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-				
姓名	[redacted]		院系	计算学部	
班级	[redacted]		学号	[redacted]	
任课教师	刘亚维		指导教师		
实验地点	格物 207		实验时间	[redacted]	
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					

计算学部

实验目的：

（注：实验报告模板中的各项内容仅供参考，可依照实际实验情况进行修改。）

本次实验的主要目的。

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。

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：

概述本次实验的主要内容，包含的实验项等。

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；（选作内容，加分 项目，可以当堂完成或课下完成）

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

5) 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）。

6) 模拟引入数据包的丢失，验证所设计协议的有效性。

7) 改进所设计的 GBN 协议，支持双向数据传输；（选作内容，加分 项目，可以当堂完成或课下完成）

8）将所设计的 GBN 协议改进为 SR 协议。（ 选作内容，加分项目， 可以当堂完成或课下完成）

实验过程：

如果想实现停等协议，只需先实现 GBN 协议，然后在对协议进行改造，设置窗口大小为 1 即可，因此，直接设计 GBN 协议

定义的局部变量

修饰符和类型	字段	说明
protected int	base	当前窗口起始位置。
protected int	DATA_NUMBER_Rev	接收方接受的分组个数。
protected int	DATA_NUMBER_Send	发送方发送分组个数。
protected java.util.List<java.lang.Integer>	dataLengthList	维护每个数据报文的中数据长度
protected byte[]	dataList	储存文件数据
protected java.net.InetAddress	destAddress	分组发送的目标地址。
protected int	destPort	发送分组的目标端口，初始化为 80

protected int	expectedSeq	期望收到的分组序列号。
protected java.lang.String	hostName	这个主机的名称。
protected boolean	isRevCarriedData	标志是否接受文件
protected boolean	isSendCarryData	标志是否发送文件
protected int	lastSave	储存上一个保存文件的报文标号，用于接受数据
protected int	MaxDataPacketSize	每个数据报文最大的储存的文件大小
protected int	nextSeq	下一个发送的分组。
protected java.lang.String	outputFileName	输出文件名
protected java.net.DatagramSocket	receiveSocket	接收分组使用的 socket。
protected java.net.DatagramSocket	sendSocket	发送数据使用的 socket。
protected int	TIMEOUT	超时时间，秒。
protected int	WINDOW_SIZE	窗口长度。

GBN 协议数据分组格式：String

\\w*: Sending to port \\d+, Seq = (\\d+) isDataCarried =(true|false) length = (\\d+) DATA_NUMBER = (\\d+)
+@@@@@ + data（二进制流）

确认分组格式： String
hostName + " responses ACK: " + seq;
hostName 为主机名
seq 为 ACK 对应的序列号

发送报文函数：

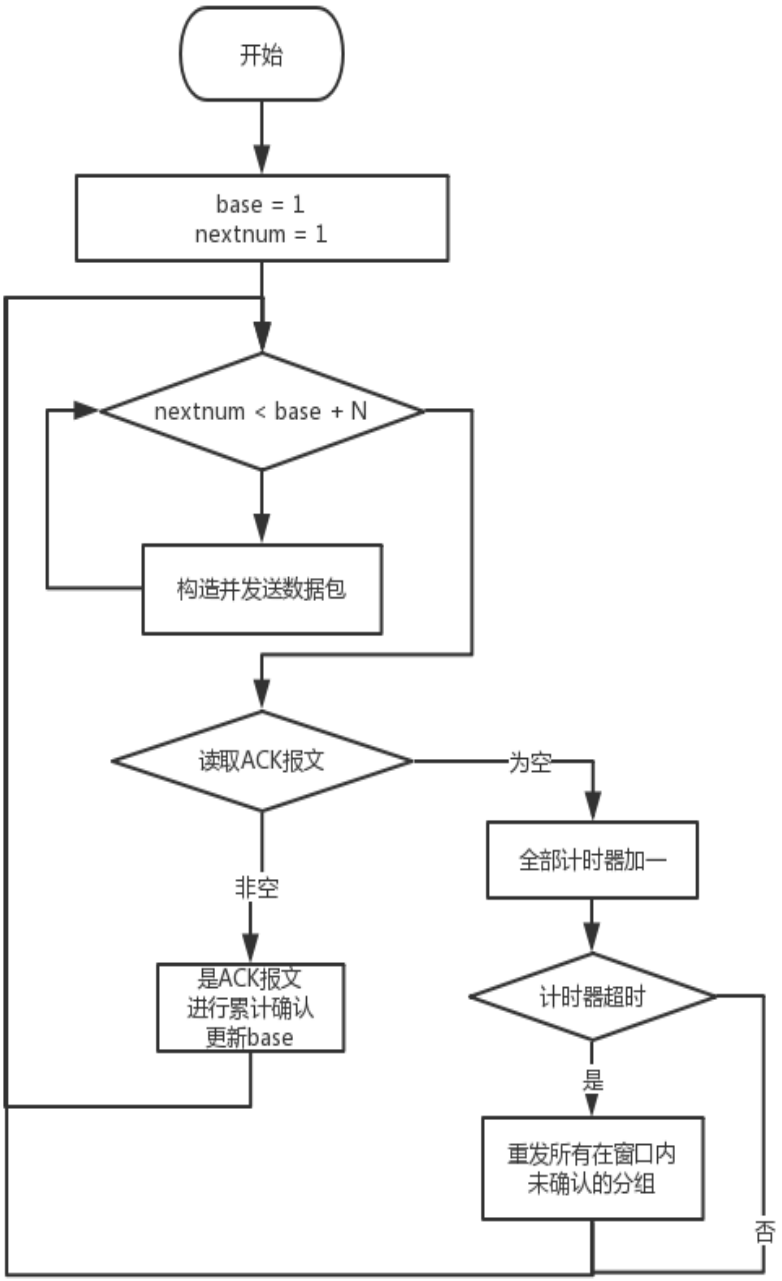
设置循环发送分组，判断 nextSeq 是否在窗口内，如果在就发送到预设的 IP 和端口，一次性发完一整个窗口内的分组并且等待对应分组内所有序列号的 ACK 被返回。

根据接收端返回的 ACK 编号调整窗口 base 的值为 ACK+1，直到窗口最大值，并设定超时时间，达到超时时间后调用 timeout()函数

```
1.  /**
2.      * 发送报文
3.      * @throws IOException
4.      */
5.     public void send() throws IOException {
6.         int maxACK = 0;
```

```
7.          while (true) {
8.              // 发送分组循环
9.              while (nextSeq < base + WINDOW_SIZE && nextSeq <= DATA_NUMBER_Send) {
10.                 // 模拟数据丢失
11.                 if (nextSeq % 5 == 0||nextSeq == 8) {
12.                     System.out.println(hostName + "模拟丢失报文：Seq = " + nextSeq);
13.                     nextSeq++;
14.                     continue;
15.                 }
16.                 byte[] data=new byte[MaxDataPacketSize];
17.                 int length=0;
18.                 if(isSendCarryData){
19.                     length=dataLengthList.get(nextSeq-1);
20.                     int curByte=0;
21.                     for(int i=0 ;i<nextSeq-1;i++){
22.                         curByte+=dataLengthList.get(i);
23.                     }
24.                     System.arraycopy(dataList,curByte,data,0,length);
25.                 }
26.                 String sendDataLabel = hostName + ": Sending to port " + destPor
27. t + ", Seq = " + nextSeq
28.                         +" isDataCarried ="+isSendCarryData+" length = "+length +"
29.  DATA_NUMBER = "+DATA_NUMBER_Send +"@@@@@";
30.
31.                 byte[] datagram = addBytes(sendDataLabel.getBytes(),data);
32.
33.
34.                 DatagramPacket datagramPacket = new DatagramPacket(datagram, datagram.
35. length, destAddress, destPort);
36.                 sendSocket.send(datagramPacket);
37.
38.                 System.out.println(hostName + "发送到" + destPort + "端口，  Seq =
39. " + nextSeq);
40.                 nextSeq=nextSeq+1;
41.                 try {
42.                     Thread.sleep(300);
43.                 } catch (InterruptedException e) {
44.                     e.printStackTrace();
45.                 }
46.
47.             }
48.
49.             // 用尽窗口后开始接收 ACK
50.             byte[] bytes = new byte[4096];
```

```
51.             DatagramPacket datagramPacket = new DatagramPacket(bytes, bytes.length);
52.             sendSocket.setSoTimeout(1000*TIMEOUT);
53.             try {
54.                 sendSocket.receive(datagramPacket);
55.             } catch (SocketTimeoutException ex) {
56.                 System.out.println(hostName + " 等待 ACK:Seq="+base+"超时");
57.                 timeOut();
58.                 continue;
59.             }
60.             // 转换成 String
61.             String fromServer = new String(datagramPacket.getData(), 0, datagramP
62. acket.getLength());
63.             // 解析出 ACK 编号
64.             int ack = Integer.parseInt(fromServer.substring(fromServer.in
65. dexOf("ACK: ") + "ACK: ".length()).trim());
66.             maxACK = Math.max(ack,maxACK);
67.             //ack 最后确认收到的报文
68.             //nextSeq 发送完本轮报文期待下一轮收到的报文
69.             base = maxACK+1;
70.
71.             System.out.println(hostName + "当前最后接收到的最大 ACK: " + maxACK);
72.
73.             if (maxACK == DATA_NUMBER_Send) {
74.                 // 如果发送完毕
75.                 // 停止计时器
76.                 System.out.println(hostName + "发送完毕，发送方收到了全部的 ACK 信息");
77.                 return;
78.             }
79.         }
80.     }
```



如果发生超时，说明发送端没有在限定时间内收到预期 ACK，会重新发送这个序列号 base 和其之后的所有分组。

超时处理函数：

```
1.  /**
2.     * 超时处理—重发数据
3.     * @throws IOException
4.     */
5.    public void timeOut() throws IOException {
6.        int curByte=0;
```

```
        for (int i =0;i<base-1&&isSendCarryData;i++){
            curByte=curByte+dataLengthList.get(i);
        }
        System.out.println(hostName+" 接受 ACK 超时，重发 Seq："+base+"--"+(nextSeq-1));
        for (int i = base; i < nextSeq; i++) {
            byte[] data=new byte[MaxDataPacketSize];
            int length=0;
            if(isSendCarryData){
                length=dataLengthList.get(i-1);
                System.arraycopy(dataList,curByte,data,0,length);
                curByte=curByte+length;
            }
            String sendDataLabel = hostName + ": Sending to port " + destPort + ",
Seq = " + i
                    +" isDataCarried ="+isSendCarryDataFalse+" length = "+length +" DATA_
NUMBER = "+DATA_NUMBER_Send +"@@@@@";

            // 模拟发送分组
            byte[] datagram = addBytes(sendDataLabel.getBytes(),data);

            DatagramPacket datagramPacket = new DatagramPacket(datagram, datagram.
length, destAddress, destPort);
            sendSocket.send(datagramPacket);

            System.out.println(hostName
                    + "重新发送发送到" + destPort + "端口， Seq = " + i);
        }
    }
```

报文接受函数：

GBN 中接收端维护一个“期望收到的序列号”，所以只要是收到了预期的序列号分组，就留下这个分组并且预期报文数值自增 1，然后返回此序列号的 ACK；如果不是，直接丢弃并返回上一个序列号的 ACK

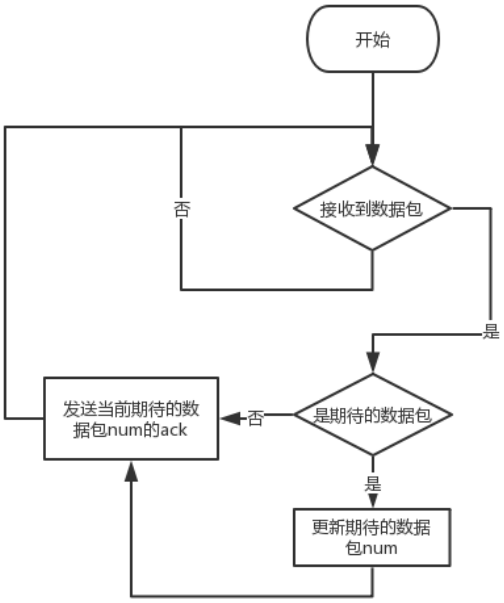
```
public void receive() throws IOException {

    File output = null;
    FileOutputStream fos = null;
    if(isRevCarriedData){
        output = new File(outputFileName);
        fos = new FileOutputStream(output);
    }

    while (true) {
```

```
11.          byte[] receivedData = new byte[Math.max(4096,MaxDataPacketSize+500)];
12.          DatagramPacket receivePacket = new DatagramPacket(receivedData, r
13. eceivedData.length);
14.          receiveSocket.setSoTimeout(1000*TIMEOUT);
15.          try {
16.              receiveSocket.receive(receivePacket);
17.          } catch (SocketTimeoutException ex) {
18.              System.out.println(hostName + " 在正在等待分组： Se
19. q= "+expectedSeq+"的到来 ");
20.              continue;
21.          }
22.
23.          // 收到的数据
24.          String receivedLabel = new String(receivedData, 0,receivedData.length );
25.          String label =receivedLabel.split("@@@@@")[0];
26.          int labelSize = (label+"@@@@@").getBytes().length;
27.
28.          String pattern = "\\w*: Sending to port \\d+, Seq = (\\d+) isDataCarried
29. =(true|false) length = (\\d+) DATA_NUMBER = (\\d+)";
30.          Matcher matcher = Pattern.compile(pattern).matcher(label);
31.
32.          if (!matcher.find()) {
33.              System.out.println(hostName + " 收到错误数据"+label);
34.              // 仍发送之前的 ACK
35.              sendACK(expectedSeq - 1, receivePacket.getAddress(), receiv
36. ePacket.getPort());
37.              continue;
38.          }
39.
40.          int receivedSeq=Integer.parseInt(matcher.group(1));
41.          isRevCarriedData=Boolean.parseBoolean(matcher.group(2));
42.          int dataLength = Integer.parseInt(matcher.group(3));
43.          DATA_NUMBER_Rev = Integer.parseInt(matcher.group(4));
44.
45.          if (receivedSeq  == expectedSeq) {
46.              // 收到了预期的数据
47.              System.out.println(hostName + " 收到了期待的数据,发送 ACK:
    Seq = " + expectedSeq);
48.              if(isRevCarriedData&&lastSave == receivedSeq -1 ){
49.
50.                  System.out.println(hostName + "写入数据 " + expectedSeq );
51.                  //System.out.println(hostName+" 收到的数据: "+new String(receivedD
52. ata,0,receivedData.length) );
53.                  fos.write(receivedData,labelSize,dataLength);
```

```
54.                 lastSave = receivedSeq ;
55.             }
56.             // 发送 ACK
57.             if (expectedSeq % 7 == 0) {
58.                 //不发送 ACK
59.                 System.out.println(hostName + "收到了期待的数据,但是模拟丢失 ACK: " +
60. expectedSeq);
61.             } else {
62.                 sendACK(expectedSeq, receivePacket.getAddre
63. ss(), receivePacket.getPort());
64.
65.             }
66.
67.             if (expectedSeq == DATA_NUMBER_Rev) {
68.                 System.out.println(hostName + "接受完成");
69.                 if(isRevCarriedData){
70.                     fos.flush();
71.                     fos.close();
72.                 }
73.
74.                 return;
75.             }
76.             // 期待值加 1
77.             expectedSeq++;
78.
79.         } else {
80.             // 未收到预期的 Seq
81.             System.out.println(hostName + " 实际收到的数据 Seq ="+receivedSeq+ ‘
82. ", 然而期待顺序收到数据 Seq = " + expectedSeq+" 因此丢弃此分组");
83.             // 仍发送之前的 ACK
84.             sendACK(expectedSeq - 1, receivePacket.get
85. Address(), receivePacket.getPort());
86.         }
87.
88.     }
89. }
```



ACK 发送函数:

根据接受到的报文的 seq 返回对应的报文

```
1.  /**
2.      * 向发送方回应 ACK。
3.      *
4.      * @param seq    ACK 序列号
5.      * @param toAddr 目的地址
6.      * @param toPort 目的端口
7.      * @throws IOException socket 相关错误时抛出
8.      */
9.     protected void sendACK(int seq, InetAddress toAddr, int toPort) throws IOException {
10.         String response = hostName + " responses ACK: " + seq;
11.         byte[] responseData = response.getBytes();
12.         // 获得来源 IP 地址和端口，确定发给谁
13.         DatagramPacket responsePacket = new DatagramPacket(responseDat
14. a, responseData.length, toAddr, toPort);
15.         receiveSocket.send(responsePacket);
16.     }
```

改造为停等协议：

只需要设定窗口大小为 1 即可

```
1.  GBN sender = new GBN(20001, 1, 20, 2, "Host Alice");
2.     sender.setDestPort(20002);
```

```
3.     GBN receiver = new GBN(20002, 1, 20, 2, "Host Bob");
```

模拟报文的丢失

报文的序列号模 5 为 0 或等于 8 的情况下模拟报文丢失，就是不发送报文

```
1.  if (nextSeq % 5 == 0||nextSeq == 8) {
2.                      System.out.println(hostName + "模拟丢失报文：Seq = " + nextSeq);
3.                      nextSeq++;
4.                      continue;
5.                  }
```

模拟丢失 ACK

模 7 为 0 不返回 ACK 即可模拟丢失 ACK

```
1.  if (expectedSeq % 7 == 0) {
2.                      //不发送 ACK
3.                      System.out.println(hostName + "收到了期待的数据,但是模拟丢失
    ACK: " + expectedSeq);
4.                  } else {
5.                      sendACK(expectedSeq, receivePacket.getAddress(), re
6.  ceivePacket.getPort());
7.
8.                  }
```

改造为双向传输：

每个 GBN 类既有传送函数也有接受函数，都包含 各自一套相互独立的变量，因此可以实现双向传输，这时候需要创建四个线程，两个主机各自启动发送和接收就可以了

添加文件传输功能

用两个函数分别包装 send()和 receive()函数 并控制文件读写即可实现文件传输功能，具体的传输和读写已在 send()和 receive()中体现

```
1.   public void sendData(String filename,int MaxDataPacketSize) throws IOException {
2.          File file=new File(filename);
3.          this.MaxDataPacketSize=MaxDataPacketSize;
4.          if(file.length()==0){
5.              System.out.println("文件为空！");
6.              return;
7.          }
8.          try {
9.              DATA_NUMBER_Send=0;
```

```
            FileInputStream fis = new FileInputStream(file);
            byte[] bytes = new byte[MaxDataPacketSize];
            int length = 0;
            while((length = fis.read(bytes, 0, bytes.length)) != -1) {
                DATA_NUMBER_Send++;
                dataList=addBytes(dataList,bytes);
                dataLengthList.add(length);
            }
            isSendCarryData=true;
            System.out.println(hostName+":文件被拆分为"+DATA_NUMBER_Send+"个包");
            fis.close();
        } catch (IOException e) {
            e.printStackTrace();
            return;
        }

        send();


    }
public void receiveData (String fileName,int MaxDataPacketSize) throws IOException {

        isRevCarriedData = true;
        this.MaxDataPacketSize=MaxDataPacketSize;
        if(!fileName.isBlank()){
            outputFileName=fileName;
            receive();

        }else {
            System.out.println("文件名为空！");
        }
```

改造为 SR 协议

只需要在 BGN 协议的基础上，维护发送方时发送过的分组，送方时收到的 ACK。收方时收到的缓存分组三个集合即可

```
    /**
   * 作为发送方时发送过的分组。
   */
  private Set<Integer> senderSentSet = new HashSet<>();

  /**
```

```
7.   * 作为发送方时收到的 ACK。
8.   */
9.  private Set<Integer> senderReceivedACKSet = new HashSet<>();
10.
11. /**
12.  * 作为接收方时收到的分组，用来作为缓存。
13.  */
14. private Set<Integer> receiverReceivedSet = new HashSet<>();
```

然后在代码中根据这些集合，在发送时每次只等待窗口最左端的 base 序号的数据包，收到则窗口右移，超时则重发，接收数据时收到乱序的但是在窗口内的数据包时不再丢弃，而是缓存起来

发送方的改造：

```
1.    /**
2.   * 作为发送方时发送过的分组。
3.   */
4.  private Set<Integer> senderSentSet = new HashSet<>();
5.
6.  /**
7.   * 作为发送方时收到的 ACK。
8.   */
9.  private Set<Integer> senderReceivedACKSet = new HashSet<>();
10.
11. /**
12.  * 作为接收方时收到的分组，用来作为缓存。
13.  */
14. private Set<Integer> receiverReceivedSet = new HashSet<>();
```

接收方的改造

```
1.  if (seq >= rcvBase && seq <= rcvBase + WINDOW_SIZE - 1) {
2.              //收到了接受窗口里存在的分组
3.              receiverReceivedSet.add(seq);
4.              System.out.println(hostName + "收到一个接收方窗口内的分组，Seq = " + seq + "已确
    认");
5.              sendACK(seq, receivePacket.getAddress(), receivePacket.getPort());
6.              if (seq == rcvBase) {
7.                  // 收到这个分组后可以开始滑动
8.                  while (receiverReceivedSet.contains(rcvBase)) {
9.                      rcvBase++;
10.                 }
11.                 if (rcvBase == DATA_NUMBER + 1) {
12.                     // 停止计时器
13.                     System.out.println(hostName + "接受完毕，发送方收到了全部的数据");
14.                     return;
```

```
                }
            }

        }  else {
            // 这个分组序列号太大，不在窗口内，应该舍弃
            System.out.println(hostName + "收到一个不在窗口内的分组，Seq = " + seq + "因此丢弃此分组");
        }
```

实验结果：

1.停等协议

创建两个 GBN 类，设定窗口大小为 1

```
GBN sender = new GBN(20001, 1, 20, 2, "Host Alice");
    sender.setDestPort(20002);
    GBN receiver = new GBN(20002, 1, 20, 2, "Host Bob");
    new Thread(() -> {
        try {
            sender.send();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }).start();
    new Thread(() -> {
        try {
            receiver.receive();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }).start();
```

程序输出：

```
  Host Alice 发送到 20002 端口， Seq = 1
Host Bob 收到了期待的数据,发送 ACK: Seq = 1
Host Alice 当前最后接收到的最大 ACK: 1
Host Alice 发送到 20002 端口， Seq = 2
Host Bob 收到了期待的数据,发送 ACK: Seq = 2
Host Alice 当前最后接收到的最大 ACK: 2
Host Alice 发送到 20002 端口， Seq = 3
Host Bob 收到了期待的数据,发送 ACK: Seq = 3
Host Alice 当前最后接收到的最大 ACK: 3
Host Alice 发送到 20002 端口， Seq = 4
```

```
Host Bob 收到了期待的数据,发送 ACK：Seq = 4
Host Alice 当前最后接收到的最大 ACK: 4
Host Alice 模拟丢失报文：Seq = 5
Host Bob 在正在等待分组： Seq= 5 的到来
Host Alice 等待 ACK:Seq=5 超时
Host Alice 接受 ACK 超时，重发 Seq：5--5
Host Alice 重新发送发送到 20002 端口， Seq = 5
Host Bob 收到了期待的数据,发送 ACK：Seq = 5
Host Alice 当前最后接收到的最大 ACK: 5
Host Alice 发送到 20002 端口， Seq = 6
Host Bob 收到了期待的数据,发送 ACK：Seq = 6
Host Alice 当前最后接收到的最大 ACK: 6
Host Alice 发送到 20002 端口， Seq = 7
Host Bob 收到了期待的数据,发送 ACK：Seq = 7
Host Bob 收到了期待的数据,但是模拟丢失 ACK: 7
Host Bob 在正在等待分组： Seq= 8 的到来
Host Alice 等待 ACK:Seq=7 超时
Host Alice 接受 ACK 超时，重发 Seq：7--7
Host Alice 重新发送发送到 20002 端口， Seq = 7
Host Bob 实际收到的数据 Seq =7，然而期待顺序收到数据 Seq = 8 因此丢弃此分组
Host Alice 当前最后接收到的最大 ACK: 7
Host Alice 模拟丢失报文：Seq = 8
Host Bob 在正在等待分组： Seq= 8 的到来
Host Alice 等待 ACK:Seq=8 超时
Host Alice 接受 ACK 超时，重发 Seq：8--8
Host Alice 重新发送发送到 20002 端口， Seq = 8
Host Bob 收到了期待的数据,发送 ACK：Seq = 8
Host Alice 当前最后接收到的最大 ACK: 8
Host Alice 发送到 20002 端口， Seq = 9
Host Bob 收到了期待的数据,发送 ACK：Seq = 9
Host Alice 当前最后接收到的最大 ACK: 9
Host Alice 模拟丢失报文：Seq = 10
Host Bob 在正在等待分组： Seq= 10 的到来
Host Alice 等待 ACK:Seq=10 超时
Host Alice 接受 ACK 超时，重发 Seq：10--10
Host Alice 重新发送发送到 20002 端口， Seq = 10
Host Bob 收到了期待的数据,发送 ACK：Seq = 10
Host Alice 当前最后接收到的最大 ACK: 10
Host Alice 发送到 20002 端口， Seq = 11
Host Bob 收到了期待的数据,发送 ACK：Seq = 11
Host Alice 当前最后接收到的最大 ACK: 11
Host Alice 发送到 20002 端口， Seq = 12
Host Bob 收到了期待的数据,发送 ACK：Seq = 12
Host Alice 当前最后接收到的最大 ACK: 12
```

```
Host Alice 发送到 20002 端口， Seq = 13
Host Bob 收到了期待的数据,发送 ACK：Seq = 13
Host Alice 当前最后接收到的最大 ACK: 13
Host Alice 发送到 20002 端口， Seq = 14
Host Bob 收到了期待的数据,发送 ACK：Seq = 14
Host Bob 收到了期待的数据,但是模拟丢失 ACK: 14
Host Bob 在正在等待分组： Seq= 15 的到来
Host Alice 等待 ACK:Seq=14 超时
Host Alice 接受 ACK 超时，重发 Seq：14--14
Host Alice 重新发送发送到 20002 端口， Seq = 14
Host Bob 实际收到的数据 Seq =14，然而期待顺序收到数据 Seq = 15 因此丢弃此分组
Host Alice 当前最后接收到的最大 ACK: 14
Host Alice 模拟丢失报文：Seq = 15
Host Bob 在正在等待分组： Seq= 15 的到来
Host Alice 等待 ACK:Seq=15 超时
Host Alice 接受 ACK 超时，重发 Seq：15--15
Host Alice 重新发送发送到 20002 端口， Seq = 15
Host Bob 收到了期待的数据,发送 ACK：Seq = 15
Host Alice 当前最后接收到的最大 ACK: 15
Host Alice 发送到 20002 端口， Seq = 16
Host Bob 收到了期待的数据,发送 ACK：Seq = 16
Host Alice 当前最后接收到的最大 ACK: 16
Host Alice 发送到 20002 端口， Seq = 17
Host Bob 收到了期待的数据,发送 ACK：Seq = 17
Host Alice 当前最后接收到的最大 ACK: 17
Host Alice 发送到 20002 端口， Seq = 18
Host Bob 收到了期待的数据,发送 ACK：Seq = 18
Host Alice 当前最后接收到的最大 ACK: 18
Host Alice 发送到 20002 端口， Seq = 19
Host Bob 收到了期待的数据,发送 ACK：Seq = 19
Host Alice 当前最后接收到的最大 ACK: 19
Host Alice 模拟丢失报文：Seq = 20
Host Bob 在正在等待分组： Seq= 20 的到来
Host Alice 等待 ACK:Seq=20 超时
Host Alice 接受 ACK 超时，重发 Seq：20--20
Host Alice 重新发送发送到 20002 端口， Seq = 20
Host Bob 收到了期待的数据,发送 ACK：Seq = 20
Host Alice 当前最后接收到的最大 ACK: 20
```

可以看出很好的实现了停等协议

2.GBN 协议：

```
GBN sender = new GBN(20001, 6, 10, 2, "Host Alice");
```

```
2.     sender.setDestPort(20002);
3.     GBN receiver = new GBN(20002, 6, 10, 2, "Host Bob");
```

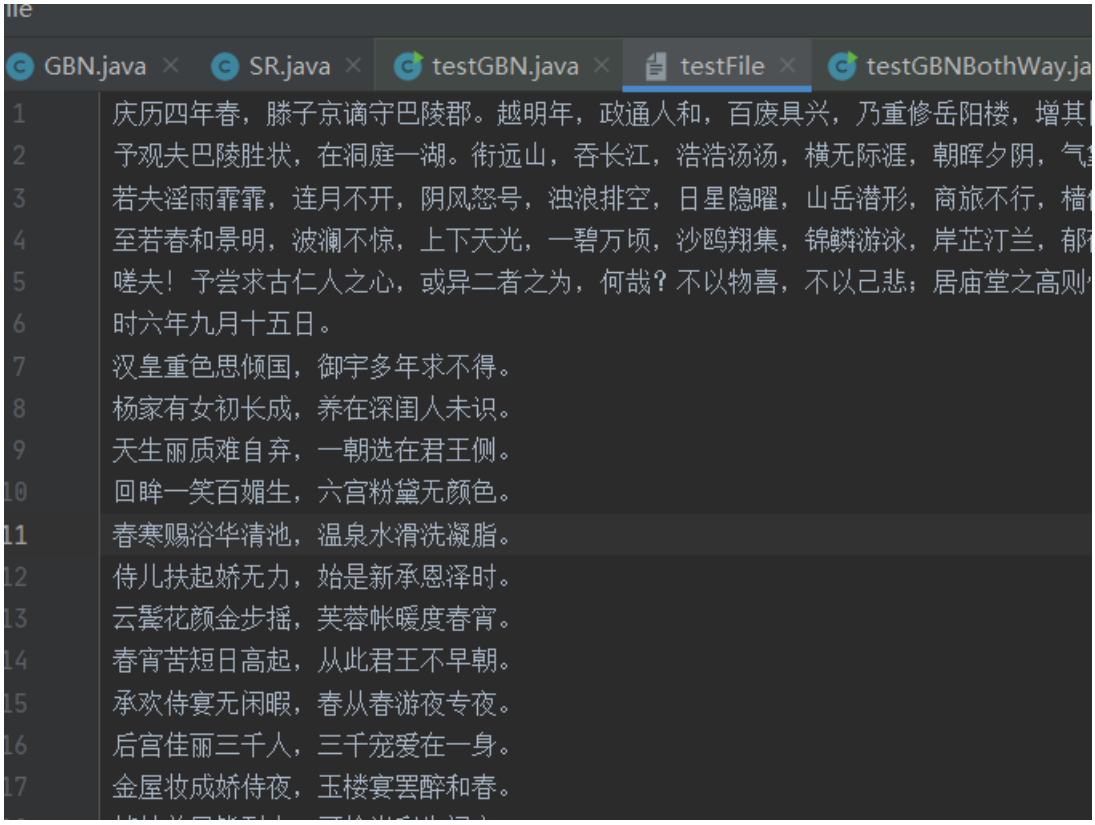
同样开两个线程，一个收一个发

```
1.  Host Bob 收到了期待的数据,发送 ACK：Seq = 1
2. Host Alice 发送到 20002 端口， Seq = 1
3. Host Alice 发送到 20002 端口， Seq = 2
4. Host Bob 收到了期待的数据,发送 ACK：Seq = 2
5. Host Alice 发送到 20002 端口， Seq = 3
6. Host Bob 收到了期待的数据,发送 ACK：Seq = 3
7. Host Alice 发送到 20002 端口， Seq = 4
8. Host Bob 收到了期待的数据,发送 ACK：Seq = 4
9. Host Alice 模拟丢失报文：Seq = 5
10. Host Alice 发送到 20002 端口， Seq = 6
11. Host Bob 实际收到的数据 Seq =6，然而期待顺序收到数据 Seq = 5 因此丢弃此分组
12. Host Alice 当前最后接收到的最大 ACK: 1
13. Host Alice 发送到 20002 端口， Seq = 7
14. Host Bob 实际收到的数据 Seq =7，然而期待顺序收到数据 Seq = 5 因此丢弃此分组
15. Host Alice 当前最后接收到的最大 ACK: 2
16. Host Alice 模拟丢失报文：Seq = 8
17. Host Alice 当前最后接收到的最大 ACK: 3
18. Host Alice 发送到 20002 端口， Seq = 9
19. Host Bob 实际收到的数据 Seq =9，然而期待顺序收到数据 Seq = 5 因此丢弃此分组
20. Host Alice 当前最后接收到的最大 ACK: 4
21. Host Alice 模拟丢失报文：Seq = 10
22. Host Alice 当前最后接收到的最大 ACK: 4
23. Host Alice 当前最后接收到的最大 ACK: 4
24. Host Alice 当前最后接收到的最大 ACK: 4
25. Host Bob 在正在等待分组： Seq= 5 的到来
26. Host Alice 等待 ACK:Seq=5 超时
27. Host Alice 接受 ACK 超时，重发 Seq：5--10
28. Host Bob 收到了期待的数据,发送 ACK：Seq = 5
29. Host Alice 重新发送发送到 20002 端口， Seq = 5
30. Host Alice 重新发送发送到 20002 端口， Seq = 6
31. Host Alice 重新发送发送到 20002 端口， Seq = 7
32. Host Alice 重新发送发送到 20002 端口， Seq = 8
33. Host Bob 收到了期待的数据,发送 ACK：Seq = 6
34. Host Alice 重新发送发送到 20002 端口， Seq = 9
35. Host Alice 重新发送发送到 20002 端口， Seq = 10
36. Host Alice 当前最后接收到的最大 ACK: 5
37. Host Alice 当前最后接收到的最大 ACK: 6
38. Host Bob 收到了期待的数据,发送 ACK：Seq = 7
39. Host Bob 收到了期待的数据,但是模拟丢失 ACK: 7
```

```
40. Host Bob 收到了期待的数据,发送 ACK：Seq = 8
41. Host Alice 当前最后接收到的最大 ACK: 8
42. Host Bob 收到了期待的数据,发送 ACK：Seq = 9
43. Host Alice 当前最后接收到的最大 ACK: 9
44. Host Bob 收到了期待的数据,发送 ACK：Seq = 10
45. Host Alice 当前最后接收到的最大 ACK: 10
```

3.发送文件

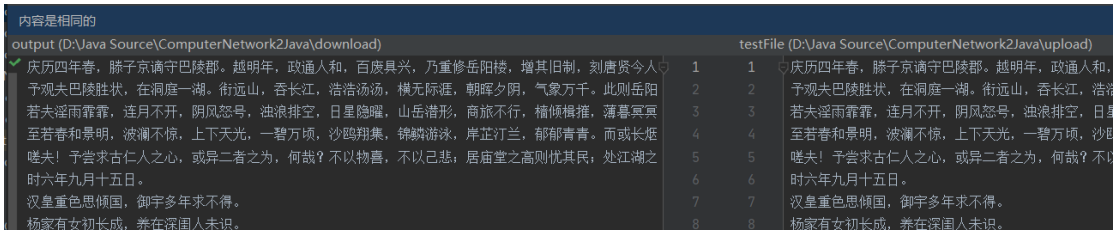
传输下图所示的文件



从控制台看到完成了传输

```
Host Bob 收到了期待的数据,发送ACK：Seq = 9
Host Bob写入数据 9
Host Alice当前最后接收到的最大ACK: 9
Host Bob接受完成
Host Alice发送完毕，发送方收到了全部的ACK信息
```

比较上传和下载的文件，可以看出两者是相同的



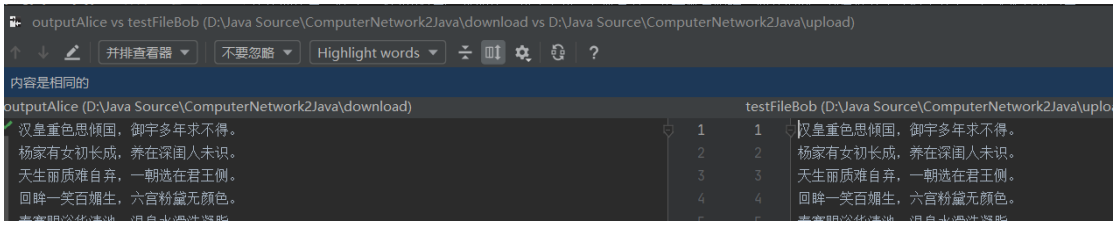
4.双向传输

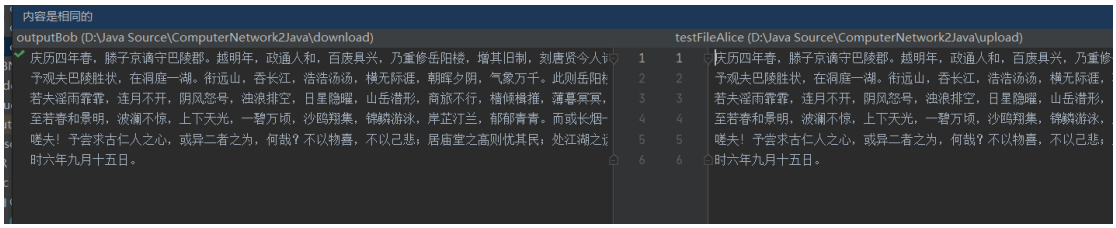
令 Alice 和 Bob 分别传输两个文件并同时接受

```
1.  Host Alice:文件被拆分为 3 个包
2.  Host Bob:文件被拆分为 6 个包
3.  Host Alice 收到了期待的数据,发送 ACK：Seq = 1
4.  Host Bob 收到了期待的数据,发送 ACK：Seq = 1
5.  Host Alice 写入数据 1
6.  Host Bob 写入数据 1
7.  Host Alice 发送到 20002 端口， Seq = 1
8.  Host Bob 发送到 20001 端口， Seq = 1
9.  Host Alice 发送到 20002 端口， Seq = 2
10. Host Bob 发送到 20001 端口， Seq = 2
11. Host Alice 收到了期待的数据,发送 ACK：Seq = 2
12. Host Alice 写入数据 2
13. Host Bob 收到了期待的数据,发送 ACK：Seq = 2
14. Host Bob 写入数据 2
15. Host Bob 发送到 20001 端口， Seq = 3
16. Host Alice 发送到 20002 端口， Seq = 3
17. Host Bob 收到了期待的数据,发送 ACK：Seq = 3
18. Host Bob 写入数据 3
19. Host Alice 收到了期待的数据,发送 ACK：Seq = 3
20. Host Alice 写入数据 3
21. Host Bob 接受完成
22. Host Bob 发送到 20001 端口， Seq = 4
23. Host Alice 当前最后接收到的最大 ACK: 1
24. Host Alice 收到了期待的数据,发送 ACK：Seq = 4
25. Host Alice 写入数据 4
26. Host Alice 当前最后接收到的最大 ACK: 2
27. Host Alice 当前最后接收到的最大 ACK: 3
28. Host Alice 发送完毕，发送方收到了全部的 ACK 信息
29. Host Bob 模拟丢失报文：Seq = 5
30. Host Bob 当前最后接收到的最大 ACK: 1
31. Host Bob 发送到 20001 端口， Seq = 6
32. Host Alice 实际收到的数据 Seq =6，然而期待顺序收到数据 Seq = 5 因此丢弃此分组
33. Host Bob 当前最后接收到的最大 ACK: 2
34. Host Bob 当前最后接收到的最大 ACK: 3
35. Host Bob 当前最后接收到的最大 ACK: 4
36. Host Bob 当前最后接收到的最大 ACK: 4
37. Host Alice 在正在等待分组： Seq= 5 的到来
38. Host Bob 等待 ACK:Seq=5 超时
39. Host Bob 接受 ACK 超时，重发 Seq：5--6
40. Host Bob 重新发送发送到 20001 端口， Seq = 5
```

```
41. Host Bob 重新发送发送到 20001 端口， Seq = 6
42. Host Alice 收到了期待的数据,发送 ACK：Seq = 5
43. Host Alice 写入数据 5
44. Host Bob 当前最后接收到的最大 ACK：5
45. Host Alice 收到了期待的数据,发送 ACK：Seq = 6
46. Host Alice 写入数据 6
47. Host Alice 接受完成
48. Host Bob 当前最后接收到的最大 ACK：6
49. Host Bob 发送完毕，发送方收到了全部的 ACK 信息
```

比较两者接受和发送的文件，都是相同的





5.SR 协议

同样使用 GBN 相同的条件进行测试：
控制台输出：

```
1.    Host Alice 发送到 20002 端口， Seq = 1
2.  Host Bob 收到一个接收方窗口内的分组，Seq = 1 已确认
3.  Host Alice 发送到 20002 端口， Seq = 2
4.  Host Bob 收到一个接收方窗口内的分组，Seq = 2 已确认
5.  Host Alice 发送到 20002 端口， Seq = 3
6.  Host Bob 收到一个接收方窗口内的分组，Seq = 3 已确认
7.  Host Alice 发送到 20002 端口， Seq = 4
8.  Host Bob 收到一个接收方窗口内的分组，Seq = 4 已确认
9.  Host Alice 模拟丢失报文：Seq = 5
10. Host Alice 当前窗口 [2,6]
11. Host Alice 收到了 ACK：1
12. Host Alice 发送到 20002 端口， Seq = 6
13. Host Bob 收到一个接收方窗口内的分组，Seq = 6 已确认
14. Host Alice 当前窗口 [3,7]
15. Host Alice 收到了 ACK：2
```

```
Host Alice 模拟丢失报文：Seq = 7
Host Alice 当前窗口 [4,8]
Host Alice 收到了 ACK: 3
Host Alice 发送到 20002 端口， Seq = 8
Host Bob 收到一个接收方窗口内的分组，Seq = 8 已确认
Host Alice 当前窗口 [5,9]
Host Alice 收到了 ACK: 4
Host Alice 发送到 20002 端口， Seq = 9
Host Bob 收到一个接收方窗口内的分组，Seq = 9 已确认
Host Alice 收到了 ACK: 6
Host Alice 收到了 ACK: 8
Host Alice 收到了 ACK: 9
Host Bob 正在等待分组的到来
Host Alice 等待 ACK:Seq=5 超时
Host Bob 收到一个接收方窗口内的分组，Seq = 5 已确认
Host Alice 重新发送发送到 20002 端口， Seq = 5
Host Alice 当前窗口 [7,11]
Host Alice 收到了 ACK: 5
Host Alice 模拟丢失报文：Seq = 10
Host Alice 发送到 20002 端口， Seq = 11
Host Bob 收到一个接收方窗口内的分组，Seq = 11 已确认
Host Alice 收到了 ACK: 11
Host Bob 正在等待分组的到来
Host Alice 等待 ACK:Seq=7 超时
Host Alice 重新发送发送到 20002 端口， Seq = 7
Host Bob 收到一个接收方窗口内的分组，Seq = 7 已确认
Host Alice 当前窗口 [10,14]
Host Alice 收到了 ACK: 7
Host Alice 发送到 20002 端口， Seq = 12
Host Bob 收到一个接收方窗口内的分组，Seq = 12 已确认
Host Alice 发送到 20002 端口， Seq = 13
Host Bob 收到一个接收方窗口内的分组，Seq = 13 已确认
Host Alice 发送到 20002 端口， Seq = 14
Host Bob 收到一个接收方窗口内的分组，Seq = 14 已确认
Host Alice 收到了 ACK: 12
Host Alice 收到了 ACK: 13
Host Alice 收到了 ACK: 14
Host Bob 正在等待分组的到来
Host Alice 等待 ACK:Seq=10 超时
Host Alice 重新发送发送到 20002 端口， Seq = 10
Host Bob 收到一个接收方窗口内的分组，Seq = 10 已确认
Host Alice 当前窗口 [15,19]
Host Alice 收到了 ACK: 10
Host Alice 模拟丢失报文：Seq = 15
```

```
60. Host Alice 发送到 20002 端口， Seq = 16
61. Host Bob 收到一个接收方窗口内的分组，Seq = 16 已确认
62. Host Alice 发送到 20002 端口， Seq = 17
63. Host Bob 收到一个接收方窗口内的分组，Seq = 17 已确认
64. Host Alice 发送到 20002 端口， Seq = 18
65. Host Bob 收到一个接收方窗口内的分组，Seq = 18 已确认
66. Host Alice 发送到 20002 端口， Seq = 19
67. Host Bob 收到一个接收方窗口内的分组，Seq = 19 已确认
68. Host Alice 收到了 ACK: 16
69. Host Alice 收到了 ACK: 17
70. Host Alice 收到了 ACK: 18
71. Host Alice 收到了 ACK: 19
72. Host Bob 正在等待分组的到来
73. Host Alice 等待 ACK:Seq=15 超时
74. Host Alice 重新发送发送到 20002 端口， Seq = 15
75. Host Bob 收到一个接收方窗口内的分组，Seq = 15 已确认
76. Host Alice 当前窗口 [20,24]
77. Host Alice 收到了 ACK: 15
78. Host Alice 模拟丢失报文：Seq = 20
79. Host Bob 正在等待分组的到来
80. Host Alice 等待 ACK:Seq=20 超时
81. Host Alice 重新发送发送到 20002 端口， Seq = 20
82. Host Bob 收到一个接收方窗口内的分组，Seq = 20 已确认
83. Host Alice 当前窗口 [21,25]
84. Host Alice 收到了 ACK: 20
85. Host Bob 接受完毕，发送方收到了全部的数据
86. Host Alice 发送完毕，发送方收到了全部的 ACK
```

问题讨论：

对实验过程中的思考问题进行讨论或回答。

遇到的问题：

JavaGBN和SR中对数据包组和数据包超时时间控制较为繁琐

使用DatagramPacket 类，并为通过设定超时时间，当超过一段时间没有收到报文就表明超时

```
1.  DatagramPacket datagramPacket = new DatagramPacket(bytes, bytes.length);
2.          sendSocket.setSoTimeout(1000*TIMEOUT);
3.          try {
4.              sendSocket.receive(datagramPacket);
5.          } catch (SocketTimeoutException ex) {
6.              System.out.println(hostName + " 等待 ACK:Seq="+base+"超时");
7.              timeOut();
8.              continue;
```

```
9.          }
```

心得体会：

结合实验过程和结果给出实验的体会和收获。

更好地掌握了停等协议，GBN协议和SR协议的内容，通过编程实现加深了对其的理解，实现了双工和文件传出，更好理解了Java的多线程和流的内容。