
哈尔滨工业大学

<<计算机网络>>

实验报告

(2019 年度秋季学期)

姓 名	张景润
学 号	1172510217
学 院	计算机科学与技术学院
教 师	刘亚维
实验名称	GBN 协议的设计与实现

目录

一、	实验目的	1
二、	实验环境	1
三、	实验内容	1
四、	实验原理	1
1.	GBN(SR)实验的几点说明	1
2.	GBN与SR协议交互过程与两端流程图	2
3.	GBN协议的单向数据传输	5
4.	GBN(SR)模拟数据包的丢失	8
5.	GBN的双向数据传输(选做,加分项)	8
6.	SR协议实现(选做,加分项)	9
五、	实验过程	11
1.	GBN协议的单向数据传输	11
2.	GBN模拟数据包的丢失	13
3.	GBN协议的双向数据传输(选做,加分项)	13
4.	SR协议实现(选做,加分项)	14
六、	实验结果	15
1.	单向GBN无丢包的实验结果	15
2.	双向GBN存在丢包的实验结果	16
3.	双向SR存在丢包的实验结果	17
七、	实验心得	19
1.	知识收获	19
2.	代码体会	19

一、实验目的

1. 理解滑动窗口协议的基本原理；
2. 掌握 GBN 的工作原理；
3. 掌握基于 UDP 设计并实现一个 GBN 协议的过程与技术；
4. 附加掌握基于 UDP 设计并实现一个 SR 协议的过程与技术。

二、实验环境

1. 接入 Internet 的实验主机；
2. Windows xp 或 Windows 7/8/10；
3. 开发语言：C/C++/Java 等

三、实验内容

1. 基于 UDP 设计一个简单的 GBN 协议，实现单向可靠数据传输（服务器到客户的数据传输）；
2. 模拟引入数据包的丢失，验证所设计协议的有效性；
3. 改进所设计的 GBN 协议，支持双向数据传输（选作，加分项）；
4. 将所设计的 GBN 协议改进为 SR 协议（选作，加分项）。

四、实验原理

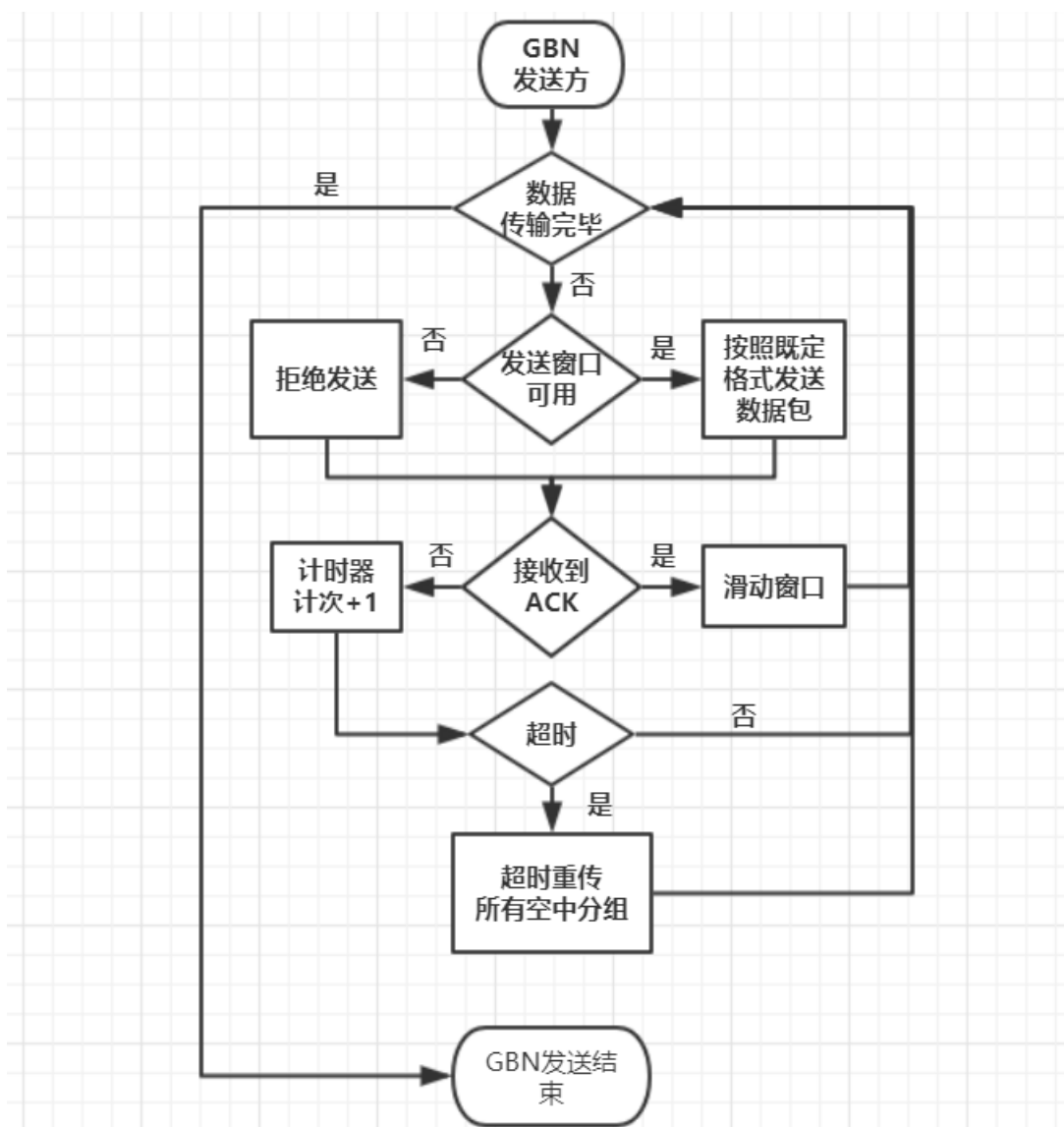
1. GBN(SR)实验的几点说明

三点说明
当本实验的 GBN 发送窗口尺寸设置为 1 时，代码和逻辑上即实现了停等协议，因此实验报告未对停等协议进行单独讨论
由于本实验是基于 UDP 实现的 GBN 和 SR 协议，因此不用进行差错检测与纠错，而是利用 UDP 协议进行差错检测
为了模拟 GBN 协议中来自上层协议的数据到达，实验发送方读取文件的数据流作为上层协议的数据到达，接收方将发送方传递的数据写入到新的文件中表示接收方成功将数据交付到上层协议中

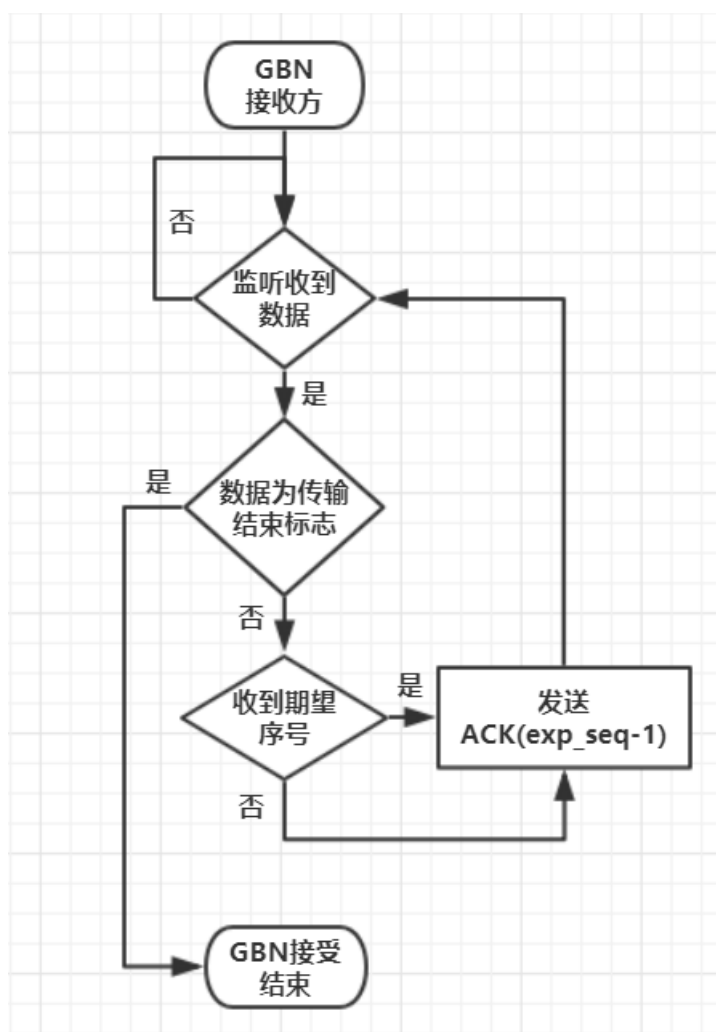
实验的几点说明

2. GBN 与 SR 协议交互过程与两端流程图

(1) **GBN 协议的流程图**：分别是发送方和接收方，如下图

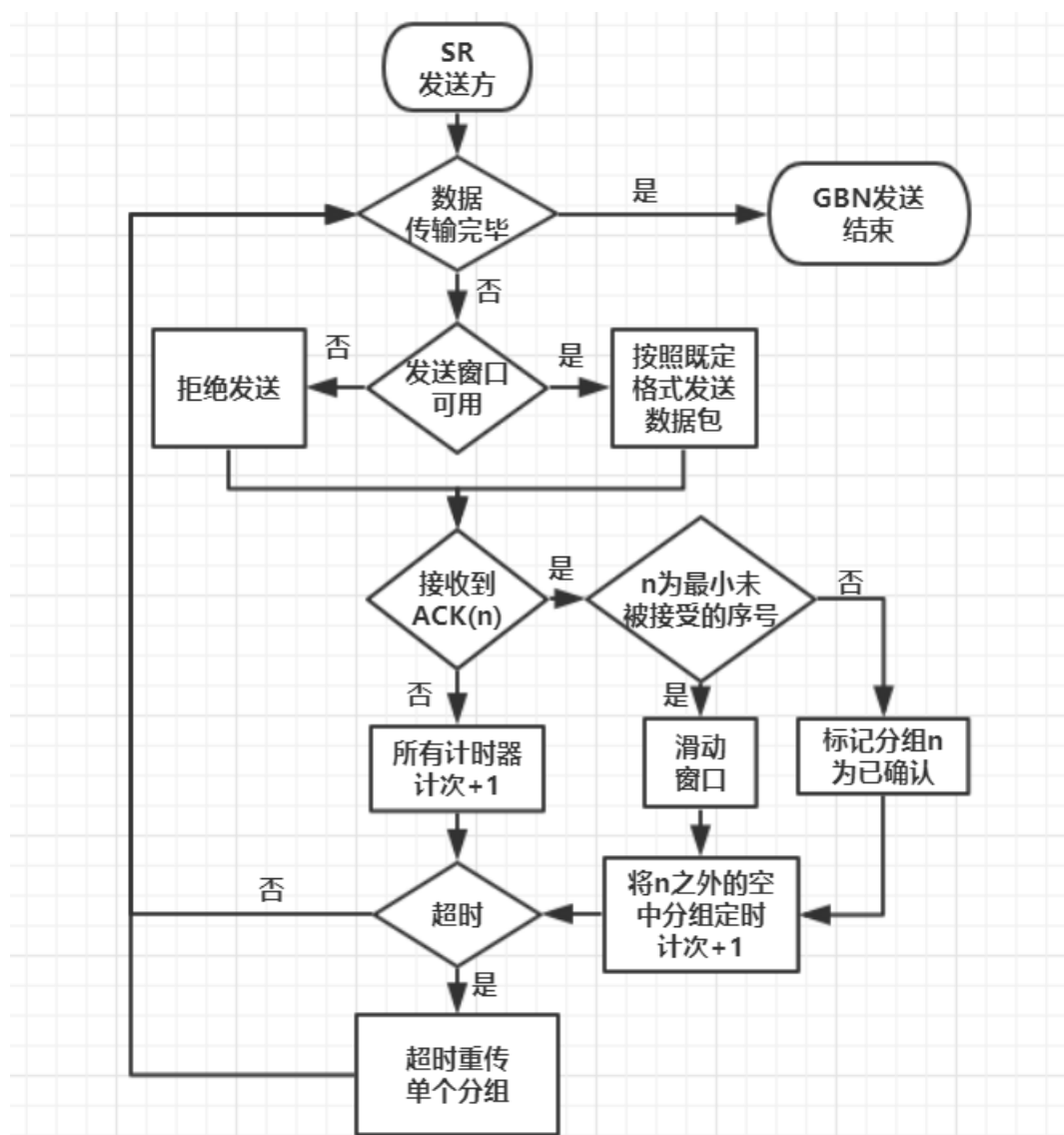


GBN 发送方流程图

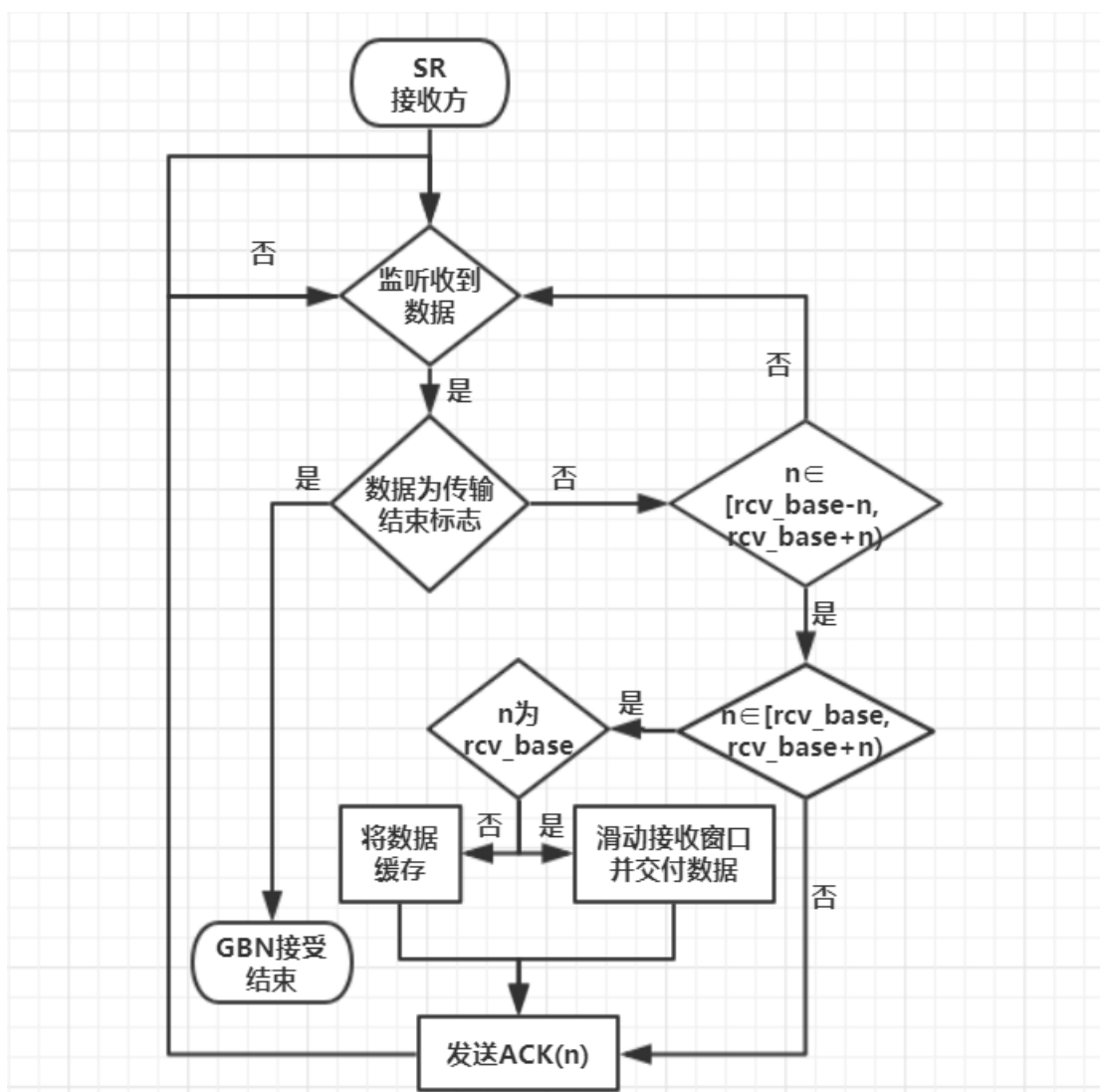


GBN 接收方流程图

(2) **SR 协议的流程图**：包括发送方流程图和接收方流程图，如下图所示。



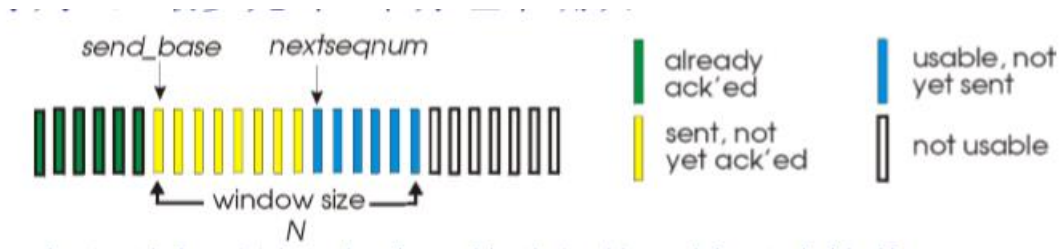
SR 发送方流程图



SR 接收方流程图

3. GBN 协议的单向数据传输

(1) **数据传输的流水线机制**：与停等协议不同的是，GBN 的发送方在收到 ACK 之前可以连续发送多个分组，这则使得 GBN 需要更大的存储空间以缓存分组。GBN 发送方使用一个序列号标识分组，窗口尺寸为 N ，表示最多允许 N 个分组未确认；接收方发送发送 ACK(n) 确认报文，用于表示收到该序号 n 及其之前的所有报文。同时 GBN 发送方为数据传输维护了一个定时器，用于超时重传，即发生超时，重传窗口内的所有报文。GBN 发送方窗口如图。



GBN 发送方窗口

(2) **数据传输格式规约**：为了实现数据传输，GBN 协议中对数据报的格式进行了规约。发送数据报格式为：**序号 数据**；接收方返回的 ACK 格式为：**序号 0**；数据发送结束的标识报文格式为：**0 0**。代码中格式规约如截图。

```
# 规定发送数据格式: [seq_num data]
# 规定发送确认格式: [exp_num-1 0]
# 规定发送结束格式: [0 0]
```

数据传输包格式规约

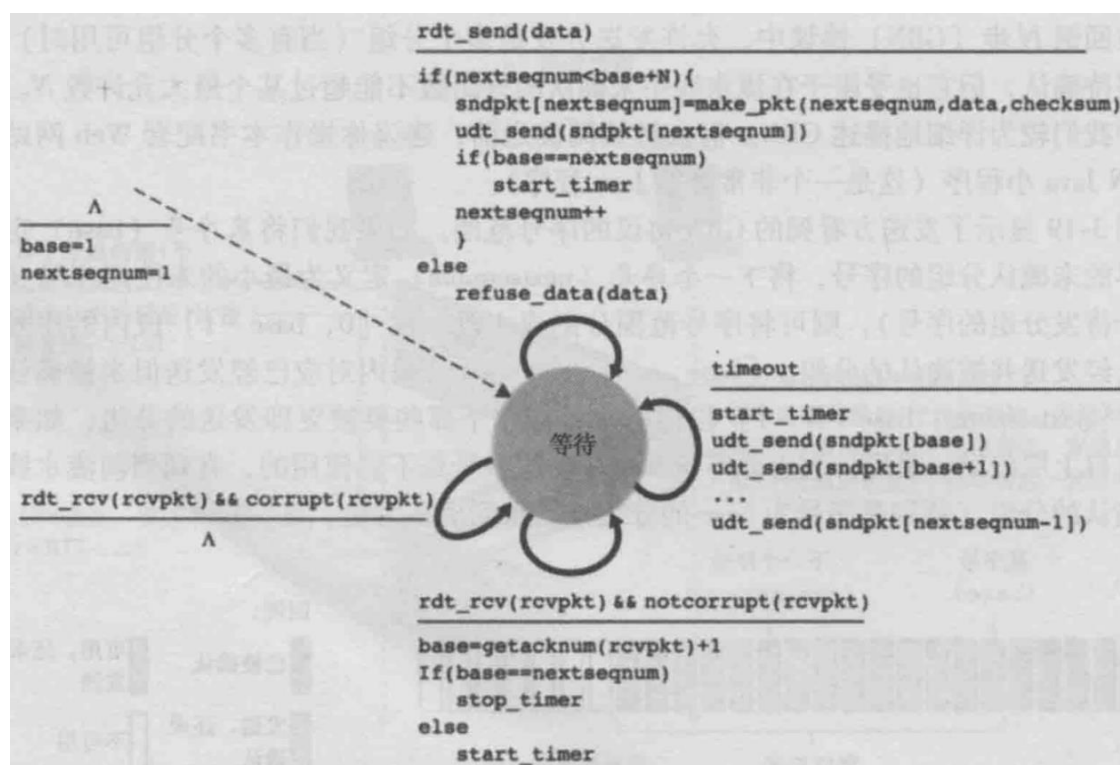
(3) **发送方维护的数据结构**：发送方维护一个发送窗口，用于流水线传输，具体的数据结构如下表所示。

数据结构	用处
send_window_size	窗口大小，表示一次可以流水线形式发送的数据数目
send_base	最小的期望收到的确认报文的序号值
next_seq	下一个可用于发送数据的序号值
time_count	记录当前未收到接收 ACK 的次数，当达到阈值时，会触发超时重传处理

发送方维护的数据结构

(4) **发送方的处理逻辑**：响应三种类型的事件，如下表。处理的 FSM 如下图。

类型	处理
上层调用	上层调用 rdt_send(), 若窗口未满, 则产生并发送一个分组并更新状态; 否则将数据返回给上层协议, 告知其窗口已满
收到 ACK(n)	GBN 协议采用累积确认方式, 即收到 ACK(n)表明接收方已经收到序号 n 及之前的所有分组。收到 ACK 后将窗口滑动到 n+1。若仍有未被确认的分组, 重启定时器; 否则终止定时器
定时器超时	发送方重传所有已发送但未被确认的分组。发送方仅使用一个定时器, 可被当做最早的已发送但未被确认的分组所使用的定时器。



发送方 FSM

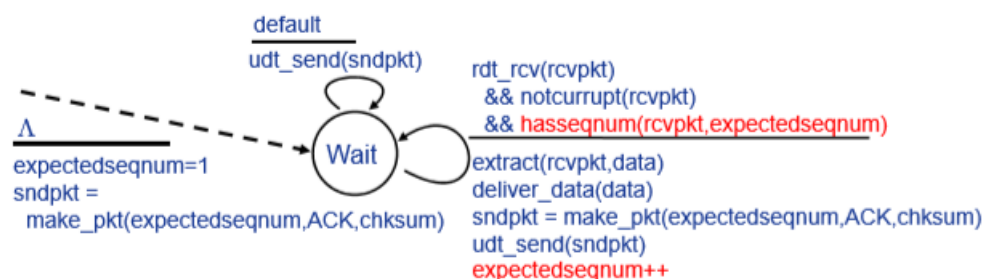
(5) **接收方维护的数据结构**: 接收方不维护接收窗口, 或者说接收方维护一个尺寸为 1 的接受窗口, 数据结构如下表。

数据结构	用处
$data_buf_size$	用于表示接收数据时的缓存大小
exp_seq	当前期望收到数据的序号

接收方维护的数据结构

(6) **接收方的处理逻辑**: 接收方响应数据到来的事件, 接收方 FSM 如下图。

类型	处理
收到失序数据	发送 ACK(n), n 为期待收到的数据序号-1, 并丢弃数据
收到按序数据	发送 ACK(n), n 为期待收到的数据序号-1, 并向上层交付数据



接收方 FSM

4. GBN(SR) 模拟数据包的丢失

(1) **概率性传输数据包**：实验状态下，如果不做相应的处理，数据包一般是不会丢失的。因此实验采用概率性发送数据包和概率性发送 ACK 确认包的方法，从而实现模拟丢包的功能。

(2) **发送方概率性传输**：发送方在处理需要相应的三种事件时，有两种情况需要向接收方发送数据：接收到上层数据和超时重传。当接收到上层数据的时候发送方在有剩余窗口的情况下，会发送数据包。当超时的时候，发送方会重传窗口内的所有数据包。在发送的时候，应该通过 python 的 random 函数的 random 方法产生一个(0,1)区间内的随机数，若产生数字大于数据包丢包率，则发送数据；否则，不发送数据；同样地，在超时重传的时候，调用相同的逻辑，发送或忽略数据的传输。

情况	处理
收到上数据	调用 python 的 random 函数产生伪随机数，随机发送数据包
超时重传处理	调用 python 的 random 函数产生伪随机数，随机发送数据包

发送方概率性传输

(3) **接收方概率性传输**：接收方在收到发送方的传输数据时，调用 python 的 random 函数产生伪随机数，随机发送 ACK 确认包。

(4) **丢包分析**：当产生发送数据包 n 丢失时，接收方不会收到相应的数据包，因此不会发送 ACK(n)，若新的数据包 n+1 到来，接收方会发送 ack(n-1)，发送方可以根据确认包重传传输数据包 n；当接收方确认包 n 丢失时，发送方会因为数据包的确认包超时而产生超时重传。因此通过超时重传、累积确认机制可以实现数据包的端到端的逻辑不丢失。

5. GBN 的双向数据传输（选做，加分项）

(1) **双向数据传输思想**：我这里理解的双向传输是指：发送方同时可以作为接收方接收另外一个发送方的数据包进行缓存并交付到上层协议。因此我们要将单向传输 GBN 协议的发送方和接收方功能组合在一起。

(2) **思想实现**：原发送方在原来的基础上要维护更多的数据结构，如下表。同时发送方要添加更多的功能：将数据交付到上层协议（实验中的实现为将数据写入到文本文件中），不断接收发送方的数据包并回复 ACK 确认报文（也要实现概率性丢包功能）

数据结构	用处
data_buf_size	用于表示接收数据时的缓存大小
exp_seq	当前期望收到数据的序号

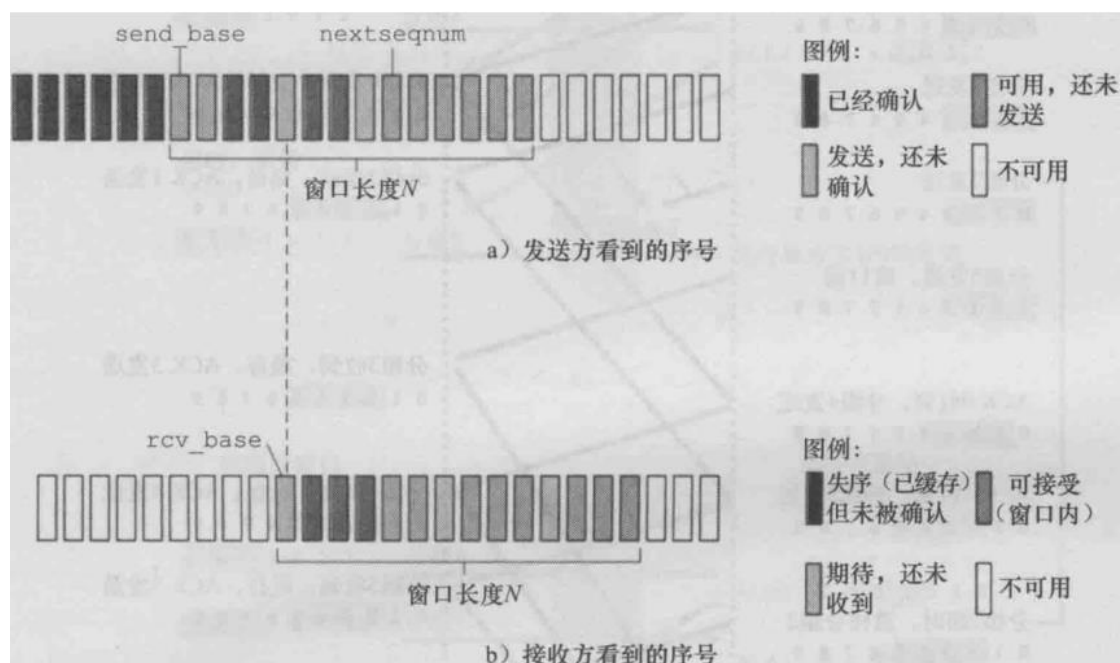
发送方要多维护的数据结构

功能	用处
<code>write_data_to_file</code>	将收到数据写到文件中，表示将数据包交付到上层协议
<code>reply_ack</code>	回复 ACK 确认报文，用于实现数据的确认

发送方要多实现的功能

6. SR 协议实现（选做，加分项）

(1) **SR 协议的思想**：不同于 GBN 协议的累计确认机制，SR 通过在接收方维护一个滑动窗口来实现单独确认每个数据包的方法，同时使用 `rcv_base` 记录仍未收到的最小序号的分组。SR 发送方为每个分组维护一个定时器，只重传那些超时时间内未收到 ACK 的分组，而接收方不会丢失失序的未被确认的分组数据，通过维护一个接收方数据缓存将这些分组暂时保存，等到数据已经按序排好后，统一交付给上层协议。



SR 协议的滑动窗口

(2) **数据传输格式规约**：为实现数据传输，GBN 协议中对数据报的格式进行了规约。发送数据报格式为：序号 数据；接收方返回的 ACK 格式为：序号 0；数据发送结束的标识报文格式为：0 0。代码中格式规约如截图。

```
# 规定发送数据格式: [seq_num data]
# 规定发送确认格式: [exp_num-1 0]
# 规定发送结束格式: [0 0]
```

数据传输包格式规约

(3) **SR 维护的数据结构**：由于 SR 同样实现了双向数据传输，因此不再区分发送方和接收方。SR 维护一个发送窗口和一个接收窗口，同时为每个

空中的发送数据分组维护一个定时器，也维护一个接收数据缓存用于缓存失序到达的数据分组。

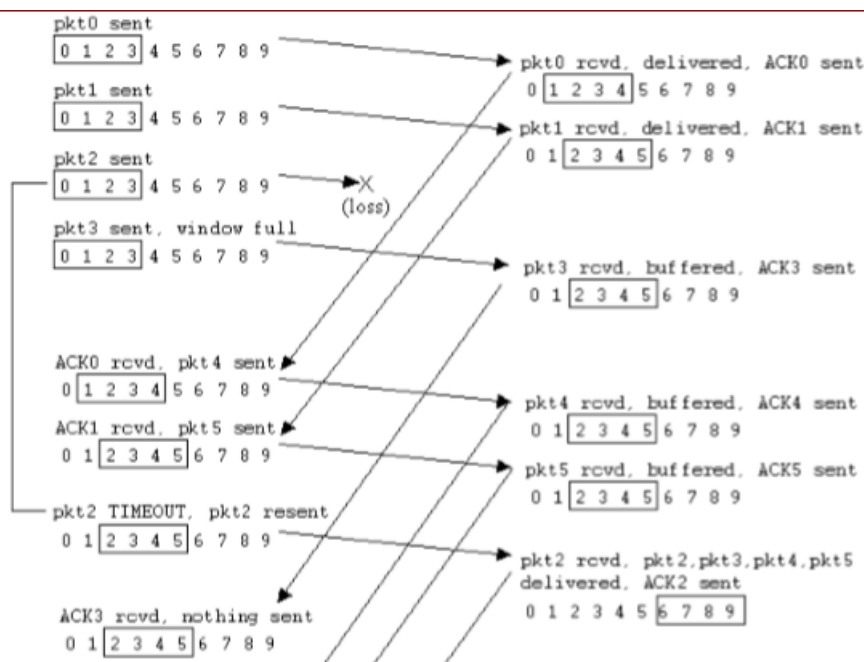
数据结构	用处
<code>send_window_size</code>	发送窗口大小，表示一次流水线形式发送的数据数目
<code>rcv_window_size</code>	接收窗口大小，表示接受的所有失序的数据
<code>send_base</code>	最小的期望收到的确认 ACK 包的序号值
<code>next_seq</code>	下一个可用于发送的数据的序号值
<code>rcv_base</code>	表示希望收到的最小的数据包的序号
<code>time_counts</code>	记录数据包在传输中未收到 ACK 的次数，当达到阈值时，会触发单个数据包超时重传处理
<code>rcv_data</code>	保存接收到的失序的数据包

SR 维护的数据结构

(4) **SR 的处理逻辑**：响应三种类型的事件，如下表。处理的 FSM 如下图。

类型	处理
上层调用	上层调用 <code>rdt_send()</code> ，若窗口未满，则产生并发送一个分组并更新状态；否则将数据返回给上层协议，告知其窗口已满
收到 ACK(n)	若 n 在窗口内则将那个该分组标记为已接受。若 <code>n=send_base</code> ，则窗口基序号移动到具有最小序号的未确认分组处
定时器超时	为每个空中分组维护一个定时器，超时重传对应的一个分组
收到 pkt(n)	若 $n \in (\text{rcv_base}, \text{rcv_base} + N)$ ，则收到的分组落在接收方窗口内，发送 ACK(n)，若该分组未被收到过，则缓存；若 $n \in [\text{rcv_base} - N, \text{rcv_base})$ ，则产生一个 ACK(rcv_base)；其他的序号则忽略
收到 pkt(rcv_base)	滑动接收窗口到最小的未被接收的数据序号处，并将这些按序的数据交付给上层协议

SR 的处理逻辑



sr 协议示例

五、 实验过程

实验代码的几点说明	
侧重于代码	由于实现具体的实现思路已经在本报告的第四章节详细叙述，在第五章节中注重代码的实现与具体的函数
主函数	main 类可选择运行 GBN 或者 SR
配置文件	实验可以通过配置文件配置主机信息

1. GBN 协议的单向数据传输

(1) **代码逻辑概述**: 代码逻辑包括: 获取数据逻辑、发送数据逻辑、接收 ACK 逻辑、超时重传逻辑、发送 ACK 逻辑、交付数据逻辑。GBN 的通信双方不断循环或发送数据直到发送完毕或接收数据直到接收完毕, 并做出对应的响应;

(2) **获取与交付数据逻辑**: 实验通过读取文本文件的数据来模拟上层数据的到来。并将读取到的数据保存到数据结构 data 中, 当 data 中数据全部被发送时, 则表示传输结束; 接收方将按序数据不断写入到本地文本中作为交付给上层协议, 直到收到结束数据包。

(3) **发送数据逻辑 (发送方)**: 如果当前发送窗口中仍剩余序号空间且数据 data 未被完全发送, 则发送 data 中下一部分的数据; 否则返回;


```

def send_data(self):
    if self.next_seq == len(self.data): # data数据已全部被发送过
        print('服务器:发送完毕, 等待确认')
        return
    diff = self.next_seq - self.send_base if self.next_seq >= self.send_base \
        else self.next_seq - self.send_base + Host.seq_length # 表示占用的窗口大小
    if diff < 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(Host.make_pkt(self.next_seq, self.data[self.next_seq]),
                               self.remote_address)
        if self.send_base == self.next_seq: # 若窗口中无分组, 则计时器重置
            self.time_count = 0
        print('服务器:成功发送数据' + str(self.next_seq))
        self.next_seq = (self.next_seq + 1) % Host.seq_length
    else: # 窗口中无可用空间
        print('服务器:窗口已满, 暂不发送数据')

```

(4) **接收 ACK 逻辑（发送方）**：发送方非阻塞地接收 ACK 包，我采用的是 select.select 方法。若接收到 ACK(n)，则滑动窗口至 n+1；否则超时计次+1；

(5) **超时处理逻辑（发送方）**：当计次+1 之后，开始超时判断，若次数超过阈值则执行超时处理，则重传所有的空中分组；否则，继续调用数据发送逻辑循环处理；

```

if len(readable) > 0: # 接收ACK数据
    rcv_ack = self.socket.recvfrom(self.ack_buf_size)[0].decode().split()[0]
    print('收到客户端ACK:' + rcv_ack)
    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: # 出发超时重传操作
        self.handle_time_out()

```

(6) **发送 ACK 逻辑（接收方）**：接收方接收到数据包时，若非结束包且为期待包，则发送确认包 ACK(exp_seq+1)并将数据写入到本地文件中，否则发送确认包 ACK(exp_seq)。

```

if rcv_seq == '0' and rcv_data == '0': # 接收到结束包
    print('客户端:传输数据结束')
    break
if int(rcv_seq) == self.exp_seq:
    print('客户端:收到期望序号数据:' + str(rcv_seq))
    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
else:
    print('客户端:收到非期望数据, 期望:' + str(self.exp_seq) + '实际:' + str(rcv_seq))

```

2. GBN 模拟数据包的丢失

(1) **随机数模拟丢包**：发送方和接收方在每一次发送数据包或者确认包时，都要根据 python 的 random 方法先产生一个(0,1)区间的随机数，若该数字不小于丢包率，则发送数据包，否则不发送；

(2) **多了一个属性**：在原来类属性的基础上，增加了两个属性，分别是 pkt_loss 和 ack_loss，用于表示数据包和确认包丢包率；

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

(3) **具体实现**：发送方在数据发送逻辑和超时重传逻辑中要进行相应的随机数产生判断。接收方在产生确认包时，也要进行相应的判断。

```
if random.random() >= self.ack_loss:
    self.socket.sendto(Host.make_pkt(self.exp_seq - 1, 0), self.remote_address)
```

3. GBN 协议的双向数据传输（选做，加分项）

(1) **代码实现分析**：我理解的双向传输指的是 GBN 的发送方同时可以作为接收方发送 ACK 报文，而接收方也可以发送数据包。因此需要在代码中将发送方和接收方的功能组合起来，因此只需要一个类名为 GBN 用于实现双向传输功能。

(2) **GBN 类维护的数据结构**：GBN 类需要综合之前的发送方和接受方的数据结构，具体如下表所示。

数据结构	用处
send_window_size	窗口大小，表示一次可以流水线形式发送的数据数目
send_base	最小的期望收到的确认报文的序号值
next_seq	下一个可用于发送数据的序号值
time_count	记录当前未收到 ACK 的次数，达到阈值会触发重传处理
data_buf_size	用于表示接收数据时的缓存大小
exp_seq	当前期望收到数据的序号

GBN 维护的数据结构

(3) **GBN 类实现的函数方法**：GBN 类需要将发送方和接受方的方法进行统一处理，具体函数方法如下表所示。

函数	用处
send_data	发送数据逻辑，在存在剩余窗口的情况下发送数据包
handle_time_out	超时重传逻辑，当超时，重传所有空中分组

get_data_from_file	用于从文本中读取数据，模拟上层协议数据的到来
server_run	发送端主要函数不断处理 3 个逻辑：发送、接收、超时
write_data_to_file	将数据写入到文本中，模拟将数据交付给上层协议
client_run	接收端主要函数不断处理数据包并返回 ACK、交付数据

GBN 的实现函数

4. SR 协议实现（选做，加分项）

(1) 代码实现分析：实现了双向数据传输 SR 协议，在类 SR 中实现了发送和接收功能。SR 协议在 GBN 协议的基础上，维护了更多的数据结构。

(2) SR 类维护的数据结构：SR 类需要综合之前的发送方和接受方的数据结构，具体如下表所示。

数据结构	用处
send_window_size	窗口大小，表示一次可以流水线形式发送的数据数目
send_base	最小的期望收到的确认报文的序号值
next_seq	下一个可用于发送数据的序号值
time_count	记录当前未收到 ACK 的次数，达到阈值会触发重传处理
data_buf_size	用于表示接收数据时的缓存大小
rcv_base	表示可以收到的数据包的最小值
rcv_window_size	接收方窗口大小
time_counts	用于记录每个空中分组的超时计次，会单独触发重传
ack_seqs	用于记录发送方接受 ACK 的情况
rcv_data	用户缓存失序的数据

双向 SR 维护的数据结构

(1) **GBN 类实现的函数方法**：GBN 类需要将发送方和接受方的方法进行统一处理，具体函数方法以及两个滑动窗口处理函数如下表和下图所示。

函数	用处
send_data	发送数据逻辑，在存在剩余窗口的情况下发送数据包
handle_time_out	超时重传逻辑，当超时，重传引起超时的单个分组
get_data_from_file	用于从文本中读取数据，模拟上层协议数据的到来
slide_send_window	当收到窗口的最小序号的 ack 时滑动发送窗口
server_run	发送端主要函数不断处理 3 个逻辑：发送、接收、超时
write_data_to_file	将数据写入到文本中，模拟将数据交付给上层协议
client_run	接收端主要函数不断处理数据包返回 ACK、交付数据
slide_rcv_window	当收到期望的数据包时，滑动接收窗口到出现未接受包

双向 SR 的实现函数

```
# 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))
```

发送方滑动窗口

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

接收方滑动窗口

六、实验结果

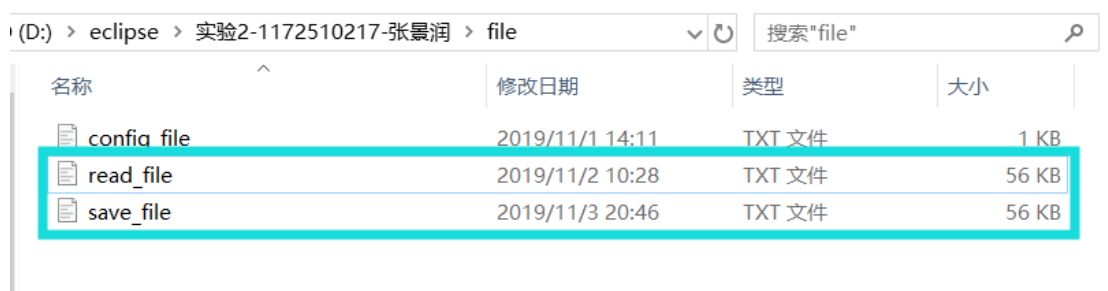
验证方式说明

验证终端输出：通过终端打印的发送端与接收端打印显示的序号信息可以判断实验代码运行结果的正确性；

验证接收到的文件：通过接收到的数据存入的文本文件与源文件对比可以得知实验代码运行结果的正确性。我的输入文件路径为`../file/read_file.txt`我的输入文件路径为`../file/save_file.txt`

1. 单向 GBN 无丢包的实验结果

(1) 一共需要传输 35 个大的数据包，文件大小为 56KB，如截图。



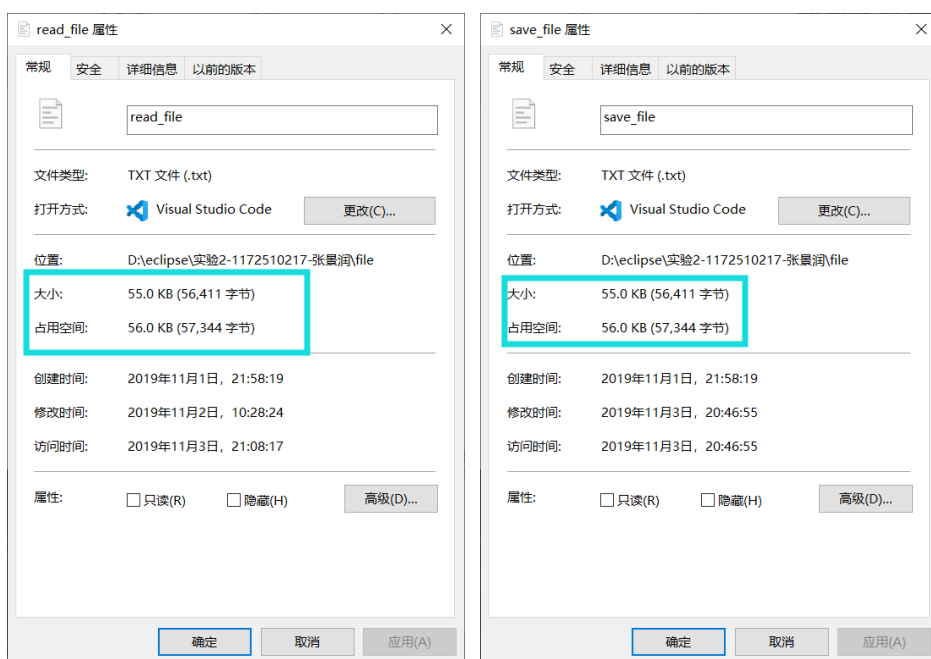
名称	修改日期	类型	大小
config_file	2019/11/1 14:11	TXT 文件	1 KB
read_file	2019/11/2 10:28	TXT 文件	56 KB
save_file	2019/11/3 20:46	TXT 文件	56 KB

(2) **终端截图：**发现每发送一个数据包，就接受一个 ACK。

```
服务器:成功发送数据32
客户端:收到期望序号数据:32
收到客户端ACK:32
服务器:成功发送数据33
客户端:收到期望序号数据:33
收到客户端ACK:33
服务器:成功发送数据34
客户端:收到期望序号数据:34
收到客户端ACK:34
服务器:成功发送数据35
客户端:收到期望序号数据:35
收到客户端ACK:35
服务器数据传输结束
客户端:传输数据结束
```

Process finished with exit code 0

(3) **文本文件对比**: 接收文本文件和发送文本文件完全一样, 可以直接通过打开对比得到 (简单的看, 两个文本大小完全一样)



2. 双向 GBN 存在丢包的实验结果

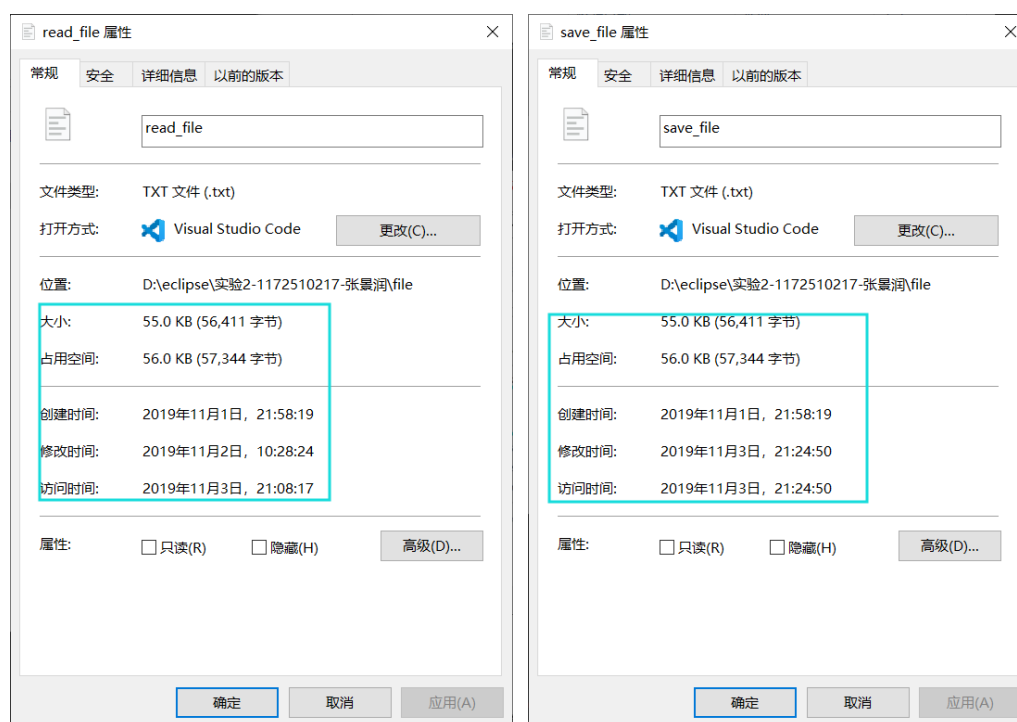
(1) 一共需要传输 35 个大的数据包, 文件大小为 56KB, 如截图。

(D:) > eclipse > 实验2-1172510217-张景润 > file					搜索"file"	
名称	修改日期	类型	大小			
confia file	2019/11/1 14:11	TXT 文件	1 KB			
read_file	2019/11/2 10:28	TXT 文件	56 KB			
save_file	2019/11/3 20:46	TXT 文件	56 KB			

(2) **终端截图**：发现发送成功一个数据包，不一定随即接收到一个对应的ACK，但是最终传输成功。

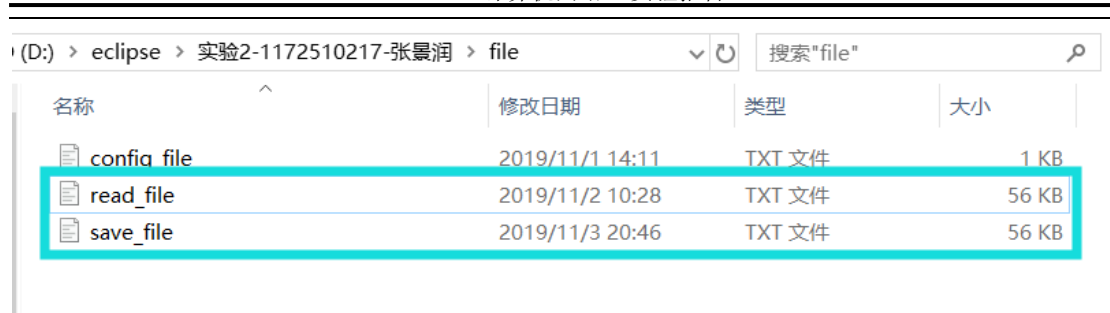
```
服务器:成功发送数据34
客户端:收到期望序号数据:32
收到客户端ACK:31
服务器:成功发送数据35
客户端:收到期望序号数据:33
收到客户端ACK:32
服务器:发送完毕,等待确认
客户端:收到期望序号数据:34
收到客户端ACK:33
服务器:发送完毕,等待确认
收到客户端ACK:34
服务器:发送完毕,等待确认
客户端:收到期望序号数据:35
收到客户端ACK:35
服务器:发送完毕
客户端:传输数据结束
```

(3) **文本文件对比**：接收文本文件和发送文本文件完全一样，可以直接通过打开对比得到（简单的看，两个文本大小完全一样）



3. 双向 SR 存在丢包的实验结果

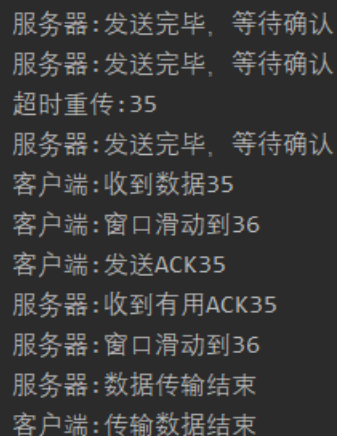
(1) 一共需要传输 35 个大的数据包，文件大小为 56KB，如截图。



The screenshot shows a Windows File Explorer window with the address bar set to (D:) > eclipse > 实验2-1172510217-张景润 > file. The search bar contains 'file'. The file list contains three items: 'config_file' (1 KB, TXT file, modified 2019/11/1 14:11), 'read_file' (56 KB, TXT file, modified 2019/11/2 10:28), and 'save_file' (56 KB, TXT file, modified 2019/11/3 20:46). The 'read_file' and 'save_file' rows are highlighted with a red border.

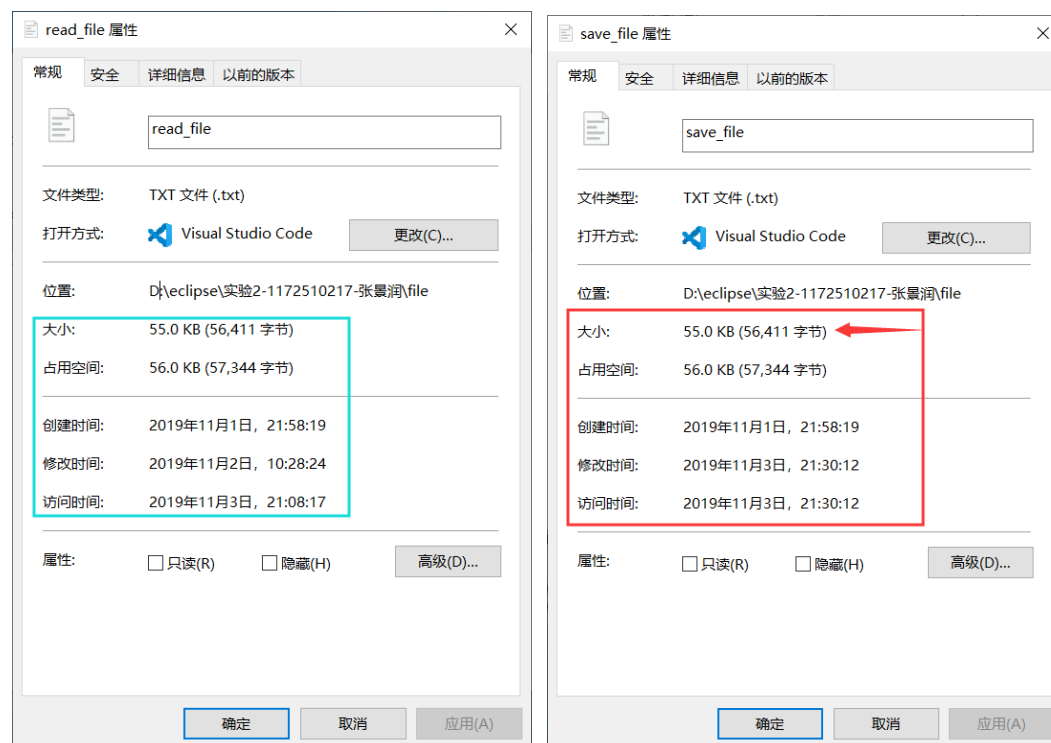
名称	修改日期	类型	大小
config_file	2019/11/1 14:11	TXT 文件	1 KB
read_file	2019/11/2 10:28	TXT 文件	56 KB
save_file	2019/11/3 20:46	TXT 文件	56 KB

(3) **终端截图**: 发现发送成功一个数据包, 不一定随即接收到一个对应的ACK, 但是最终传输成功。



The terminal window displays the following logs:
服务器: 发送完毕, 等待确认
服务器: 发送完毕, 等待确认
超时重传: 35
服务器: 发送完毕, 等待确认
客户端: 收到数据35
客户端: 窗口滑动到36
客户端: 发送ACK35
服务器: 收到有用ACK35
服务器: 窗口滑动到36
服务器: 数据传输结束
客户端: 传输数据结束

(4) **文本文件对比**: 接收文本文件和发送文本文件完全一样, 可以直接通过打开对比得到 (简单的看, 两个文本大小完全一样)



七、 实验心得

1. 知识收获

(1) 收获了滑动窗口协议的基本原理；并通过具体的实践感受到了滑动窗口协议在 GBN 与 SR 中的体现；

(2) 掌握了基于 UDP 设计并实现一个 GBN 协议的过程与技术，同时也体会到了如何实现模拟上层数据到来以及模拟丢包事件的产生；

(3) 附加掌握了基于 UDP 设计并实现一个 SR 协议的过程与技术。

2. 代码体会

(1) 通过编程训练，掌握了 MOOC 中学习不到的知识，比如：GBN 和 SR 具体的的工作原理及其实现；

(2) 同时也掌握了重点知识的细节内容，比如：掌握了 GBN 的累计确认机制，掌握了 ACK(n)表示 n 及 n 之前的所有数据已经确认收到，也掌握了发送数据包和确认包的规约格式实现；

(3) 通过 python 实现（实验 1 用 java 实现）发现了与 java 不同的套接字通信的特点，比如 python 关于套接字方法更接近于 C 实现，而 java 则使用流处理来进行相关的操作。