' ack=' + str(self.exp_seq - 1) + '已重发!', end='\n')
            if isLose == 1:
                print('服务器: ' + self.hostname + '\t' + '丢失数据包: ' + str(i), end='\n')

    # 从文本中读取数据用于模拟上层协议数据的到来
    def get_data_from_file(self):
        f = open(self.read_path, 'r', encoding='gb2312')
        while True:
            send_data = f.read(self.pkg_size)
            if len(send_data) <= 0:
                break
            self.data.append(send_data)  # 将读取到的数据保存到 data 数据结构中

    # 线程执行函数，不断发送数据并接收 ACK 报文做相应的处理
    def server_run(self):
        self.state_flag = 1
        rcv_finish = 0
        send_finish = 0
        while True:
            self.send_data()  # 发送数据逻辑
            readable = select.select([self.socket], [], [], 1)[0]
            if len(readable) > 0:  # 接收到数据
                rcv_data = self.socket.recvfrom(self.data_buf_size)[0].decode('gb2312')
                rcv_seq = rcv_data.split()[0]  # 按照格式规约获取数据序号
                rcv_ack = rcv_data.split()[1]
                rcv_data = rcv_data.replace(rcv_seq + ' ' + rcv_ack + ' ', '')  # 按照格式规约获取数据
                print('服务器: ' + self.hostname + '\t' + '接收到数据: ' + rcv_seq + ' ack: ' + rcv_ack, end='\n')
                if int(rcv_seq) == self.exp_seq:  # 接收到按序数据包
                    print('服务器: ' + self.hostname + '\t' + '接收到期望序号数据: ' + rcv_seq, end='\n')
                    self.write_data_to_file(rcv_data)  # 保存服务器端发送的数据到本地文件中
                    self.exp_seq = self.exp_seq + 1  # 期望数据的序号更新
```

```
                elif rcv_seq == '0' and rcv_data == '0':  # 接收到结束包
                    print('服务器: ' + self.hostname + '\t' + '接收数据结束', end='\n')
                    rcv_finish = 1
                else:
                    print('服务器: ' + self.hostname + '\t' + '接收到非期望数据，期望: ' +
                          str(self.exp_seq) + '实际:' + str(rcv_seq), end='\n')
                if int(rcv_ack) < self.send_base:
                    self.time_count += 1
                else:
                    self.send_base = int(rcv_ack) + 1  # 滑动窗口的起始序号
                    self.time_count = 0
                print('服务器: ' + self.hostname + '\t' + 'self.send_base=' + str(self.send_base), end='\n')
            else:  # 未收到包
                self.time_count += 1  # 超时计次+1
            if self.time_count > self.time_out:  # 触发超时重传操作
                self.handle_time_out()
            if self.send_base == len(self.data):  # 判断数据是否传输结束
                self.socket.sendto(util.mk_pkt(0, self.exp_seq - 1, 0), self.remote_address)  # 发送结束报文
                print('服务器: ' + self.hostname + '\t' + '发送完毕', end='\n')
                send_finish = 1
            if rcv_finish == 1 and send_finish == 1:
                print('服务器: ' + self.hostname + '\t' + '发送接收均完毕!!', end='\n')
                break
        self.state_flag = 0

    # 保存来自服务器的合适的数据
    def write_data_to_file(self, data, mode='a'):
        with open(self.save_path, mode, encoding='gb2312') as f:
            f.write(data)  # 模拟将数据交付到上层

    def isHostAlive(self):
        return self.state_flag

    def shut_socket(self):
        self.socket.close()
```

main.py

```
import threading
from time import sleep
import gbn


def run_gbn(local: tuple[[str, int]],
            remote: tuple[[str, int]],
```

```
                window_num: int = 4,
                window_size: int = 1024):
    threading.Lock()
    host_1 = gbn.GBN(local, remote, window_num, window_size)
    host_2 = gbn.GBN(remote, local, window_num, window_size)
    client = threading.Thread(target=host_1.client_run, args=())
    server = threading.Thread(target=host_2.server_run, args=())
    server.start()
    client.start()
    while host_1.isHostAlive() or host_2.isHostAlive():
        sleep(0.2)
    host_1.shut_socket()
    host_2.shut_socket()


#print('停等协议:\n')
#run_gbn(('127.0.0.1', 5000), ('127.0.0.1', 5001), 1, 10)
print('gbn:\n')
run_gbn(('127.0.0.1', 5000), ('127.0.0.1', 5001), 4, 10)
```

双向 gbn.py

```
import random
import select
import socket


class util:
    @staticmethod
    def mk_pkt(seqSen, seqRec, msg):
        return (str(seqSen) + ' ' + str(seqRec) + ' ' + str(msg)).encode('gb2312', 'ignore')

    @staticmethod
    def mk_pkt1(seq, msg):
        return (str(seq) + ' ' + str(msg)).encode('gb2312', 'ignore')


class GBN:

    def __init__(self,
                 local_address: tuple[[str, int]],
                 remote_address: tuple[[str, int]],
                 inputWindow_size: int = 4,
                 inputPkg_size: int = 1024,
```

```
                hostNameInput: str = ''):

        self.hostname = hostNameInput
        self.fin_flag = 0
        self.pkg_size = inputPkg_size   # 单格窗口大小
        self.window_size = inputWindow_size   # 窗口尺寸
        self.send_base = 0   # 发送窗口的最左序号
        self.next_seq = 0   # 当前未被利用的序号
        self.time_count = 0   # 记录当前传输时间
        self.time_out = 5   # 设置超时时间
        self.local_address = local_address   # 设置本地 socket 地址
        self.remote_address = remote_address   # 设置远程 socket 地址
        self.data = []   # 缓存发送数据
        self.read_path = 'file/gbn/' + self.hostname + '2read.txt'   # 需要发送的源文件数据
        self.ack_buf_size = 1678
        self.get_data_from_file()

        self.data_buf_size = 1678   # 作为客户端接收数据缓存
        self.exp_seq = 0   # 当前期望收到该序号的数据
        self.save_path = 'file/gbn/' + self.hostname + '2save.txt'   # 接收数据时，保存数据的地址
        self.write_data_to_file('', mode='w')

        self.pkt_loss = 0.1   # 发送数据丢包率
        self.ack_loss = 0.1   # 返回的 ack 丢包率
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.bind(self.local_address)
        self.state_flag = 0

    def send_data(self):
        if self.send_base == len(self.data):   # data 数据已全部发送并收到 ack
            self.socket.sendto(util.mk_pkt(0, self.exp_seq - 1, 0), self.remote_address)
            print('服务器: ' + self.hostname + '\t' + '发送完毕!')
            return
        if self.next_seq == len(self.data):   # data 数据已全部发送，无需发送
            return
        if self.next_seq - self.send_base < self.window_size:   # 窗口中仍有可用空间
            if random.random() > self.pkt_loss:   # 随机产生丢包行为
                self.socket.sendto(util.mk_pkt(self.next_seq, self.exp_seq - 1, self.data[self.next_seq]),
                                   self.remote_address)
                isLose = 0
            else:
                isLose = 1
            print('服务器: ' + self.hostname + '\t' + '成功发送数据: ' + str(self.next_seq)
                  + ' ack=' + str(self.exp_seq - 1), end='\n')
```

```
            if isLose == 1:
                print('服务器: ' + self.hostname + '\t' + '丢失数据包: ' + str(self.next_seq))
            self.next_seq = self.next_seq + 1
        else:  # 窗口中无可用空间
            print('服务器: ' + self.hostname + '\t' + '窗口已满，暂不发送数据', end='\n')

# 超时处理函数：计时器置 0
def handle_time_out(self):
    print('服务器: ' + self.hostname + '\t' + '数据包: ' + str(self.send_base) + '超时，开始重传!', end='\n')
    self.time_count = 0  # 超时计次重启
    for i in range(self.send_base, self.next_seq):  # 发送空中的所有分组
        if random.random() > self.pkt_loss:  # 概率性重传
            self.socket.sendto(util.mk_pkt(i, self.exp_seq - 1, self.data[i]), self.remote_address)
            isLose = 0
        else:
            isLose = 1
        print('服务器: ' + self.hostname + '\t' + '数据: ' + str(i)
              + ' ack=' + str(self.exp_seq - 1) + '已重发!', end='\n')
        if isLose == 1:
            print('服务器: ' + self.hostname + '\t' + '丢失数据包: ' + str(i), end='\n')

# 从文本中读取数据用于模拟上层协议数据的到来
def get_data_from_file(self):
    f = open(self.read_path, 'r', encoding='gb2312')
    while True:
        send_data = f.read(self.pkg_size)
        if len(send_data) <= 0:
            break
        self.data.append(send_data)  # 将读取到的数据保存到 data 数据结构中

# 线程执行函数，不断发送数据并接收 ACK 报文做相应的处理
def server_run(self):
    self.state_flag = 1
    rcv_finish = 0
    send_finish = 0
    while True:
        self.send_data()  # 发送数据逻辑
        readable = select.select([self.socket], [], [], 1)[0]
        if len(readable) > 0:  # 接收到数据
            rcv_data = self.socket.recvfrom(self.data_buf_size)[0].decode('gb2312')
            rcv_seq = rcv_data.split()[0]  # 按照格式规约获取数据序号
            rcv_ack = rcv_data.split()[1]
            rcv_data = rcv_data.replace(rcv_seq + ' ' + rcv_ack + ' ', '')  # 按照格式规约获取数据
            print('服务器: ' + self.hostname + '\t' + '接收到数据: ' + rcv_seq + ' ack: ' + rcv_ack, end='\n')
```

```
                if int(rcv_seq) == self.exp_seq:  # 接收到按序数据包
                    print('服务器: ' + self.hostname + '\t' + '接收到期望序号数据: ' + rcv_seq, end='\n')
                    self.write_data_to_file(rcv_data)  # 保存服务器端发送的数据到本地文件中
                    self.exp_seq = self.exp_seq + 1  # 期望数据的序号更新
                elif rcv_seq == '0' and rcv_data == '0':  # 接收到结束包
                    print('服务器: ' + self.hostname + '\t' + '接收数据结束', end='\n')
                    rcv_finish = 1
                else:
                    print('服务器: ' + self.hostname + '\t' + '接收到非期望数据，期望: ' +
                          str(self.exp_seq) + '实际:' + str(rcv_seq), end='\n')
                if int(rcv_ack) < self.send_base:
                    self.time_count += 1
                else:
                    self.send_base = int(rcv_ack) + 1  # 滑动窗口的起始序号
                    self.time_count = 0
                print('服务器: ' + self.hostname + '\t' + 'self.send_base=' + str(self.send_base), end='\n')
            else:  # 未收到包
                self.time_count += 1  # 超时计次+1
            if self.time_count > self.time_out:  # 触发超时重传操作
                self.handle_time_out()
            if self.send_base == len(self.data):  # 判断数据是否传输结束
                self.socket.sendto(util.mk_pkt(0, self.exp_seq - 1, 0), self.remote_address)  # 发送结束报文
                print('服务器: ' + self.hostname + '\t' + '发送完毕', end='\n')
                send_finish = 1
            if rcv_finish == 1 and send_finish == 1:
                print('服务器: ' + self.hostname + '\t' + '发送接收均完毕!!', end='\n')
                break
        self.state_flag = 0

    # 保存来自服务器的合适的数据
    def write_data_to_file(self, data, mode='a'):
        with open(self.save_path, mode, encoding='gb2312') as f:
            f.write(data)  # 模拟将数据交付到上层

    def isHostAlive(self):
        return self.state_flag

    def shut_socket(self):
        self.socket.close()
```

sr.py

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

```
import gbn


class SR:

    def __init__(self,
                 local_address: tuple[[str, int]],
                 remote_address: tuple[[str, int]],
                 inputWindow_size: int = 4,
                 inputPkg_size: int = 1024):

        self.pkg_size = inputPkg_size  # 单格窗口大小
        self.send_window_size = inputWindow_size  # 窗口尺寸
        self.send_base = 0  # 最小的被发送的分组序号
        self.next_seq = 0  # 当前未被利用的序号
        self.time_out = 5  # 设置超时时间
        self.local_address = local_address  # 设置本地 socket 地址
        self.remote_address = remote_address  # 设置远程 socket 地址
        self.data = []  # 缓存发送数据
        self.read_path = 'file/sr/2read.txt'  # 需要发送的源文件数据
        self.ack_buf_size = 10
        self.get_data_from_file()

        self.rcv_window_size = inputWindow_size  # 接受窗口尺寸
        self.data_buf_size = 1678  # 作为客户端接收数据缓存
        self.save_path = 'file/sr/2save.txt'  # 接收数据时，保存数据的地址
        self.write_data_to_file('', mode='w')

        self.pkt_loss = 0.1  # 发送数据丢包率
        self.ack_loss = 0  # 返回的 ack 丢包率

        self.time_counts = {}  # 存储窗口中每个发出序号的时间
        self.ack_seqs = {}  # 储存窗口中每个序号的 ack 情况

        self.rcv_base = 0  # 最小的需要接收的数据的分组序号
        self.rcv_data = {}  # 缓存失序的接收数据
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.bind(self.local_address)

        self.state_flag = 0

    # 若仍剩余窗口空间，则构造数据报发送；否则拒绝发送数据
    def send_data(self):
        if self.next_seq == len(self.data):  # 判断是否还有缓存数据可以发送
```

```
            print('服务器:发送完毕，等待确认')
            return
        if self.next_seq < self.send_base + self.send_window_size:  # 窗口中仍有可用空间
            if random.random() > self.pkt_loss:
                self.socket.sendto(gbn.util.mk_pkt1(self.next_seq, self.data[self.next_seq]),
                                   self.remote_address)
            self.time_counts[self.next_seq] = 0  # 设置计时器
            self.ack_seqs[self.next_seq] = False  # 设置为未接受确认包
            print('服务器:发送数据' + str(self.next_seq))
            self.next_seq += 1
        else:  # 窗口中无可用空间
            print('服务器:窗口已满，暂不发送数据')

    # 超时处理函数：计时器置 0，设为未接受 ACK，同时发送该序列号数据
    def handle_time_out(self, time_out_seq):
        print('超时重传:' + str(time_out_seq))
        self.time_counts[time_out_seq] = 0  # 重新定时
        if random.random() > self.pkt_loss:  # 随机发送数据包
            self.socket.sendto(gbn.util.mk_pkt1(time_out_seq, self.data[time_out_seq]),
                               self.remote_address)

    # 从文件中读取数据，并存储到 data 属性里
    def get_data_from_file(self):
        f = open(self.read_path, 'r', encoding='gb2312')
        while True:
            send_data = f.read(self.pkg_size)  # 一次读取 1024 个字节（如果有这么多）
            if len(send_data) <= 0:
                break
            self.data.append(send_data)  # 将读取到的数据保存到 data 数据结构中

    # 滑动窗口，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('服务器:窗口滑动到' + str(self.send_base))

    def server_run(self):
        self.state_flag = 1
        while True:
            self.send_data()  # 发送数据
            readable = select.select([self.socket], [], [], 0.1)[0]  # 非阻塞方法
            if len(readable) > 0:  # 接收 ACK 数据
```

```
                rcv_ack = self.socket.recvfrom(self.ack_buf_size)[0].decode('gb2312').split()[0]
                if self.send_base <= int(rcv_ack) < self.next_seq:  # 收到 ack，则标记为已确认且超时计数为 0
                    print('服务器:收到有用 ACK' + rcv_ack)
                    self.ack_seqs[int(rcv_ack)] = True  # 设为已接受
                    if self.send_base == int(rcv_ack):  # 收到的 ack 为最小的窗口序号
                        self.slide_send_window()  # 则滑动窗口
                else:
                    print('服务器:收到无用 ACK' + rcv_ack)
            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
            if self.send_base == len(self.data):  # 数据传输结束
                self.socket.sendto(gbn.util.mk_pkt1(0, 0), self.remote_address)  # 发送传输结束包
                print('服务器:数据传输结束')
                break
        self.state_flag = 0

    # 保存来自服务器的合适的数据
    def write_data_to_file(self, data, mode='a'):
        with open(self.save_path, mode, encoding='gb2312') as f:
            f.write(data)

    # 主要执行函数，不断接收服务器发送的数据，若失序则保存到缓存；若按序则滑动窗口；否则丢弃
    def client_run(self):
        self.state_flag = 1
        while True:
            readable = select.select([self.socket], [], [], 0.1)[0]  # 非阻塞接受
            if len(readable) > 0:
                rcv_data = self.socket.recvfrom(self.data_buf_size)[0].decode('gb2312')
                rcv_seq = rcv_data.split()[0]  # 提取数据包序号
                rcv_data = rcv_data.replace(rcv_seq + ' ', '')  # 提取数据包数据
                if rcv_seq == '0' and rcv_data == '0':  # 收到传输数据结束的标志
                    print('客户端:传输数据结束')
                    break
                print('客户端:收到数据' + rcv_seq)
                if self.rcv_base - self.rcv_window_size <= int(
                        rcv_seq) < self.rcv_base + self.rcv_window_size:
                    if self.rcv_base <= int(rcv_seq) < self.rcv_base + self.rcv_window_size:  # 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(gbn.util.mk_pkt1(int(rcv_seq), 0), self.remote_address)
                print('客户端:发送 ACK' + rcv_seq)
        self.state_flag = 0

    # 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('客户端:窗口滑动到' + str(self.rcv_base))

    def isHostAlive(self):
        return self.state_flag

    def shut_socket(self):
        self.socket.close()
```

main.py

```
import threading
from time import sleep
import gbn
import sr


def run_gbn(local: tuple[[str, int]],
            remote: tuple[[str, int]],
            window_num: int = 4,
            window_size: int = 1024):
    threading.Lock()
    host_1 = gbn.GBN(local, remote, window_num, window_size, 'h1')
    host_2 = gbn.GBN(remote, local, window_num, window_size, 'h2')
    server1 = threading.Thread(target=host_1.server_run, args=())
    server2 = threading.Thread(target=host_2.server_run, args=())
    server1.start()
    server2.start()
    while host_1.isHostAlive() or host_2.isHostAlive():
        sleep(0.2)
    host_1.shut_socket()
    host_2.shut_socket()
```

```
def run_sr(local: tuple[[str, int]],
                remote: tuple[[str, int]],
                window_num: int = 4,
                window_size: int = 1024):
    host_1 = sr.SR(local, remote, window_num, window_size)
    host_2 = sr.SR(remote, local, window_num, window_size)
    threading.Thread(target=host_1.server_run).start()
    threading.Thread(target=host_2.client_run).start()
    while host_1.isHostAlive() or host_2.isHostAlive():
        sleep(0.2)
    host_1.shut_socket()
    host_2.shut_socket()


#print('gbn:\n')
#run_gbn(('127.0.0.1', 5000), ('127.0.0.1', 5001), 4, 10)
print('sr:\n')
run_sr(('127.0.0.1', 5000), ('127.0.0.1', 5001), 4, 10)
```