GO-BACK-N(GBN) Protocol
实验报告

学　　院　计算机与信息技术学院

专　　业　计算机科学与技术

年级班别　2018 级

组长　田　震（学号 18301017 ）

组员　王子龙（学号 18281218 ）

组员　周天宸（学号 18301121 ）

组员　武斯全（学号 18231420 ）

组员　万奕晨（学号 18291020 ）

成　　绩

实验题目 GO-BACK-N(GBN) Protocol

一、 实验目的

理解 GBN（GO-BACK-N）协议，并通过编程实现 GBN 协议，对可靠数据传输原理有进一步的理解和掌握。

二、 实验环境

操作系统： macOS 11.0.1 Big Sur

编程语言： Java (Java HotSpot(TM) 64-Bit Server VM (build 13.0.1+9, mixed mode, sharing))

集成环境： IntelliJ IDEA 2020.2 (Ultimate Edition)

三、 实验内容和要求

GBN 实现的伪代码：

```
N  := window size
Rn := request number
Sn := sequence number
Sb := sequence base
Sm := sequence max


function receiver is
    Rn := 0
    Do the following forever:
        if the packet received = Rn and the packet is
error free then
            Accept the packet and send it to a higher
layer
            Rn := Rn + 1
        else
            Refuse packet
        Send a Request for Rn


function sender is
    Sb := 0
    Sm := N + 1
    Repeat the following steps forever:
        if you receive a request number where Rn > Sb then
            Sm := (Sm - Sb) + Rn
            Sb := Rn
        if no packet is in transmission then
```

```
Transmit a packet where Sb ≤ Sn ≤ Sm.
Packets are transmitted in order.
```

实现时使用了 java.util.Timer 和 java.util.TimerTask 这两个类。Timer 类的设置发送多久未收到相应的 ACK 为超时，设置时新建一个继承 TimerTask 类的 TimeOut 类，用于重传数据。当 Timer 设置的时间到了，则意味始终没有收到正确的 ACK，那么就程序就会执行 Timeout 类的 run 函数重发相应的数据分组。

```
class Timeout extends TimerTask {
    Timer timer;

    public Timeout(Timer timer){
        this.timer = timer;
    }

    @Override
    public void run() {
        try {
            TimeOut();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Timeout 类

```
public void start_timer(){

    timer.schedule(new Timeout(timer), 20); // 启动定时器

}

public void stop_timer(){
    timer.cancel();
    timer = new Timer();
}
```

启动和关闭定时器函数

重传的实现需要发送方先将发送的数据和相关信息缓存下来，超时则将分组编号为 base 到 nextseqnum - 1 的数据重发。实现的函数是 TimeOut 函数中，由于正常发送数据和重发数据是两个线程，二者运行相互独立，所以为了使得程序在重发的数据未收到相应的 ACK 消息时，正常发送的线程不会发送数据，函数实现时使用了 while 循环和布尔类型参数 stop，保证重传的逻辑正确。但有一点值得注意，重传一次后迅速再次重传是不对的，应该先让线程等待休眠，等待 ACK 消息，如果仍然没有接收到相应的 ACK 消息再能重发。

```
public void TimeOut() throws InterruptedException {
        int tmp = nextseqnum;
        stop = false;
        while(base != tmp && base != nextseqnum){
```

```
                int begin = base;
                int end = nextseqnum;
                for(int i = begin; i < end; i++){
                    buffer[0] = (byte)(i % 128);
                    buffer[1] = (byte)(i / 128);
                    rerdtSend(buffer, packetsize[i]);
                    if(i == begin){
                        start_timer();
                    }
                    System.out.println("ReSend a packet, id = " + i
+ ", size = " + packetsize[i]);
                }

                Thread.sleep(N * 25); // 避免无效的重复发送

            }
            stop = true;
        }
```

Timeout 函数

使用校验和进行差错重传。设置发送的数据块大小范围为(500，1000)，使用 buffer[1010]来存储校验和，校验和由数据的各个数据块的值相加得到，而 buffer[1011]到 buffer[1015]用来存储数据包含的数据块的数量及其位数。

```
    public byte[] make_packet(byte[] buffer, int len){
        long checksum = 0;
        for(int i = 2; i < len; i++){
            checksum += buffer[i];
        }
        buffer[1010] = (byte)checksum;
        int begin = 1011;
        int digit = 0;
        while(len > 0){
            buffer[begin++] = (byte)(len % 10);
            len /= 10;
            digit++;
        }
        buffer[1015] = (byte)digit;
        long cur = buffer[0] + buffer[1] * 128;
        return buffer;
    }
```

make_packet 函数

发送发在发送前将调用 make_packet 函数对数据打包，将计算得出数据的校验和与数据的长度存入 buffer 数组未使用的部分，数据包大小统一为 1111。接收方接收到数据包时，会调用 corrupt 函数来检查校验和，首先根据 buffer[1011]到 buffer[1015]计算出数据的长度，然后计算所有数据块的值得和，最终的结果与发送端发送过来的校验和即 buffer[1010]进行比

较，如果相等则说明数据完整，可以进行下一步判断，否则数据有丢失，丢弃。

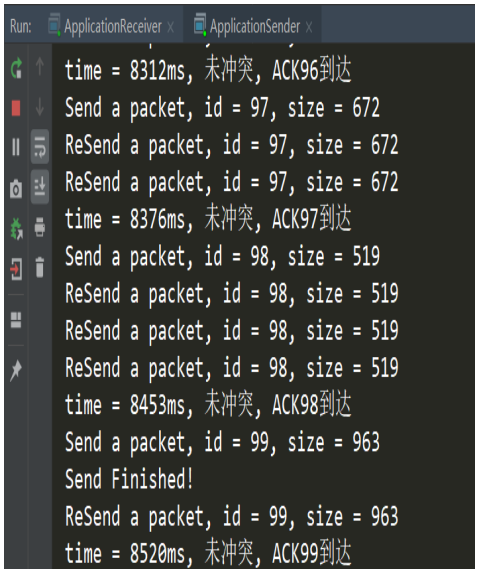
```
public boolean corrupt(byte[] buffer){ // 检查校验和
    long checksum = 0;
    int length = 0;
    int digit = buffer[1015];
    for(int i = 1011 + digit - 1; i >= 1011; i--){
        length = length * 10 + buffer[i];
    }
    for(int i = 2; i < length; i++){
        checksum += buffer[i];
    }
    if(checksum == buffer[1010]){
        return true;
    }
    return false;
}
```

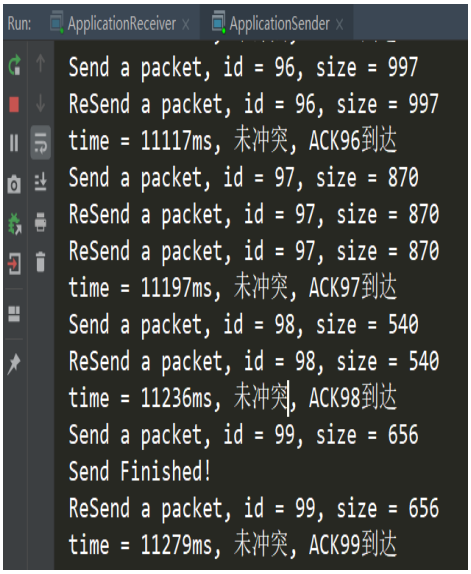
对于 Go-Back-N，如果 N 的值设为 1，则相当于停等式（Stop-Wait）传输，比较不同 N 值对传输速率的影响。

在下此次测试中，带宽、丢包率、误比特率和延迟时间统一设置如下：

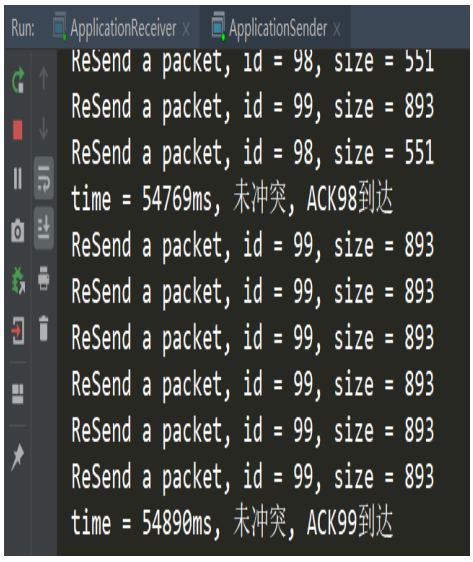
```
udtChannel.setBandwidth(1);
udtChannel.setLossRatio(1);
udtChannel.setBRT(1);
udtChannel.setPropDelay(5,15);
```

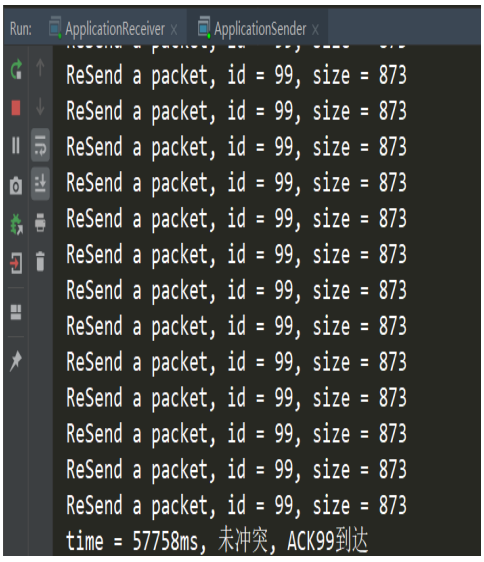
另外，程序还存在其他的影响因素，例如，当发送的报文数已达到窗口的最大限度时，线程休眠等待 20ms；每次重发结束后，线程等待 20ms,让重发的报文能收到相应的 ACK 消息，而不能立刻重发，这样不但会浪费报文，还会因为发送和传播延迟导致效率低。



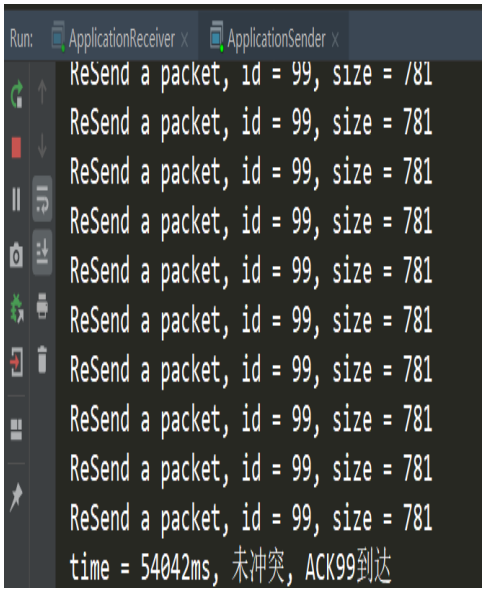


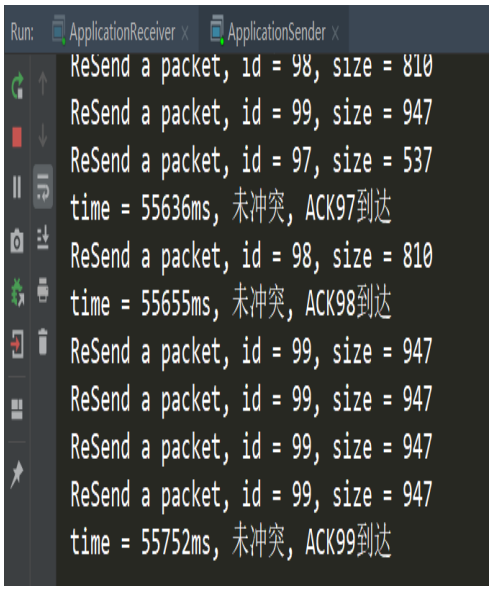
（1）N = 1, Delay = 20ms, Packet = 100





(2）N = 10, Delay = 20ms, Packet = 100





（3）N = 50, Delay = 20ms, Packet = 100

其他情况(无截图)：

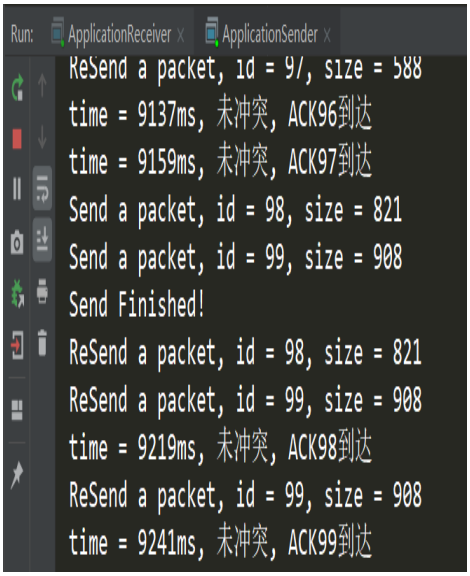
N	Delay	Packets	传输时间（ms）	
5	20	100	52593	52304
8	20	100	54764	56203
30	20	100	48953	50613
100	20	100	67684	76602
2	20	100	91617	90851
3	20	100	64214	64577
4	20	100	57487	58811

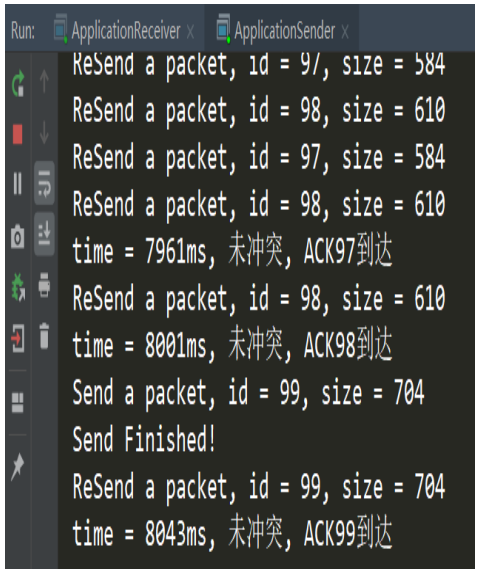
结果分析：

由以上十几组对照实验的结果可以看出，停等协议的传输效率是最高的，

当 N>1 时，传输的时间至少是停等协议的 5 倍以上，之所出现这样的情况是因为，程序模拟了丢包和误比特率的真实情况，这让数据在传输会出现错误，而出错重传的时间代价，GBN 协议远比停等协议要大。

而当 N >= 5 时，GBN 的传输效率几乎差不多，都是 50 多秒，而当 N < 5 时，传输效率反而更低，尤其是 N = 2 时，传输时间为 90 多秒，传输效率非常低。根据我的反复实验和分析得出，重发报文后线程休眠等待的时间对传输效率的影响很大，当我将此休眠等待时间设为 50ms 时，传输效率大大提升了，测试截图如下。这是因为当你重发时，重发的报文在被接收方接收到和接收方发送 ACK 消息到发送发都有端对端的延迟，反复的重发，不但有很多报文浪费掉了，而且时间开销是巨大的，所以让线程休眠等待一个合适的时间，





这才是符合实际的设计。

2、请设置不同的丢包率、误比特率、带宽、传播时延，模拟不同类型的通信信道，分析不同信道情况下，如何设置报文大小和窗口大小 N，实现传输速率最大化。

丢包率	误比特率	带宽	传播时延	窗口大小 N	报文大小	传输时间（ms）
0.01	0.000005	1Mbs	(5,15)	1	1111	8639
0.01	0.000005	1Mbs	(5,15)	1	1439	12267
0.01	0.000005	1Mbs	(5,15)	5	1439	18069

0.01	0.000005	1Mbs	(5,15)	10	1111	17729
0.05	0.000005	1Mbs	(5,15)	1	1111	9697
0.05	0.000005	1Mbs	(5,15)	1	1439	8570
0.05	0.000005	1Mbs	(5,15)	5	1439	15147
0.05	0.000005	1Mbs	(5,15)	10	1111	22329
0.01	0.000005	5Mbs	(5,15)	1	1111	7385
0.01	0.000005	5Mbs	(5,15)	1	1439	8025
0.01	0.000005	5Mbs	(5,15)	5	1439	13338
0.01	0.000005	5Mbs	(5,15)	10	1111	19498
0.01	0.000005	1Mbs	(10,20)	1	1111	7388
0.01	0.000005	1Mbs	(10,20)	1	1439	8273
0.01	0.000005	1Mbs	(10,20)	5	1439	15943
0.01	0.000005	1Mbs	(10,20)	10	1111	17170
0.01	0.00001	1Mbs	(5,15)	1	1111	9279
0.01	0.00001	1Mbs	(5,15)	1	1439	11273
0.01	0.00001	1Mbs	(5,15)	5	1439	30925
0.01	0.00001	1Mbs	(5,15)	2	1111	11533
0.01	0.00001	1Mbs	(5,15)	10	1111	35660

结果分析：

根据控制变量的对照实验，在不同的信道环境下，窗口越小，报文大小越小，传输效率越高。

当 N = 1 即相当于停等协议时传输效率很高，但停等协议有着非常低的发送方利用率，而 GBN 协议，虽然接收方会丢失失序分组，但为了让接收方按序将数据交付上层，这样做是有道理的，还能有效的提升发送方的利用率。

四、 个人贡献说明

（每组人数最多 5 人，请说明完成实验过程中本人的分工或贡献。）

我在本实验中一同参与了代码的编写，还负责了程序的调试。