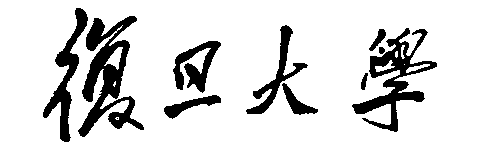
****

**计算机网络与安全 · 期末设计实验报告**

**专业班级：** 信息安全 **学号：** 19307130247 **姓名：** 刘定一

1. **设计目的**

进一步深入理解传输层协议：滑动窗口协议的基本原理;掌握 GBN/SR 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,并在此基础上改进实现选择重传（SR）协议。

1. **设计要求**

1) 基于UDP设计一个简单的GBN协议，实现双向可靠数据传输。

2) 在此基础上改进并实现SR协议。

3) 模拟引入数据包的丢失，验证所设计协议的有效性。

扩展：在此基础上可以增加更多功能，如拥塞控制。

1. **实验原理及过程**

本次实验中，实现了要求的1)、2)、3)，**并且在GBN中实现了简单的拥塞控制**（SR本身控制较好，我认为无需再做拥塞控制）：根据贪婪原则，通过判断接收到的ACK的完整情况动态改变滑动窗口的大小。如果实时丢包率小，则扩大滑动窗口，一次发送更多的包，反之则缩小滑动窗口，减少无效传输（但是存在最小值，代码中设为3），具体原理和代码实现见下：

**（一）、基于UDP设计一个简单的GBN协议，实现双向可靠数据传输**

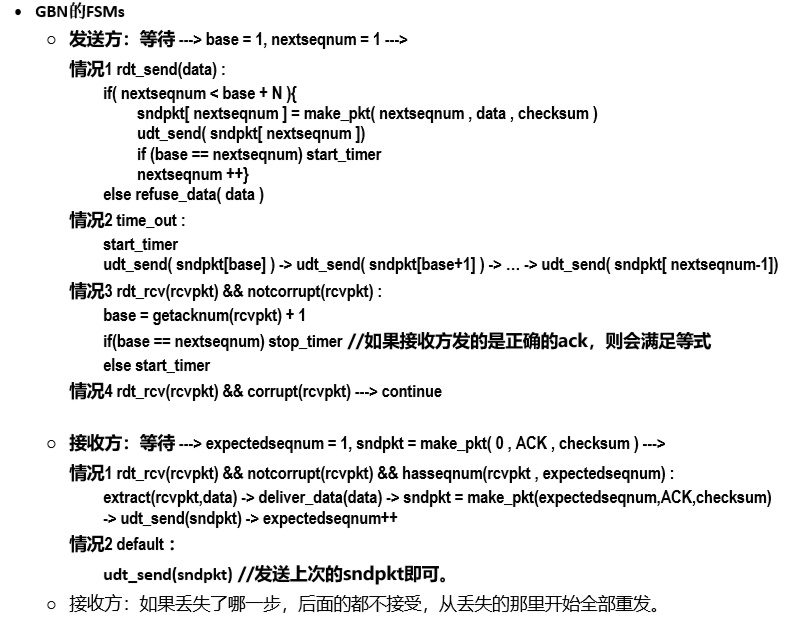
1.实验原理介绍

GBN的原理可以比较简单的概括：对于发送方，将数据按照发送窗口大小的数目发送，只要接收到发送窗口内的ACK，则右移到ACK的下一个位置，否则，在超时时间后重传发送窗口中的所有数据；对于接收方，如果收到了期望的包，则返回对应的ACK，否则，返回最后接收到的期望包的ACK。

由于要实现双向可靠数据传输，所有Client和Server的界限将变得模糊一些。两者都需要同时具备Client和Server的功能：发送数据，接收数据，发送ACK，接收ACK，并分析这些报文。

2.具体实验过程

这里以实现GNBClient为例（其实GNBServer是几乎一样的），详细介绍GBN的以python为基础的代码实现，实现包括了双向的可靠数据传输和简单的拥塞控制。

在伪代码中，先观察GNB的状态机实现（代码中有注释）：

图一：GBN的FSMs状态机伪代码实现

于是根据伪代码，可以开始编写程序GBNClient.py：

1. 初始化变量def \_\_init\_\_(self)：

主要有三个部分的变量：第一种，GBN的基本变量，包括base（已删去，python可以简化地直接移动数组），nextseqnum（待发序列号），timer（包括等待ACK和等待数据时间），window（滑动窗口），rev\_data（接收的数据），N（窗口大小），fd\_num（发送包的总数量）；第二种，socket连接变量，包括addr（本机地址和端口），server\_addr（服务器地址和端口），socket（套接字）；第三种，**拥塞控制所需要的变量，包括timer（循环计数器），add\_condition（窗口增加条件），add\_window（窗口增加大小），origin\_N（原始窗口大小）。**

1. 主要函数send(self,buffer)：在无限循环中负责所有交互
   1. 初始化变量：

fd\_timer（client相关变量：接收ACK计时）

data\_timer（server相关变量：接收数据计时）

fd\_num（client相关变量：传入buffer的length）

last\_ack（server变量：相关最新的ACK）。

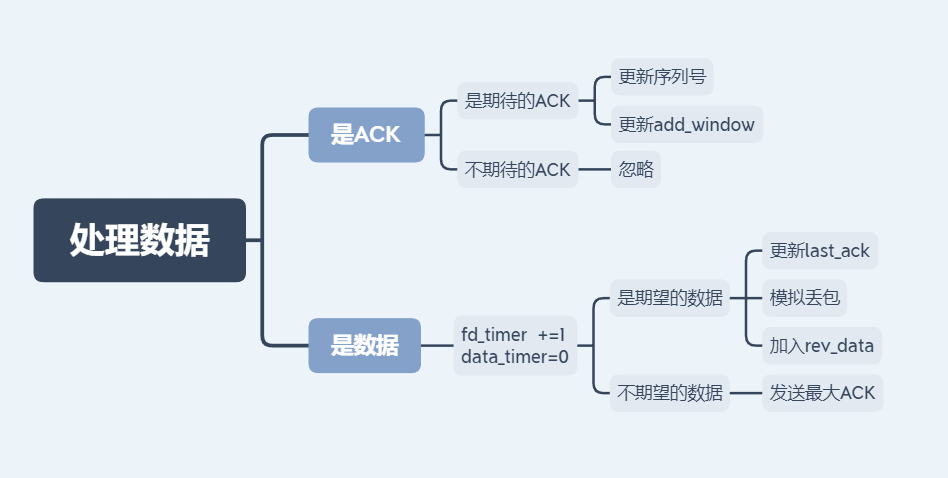
* 1. 在无限循环中，实现发送数据的代码：

每次循环等待时间为0.5s，每次timer加1，滑动窗口为origin\_N + add\_window输出当前滑动窗口（用以体现拥塞控制）

1. 情况1，窗口未满。如果仍有数据未发送，则将最多N个数据封装为可解码的报文放入window，并更新序列号。这里暂时不发送数据，而是和超时重传数据一起发送，以便简化代码。
2. 情况2，超时情况，当判断当前发送窗口的第一个包超时后仍未收到ACK时，将add\_window减半，并且将当前窗口的所有包由已发送标记为未发送，方便后续发送。
3. 特殊情况：退出。容易知道，退出时应该：发送数据和接收数据全部完成，对应：发送数据的最后一个包的ACK已收到和最后一个接收数据的ACK已发送。即：等待一段时间后既没有收到ACK也没有收到数据。退出时，将数据写入sreceive.txt。
4. 特殊情况：发送。将窗口内所有标记为未发送的包全部发送，同时将各包标记为已发送。
5. 情况3&4：接收数据。首先这里使用了select模块实现无阻塞式的socket连接，即进程指定内核监听文件描述符，当没有文件描述符事件发生时,进程被阻塞;当一个或者多个文件描述符事件发生时,进程被唤醒[[1]](#footnote-1)。可以较为方便地处理接二连三地发送的数据。

收到数据后，首先，处理数据。

***[在这里详细介绍Helper.py中的Helper类，其中有初始化变量：data（有效载荷），is\_ack（报文类型，若为ack，则为1，否则，为0），seq（报文序列号），is\_send（是否发送，已发送为1，否则为0，在SR中，添加功能，若已发送且已获得ACK，则为2）。同时，帮助封装数据为seq + ack + data ,其中三者之间用逗号隔开，方便用正则表达式切割。]***

在这里主要介绍处理数据的逻辑，具体实现代码中有详细注释：

图二：收到数据后的处理逻辑

首先，判断是ACK还是数据。

如果是ACK，判断是否为期待ACK。如果是，更新窗口，并且连续收到窗口内所有发送数据包的ACK，则将add\_window加1。否则，忽略，等待超时。

如果是数据，判断是否为期待数据。如果是，模拟丢包或丢失ACK，这里使用random函数，使丢包率为20%，如果判定丢包，那么一半可能丢失数据包，一半可能丢失ACK（即收到了数据，但发送的ACK对方无法收到）。否则，返回最大ACK。

1. 启动函数 start(self)，并启动：

从csend.txt文件中每次读取100字节长度放入buf中，最后调用sendto函数发起测试，调用send函数开始传输数据。最后构造GBNClient类实例并调用start函数即可。

3. 实验总结

到此，GBNClient.py的所有代码逻辑分析完毕。实际上，GBNServer的代码几乎和此相同，唯一的不同点在：所有log打印中，以server开头；在start函数中，进入无限循环，以体现其服务器的不休眠性质。但是，服务器只会在第一次连接发送数据给客户端，来避免资源的浪费。

**（二）、在GBN基础上改进并实现SR协议。**

1.实验原理介绍

对于SR而言，其可以说是GBN的优化版本。对于发送方来说，在消耗更多资源记录每个包的计时器的同时，可以做到只重传丢失的包；对于接收方来说，可以构建接收窗口来缓存一些（有条件的）非预期的未来数据包，而不是简单地丢弃。

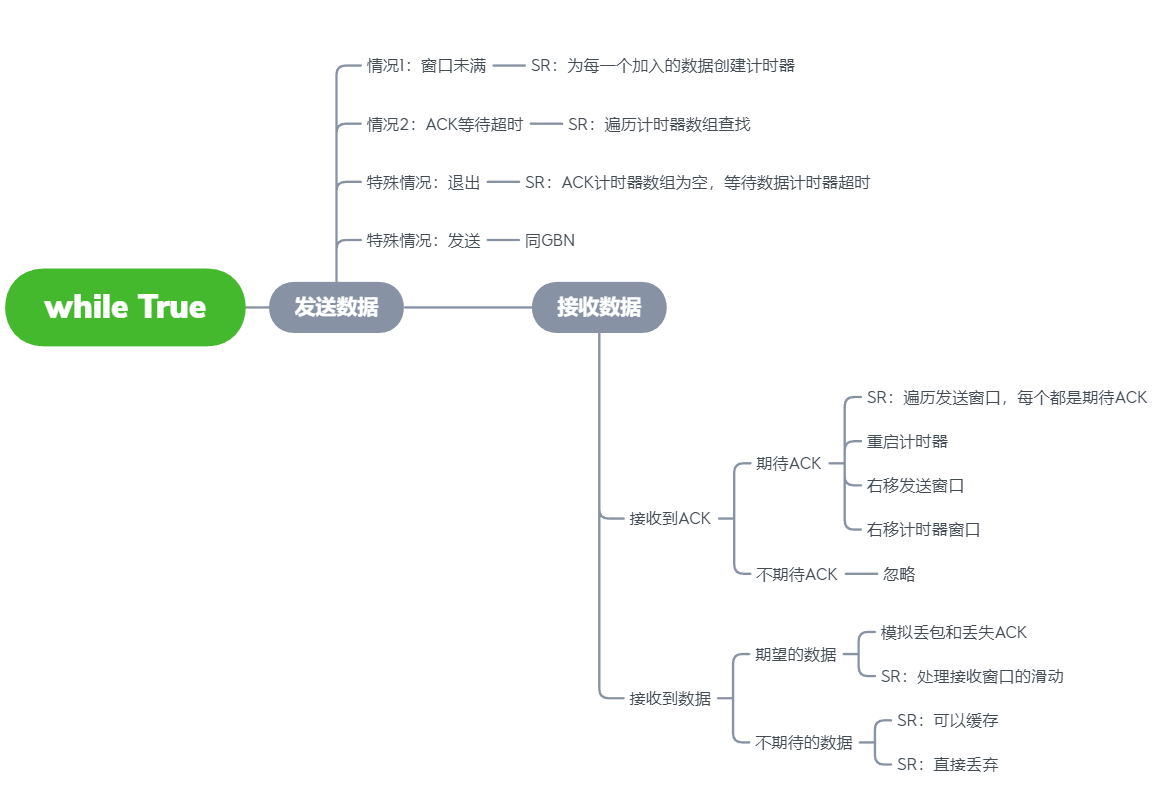
2.具体实现过程

这里使用SRServer.py为例（同理，SRClient几乎也是一样的），详细介绍SR的以GBN的python实现为基础的代码逻辑，在源代码中的注释也着重标注了GBN和SR的实现不同点。

1. \_\_init\_\_(self)初始化变量：

不同于BGN的变量有：M（接收窗口大小），receive\_window（接收窗口），同时之前的window改为send\_window,避免歧义。

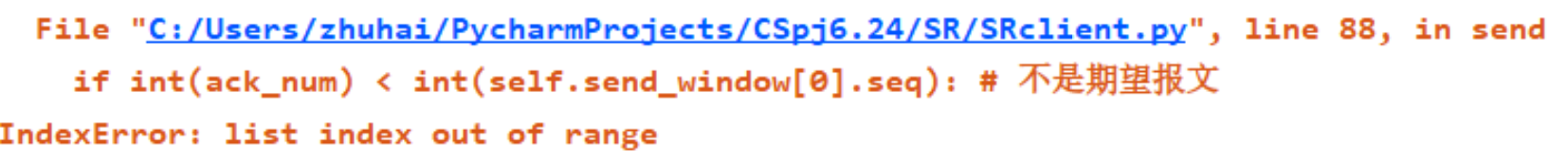
1. 主要实现函数send(self,buffer)：

其主要逻辑如下图，不同于GBN的步骤重点标出：

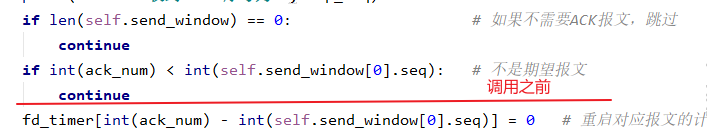
图三：SRServer中send函数的实现逻辑

对于一些重点逻辑的分析：

1. 在收到ACK报文时，我们不妨假设出现一些极端情况，例如重传报文后收到了姗姗来迟的原报文ACK，这时会导致window已经右移清空，但仍处理一个ACK报文，这时如果取访问send\_window[0].seq获取其发送窗口最左端的序列号，则会出现访问错误，因为此时为空，结果方法，在调用之前，判断send\_window是否为空。



图四：报错，list index out of range



图五：调用之前进行判断

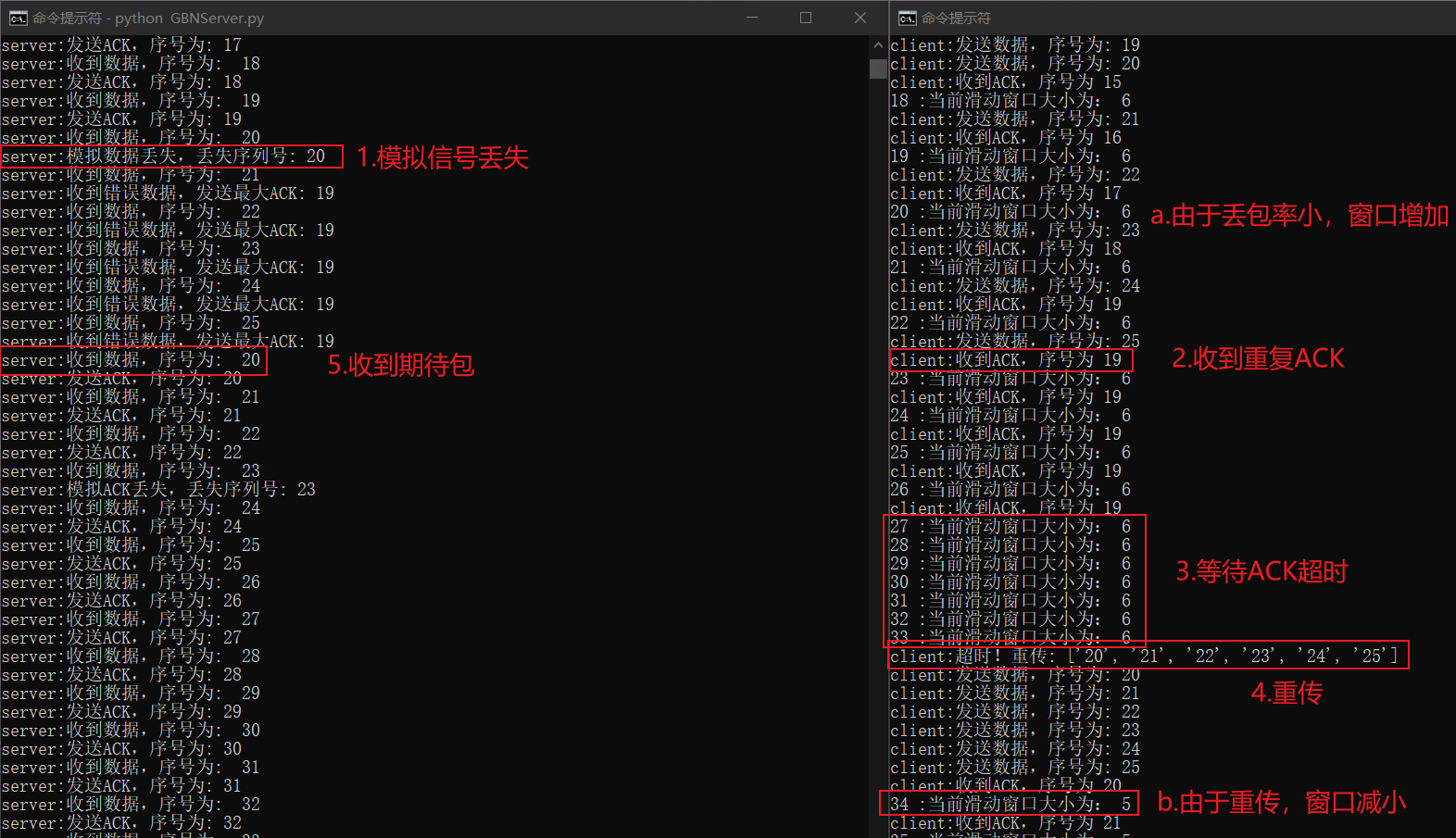
1. 对于fd\_timer计时器数组，其可以很方便地跟着send\_window一起移动，但是对于receive\_window，其移动实现比较困难，因为要缓存，判断条件比较复杂。于是使用另外一种方法：将收到的序列号插入receive\_window中，如果收到的包为last\_ack+1，那么收到了期望包，通过for循环判断后面的seq是否在receive\_window中，如果在，一并删除，以此抽象化接收窗口的右移。
2. 关于判断是缓存还是丢弃数据：if (last\_ack + 1 + self.M) > ack\_num > last\_ack and ack\_num not in self.receive\_window，即收到的seq在接收窗口之内且没有被缓存过，则缓存该数据；elif ack\_num <= last\_ack: 如果是已接收的包，考虑到防止对方的发送窗口无法移动，返回其希望的ACK；其他情况则丢弃该数据报。

3.实验总结

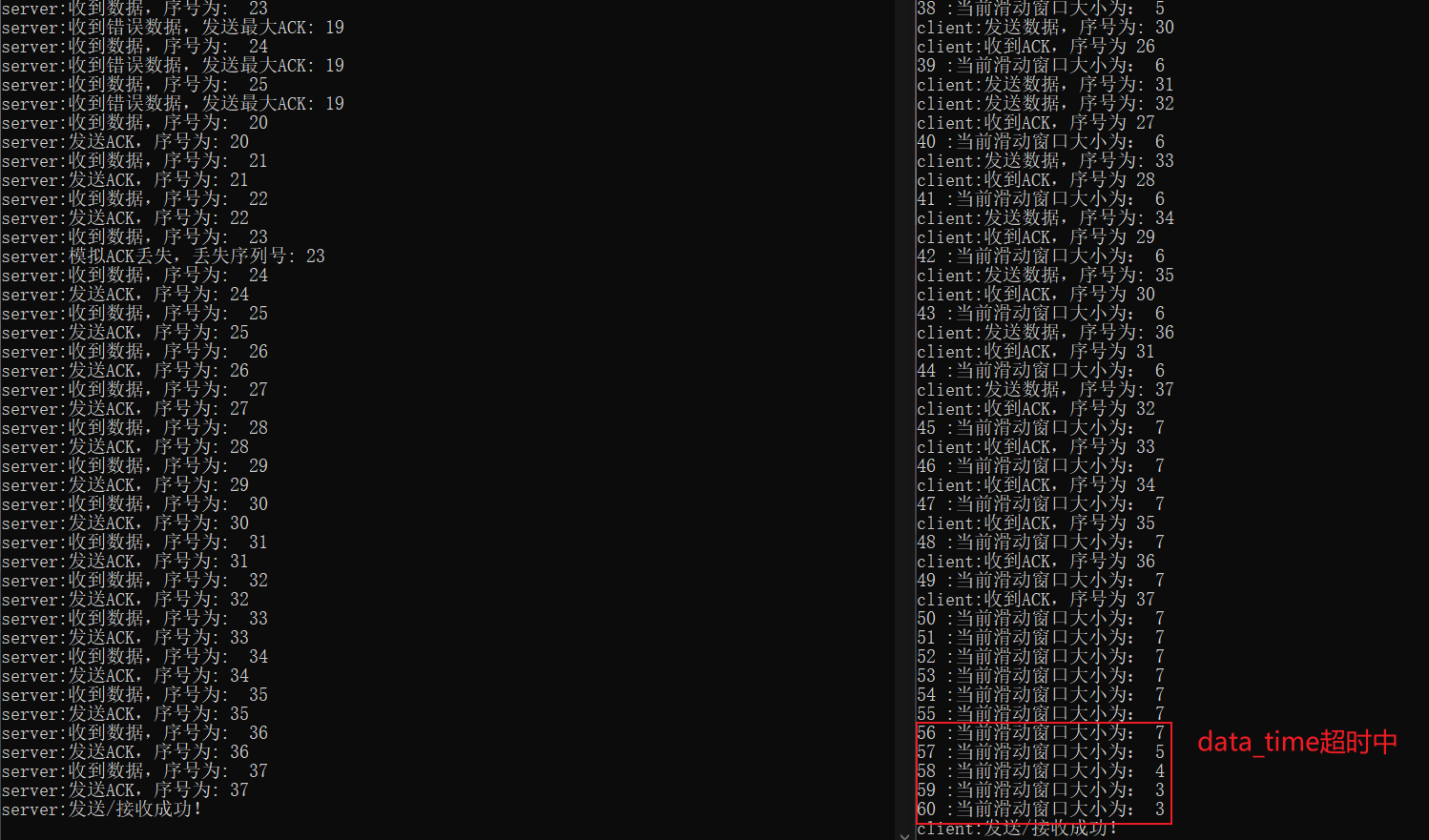
SR协议实现略显复杂，因为需要处理多个数组，包括发送窗口，接收窗口，计时器窗口。

1. **实验测试**
2. GNB测试

对于GBN，参数设置：buff大小100字节，初始滑动窗口3，丢包率为20%。测试结果如下：

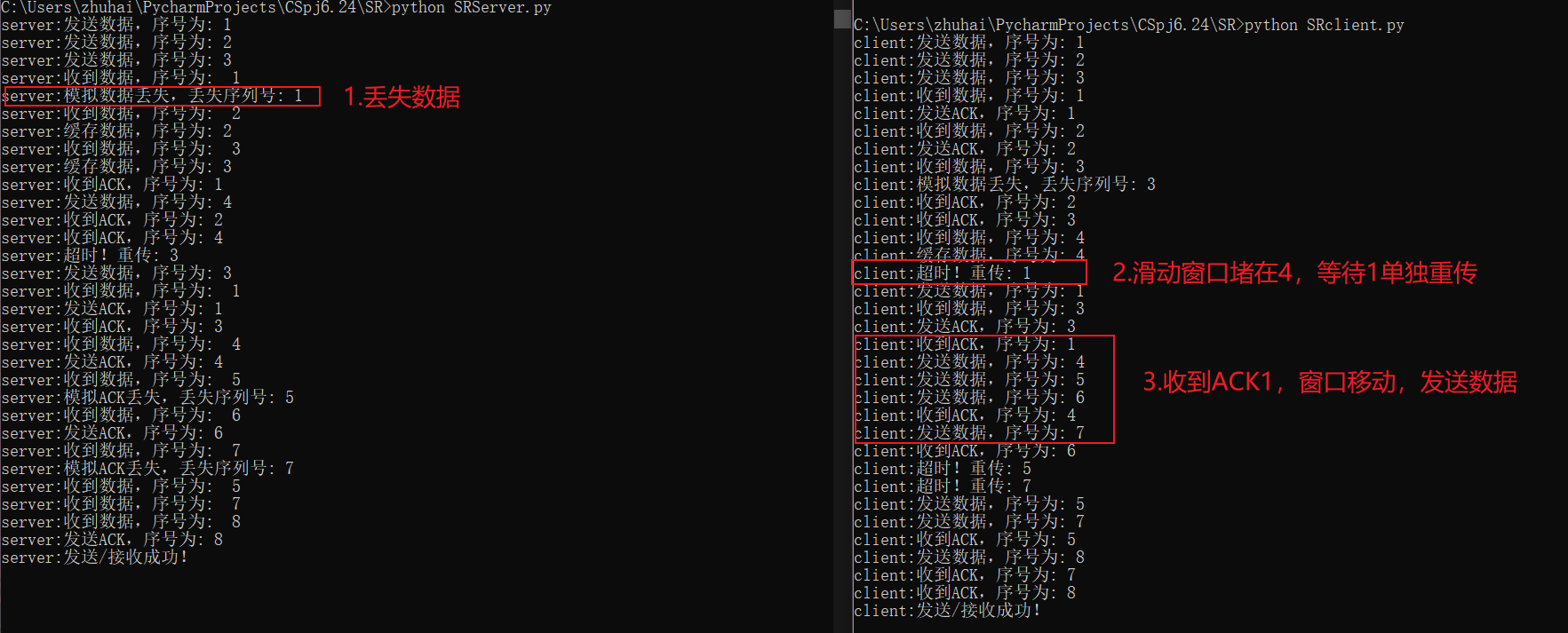


图六：GBN测试1，模拟丢包时



图七：GBN测试2，结束时

1. SR测试

对于SR，参数设置，buf为500字节，N,W均为3，丢包率为20%。

图九：SR测试

1. 实验结论

本次计算机网络安全期末设计完成了基于UDP的GBN和SR协议的python代码实现，将一个文件（例如csend.txt）通过socket套接字写入另一个文件（例如sreceive.txt）。实现前期有诸多不易，但是随着资料的查阅和对协议本身的理解加深，最后也成功实现了代码。

感谢以下参考博客：

[python re库 详解(正则表达式）\_孙天成-全栈工程师-CSDN博客](https://blog.csdn.net/stcaaa/article/details/83864210?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522162454657816780262565033%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=162454657816780262565033&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-4-83864210.first_rank_v2_pc_rank_v29_2&utm_term=python+re&spm=1018.2226.3001.4187)

[python 套接字socket的send/sendto、recv/recvfrom的阻塞与非阻塞\_Dontla的博客-CSDN博客](https://blog.csdn.net/Dontla/article/details/103818948?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522162454114416780255267100%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request_id=162454114416780255267100&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_v2~rank_v29-1-103818948.first_rank_v2_pc_rank_v29_2&utm_term=python+socket+sendto+%E5%92%8C+send&spm=1018.2226.3001.4187)

[Python实现socket的非阻塞式编程 - 简书 (jianshu.com)](https://www.jianshu.com/p/95b1405dade4)

[python random()函数\_gxz987的博客-CSDN博客\_random()函数](https://blog.csdn.net/gxz987/article/details/93322581?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522162453613816780262587543%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=162453613816780262587543&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-2-93322581.first_rank_v2_pc_rank_v29_2&utm_term=random%28%29+python&spm=1018.2226.3001.4187)

[（亲测解决）PyCharm 导包提示 unresolved reference（完整图解）\_TianXinCoord的博客-CSDN博客](https://blog.csdn.net/sinat_34104446/article/details/80951611?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522162453465716780262561241%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=162453465716780262561241&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-80951611.first_rank_v2_pc_rank_v29_2&utm_term=Unresolved+reference+pycharm&spm=1018.2226.3001.4187)

1. select模块参考：[我的python学习笔记之select模块\_zhuangzi123456的博客-CSDN博客](https://blog.csdn.net/zhuangzi123456/article/details/84400108?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522162461775116780261934316%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=162461775116780261934316&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-84400108.first_rank_v2_pc_rank_v29_2&utm_term=python+select&spm=1018.2226.3001.4187) [↑](#footnote-ref-1)