哈尔滨工业大学
Harbin Institute of Technology

计算机网络
课程实验报告

实验名称	可靠数据传输协议的设计与传输				
姓名	李世轩		院系	软件工程	
班级	2037102		学号	120L022109	
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 年 10 月 14 日	
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：

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。
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。

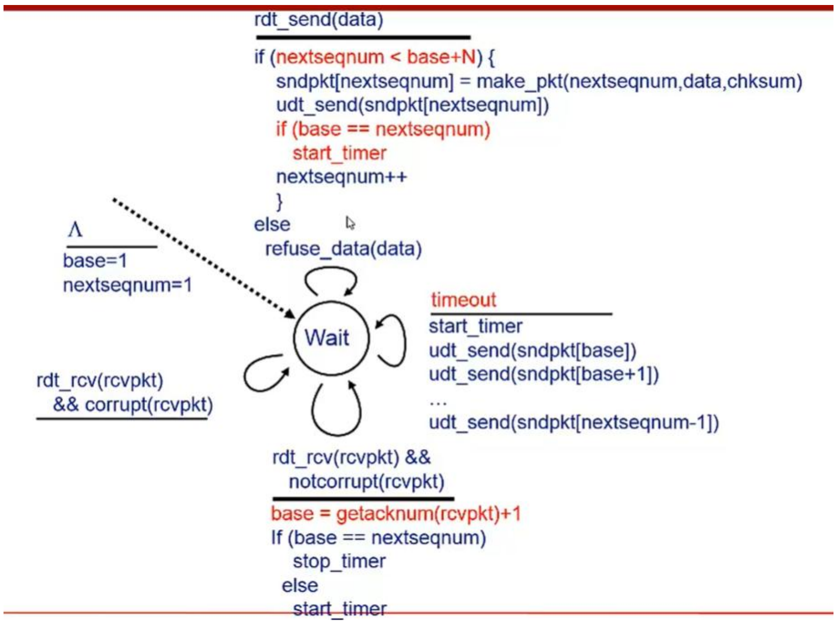
实验内容：

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。
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) 改进所设计的 GBN 协议，支持双向数据传输；
7) 将所设计的 GBN 协议改进为 SR 协议。

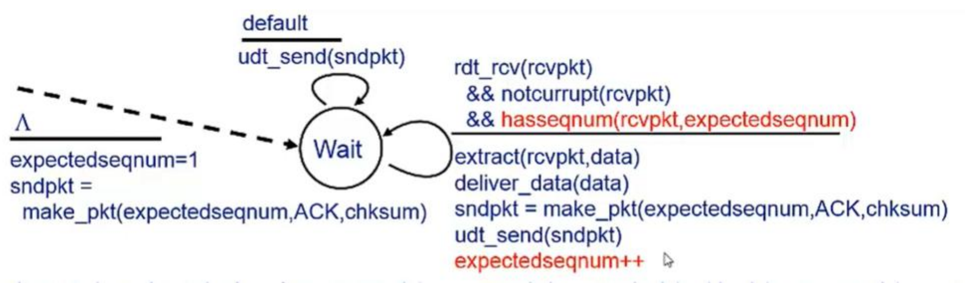
实验过程：

首先实现GBN：

先对GBN协议进行一些分析，其发送方FSM如下：



接收方FSM如下

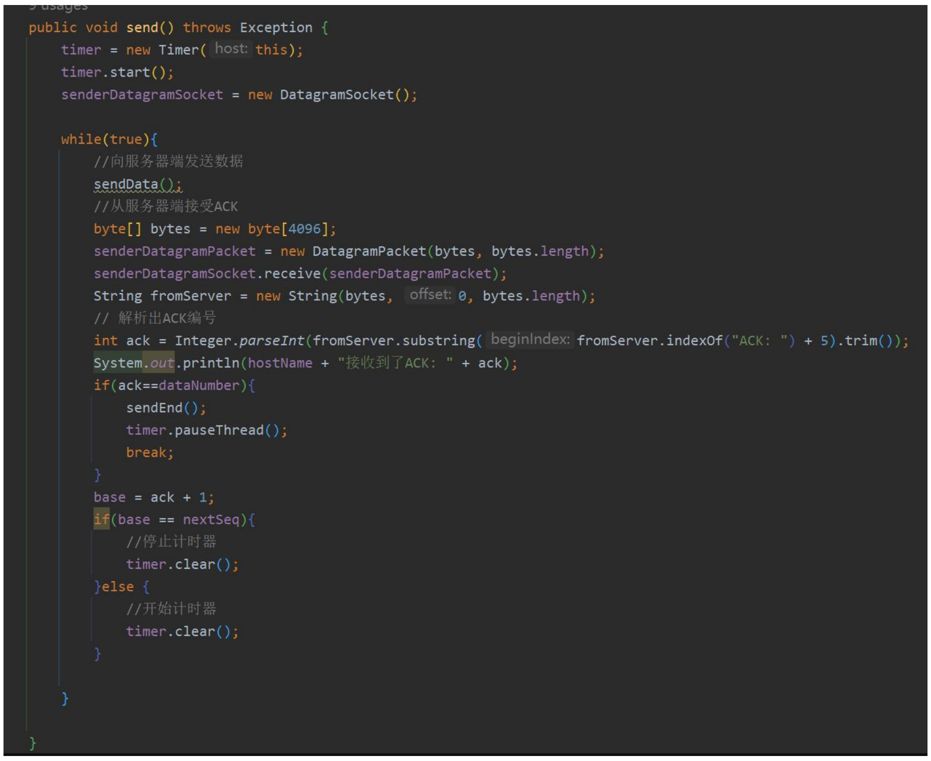


基于这两个FSM可以较为简单的实现GBN协议。

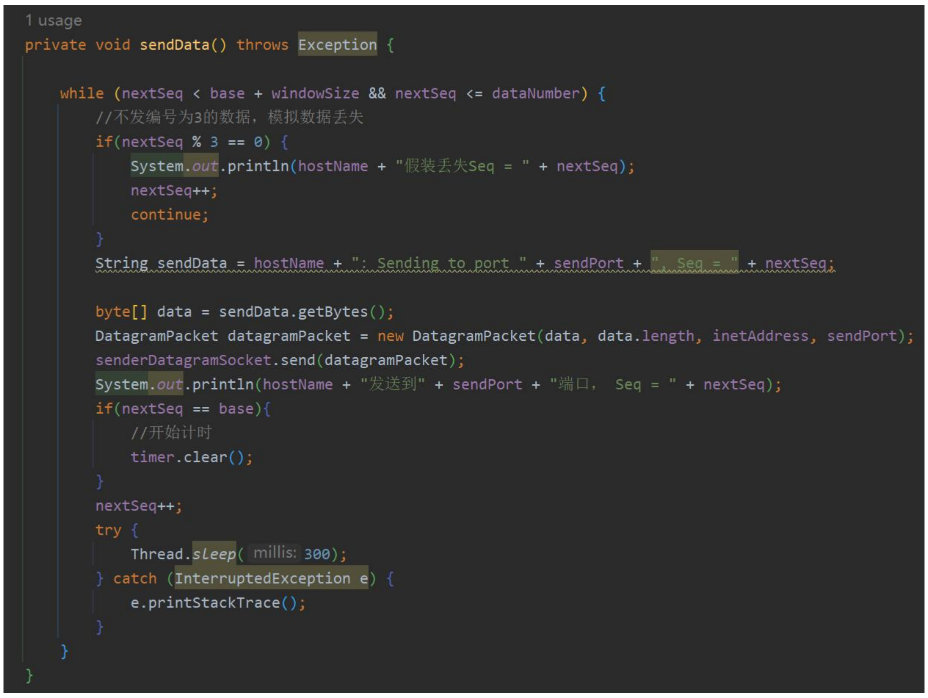
如下是GBN的接受方代码，其代码结构与FSM是一致的，首先发送数据，将窗口内的

数据全部发出。然后等待接受来自接受放的ACK，并不断重复这一过程，如果对某一个分组的ACK等待超时，就重发窗口内的所有数据。

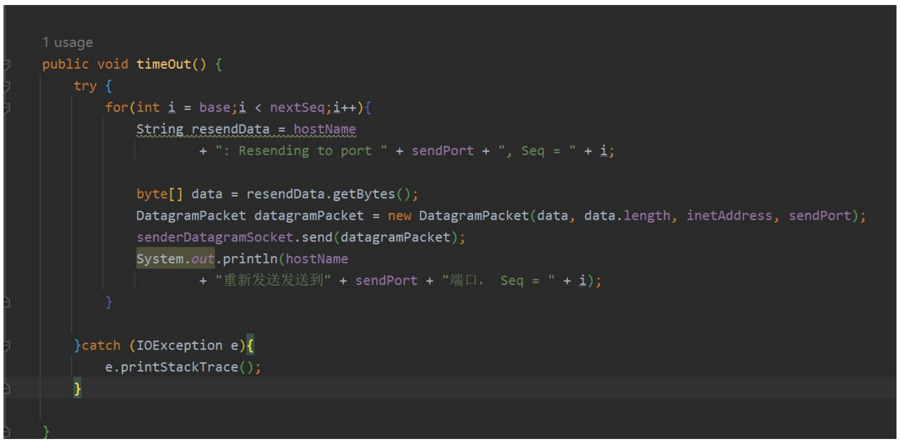
另外，可以看到，当数据包序列号为3的倍数时，发送方模拟丢包。



```java
public void send() throws Exception {
    timer = new Timer( host: this);
    timer.start();
    senderDatagramSocket = new DatagramSocket();

    while(true){
        //向服务器端发送数据
        sendData();
        //从服务器端接受ACK
        byte[] bytes = new byte[4096];
        senderDatagramPacket = new DatagramPacket(bytes, bytes.length);
        senderDatagramSocket.receive(senderDatagramPacket);
        String fromServer = new String(bytes,  offset: 0, bytes.length);
        // 解析出ACK编号
        int ack = Integer.parseInt(fromServer.substring( beginIndex: fromServer.indexOf("ACK: ") + 5).trim());
        System.out.println(hostName + "接收到了ACK: " + ack);
        if(ack==dataNumber){
            sendEnd();
            timer.pauseThread();
            break;
        }
        base = ack + 1;
        if(base == nextSeq){
            //停止计时器
            timer.clear();
        }else {
            //开始计时器
            timer.clear();
        }

    }

}
```

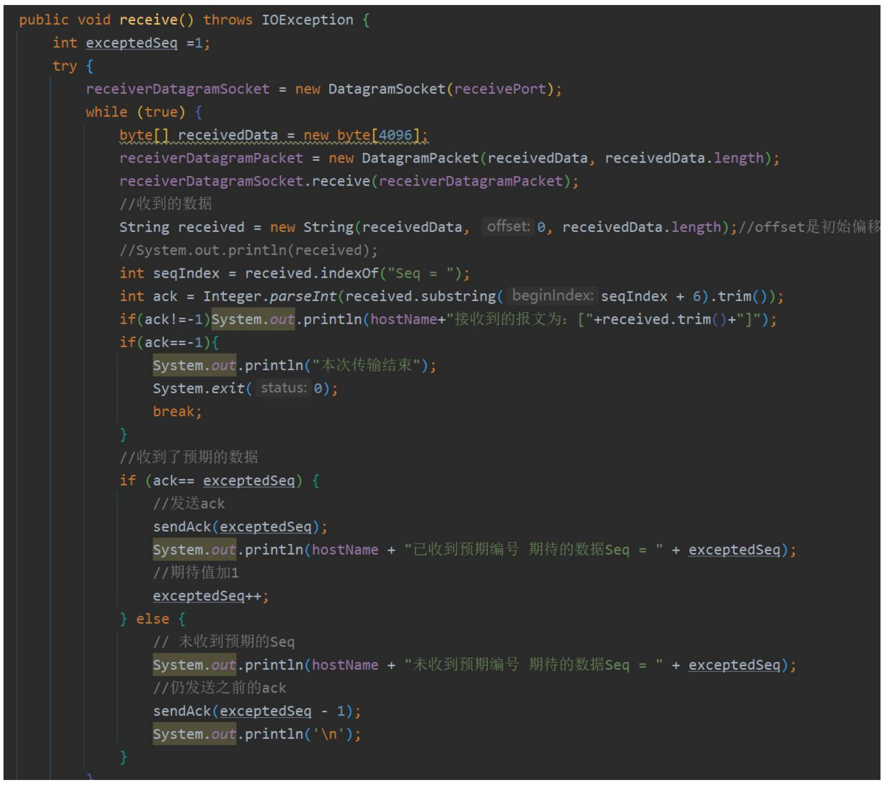


```java
1 usage
private void sendData() throws Exception {

    while (nextSeq < base + windowSize && nextSeq <= dataNumber) {
        //不发编号为3的数据，模拟数据丢失
        if(nextSeq % 3 == 0) {
            System.out.println(hostName + "假装丢失Seq = " + nextSeq);
            nextSeq++;
            continue;
        }
        String sendData = hostName + ": Sending to port " + sendPort + ", Seq = " + nextSeq;

        byte[] data = sendData.getBytes();
        DatagramPacket datagramPacket = new DatagramPacket(data, data.length, inetAddress, sendPort);
        senderDatagramSocket.send(datagramPacket);
        System.out.println(hostName + "发送到" + sendPort + "端口， Seq = " + nextSeq);
        if(nextSeq == base){
            //开始计时
            timer.clear();
        }
        nextSeq++;
        try {
            Thread.sleep( millis: 300);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```



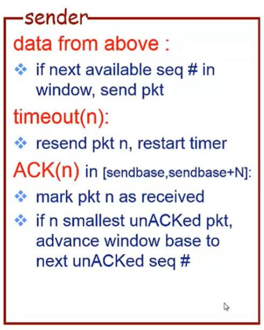
```
1 usage
public void timeOut() {
    try {
        for(int i = base;i < nextSeq;i++){
            String resendData = hostName
                    + ": Resending to port " + sendPort + ", Seq = " + i;

            byte[] data = resendData.getBytes();
            DatagramPacket datagramPacket = new DatagramPacket(data, data.length, inetAddress, sendPort);
            senderDatagramSocket.send(datagramPacket);
            System.out.println(hostName
                    + "重新发送发送到" + sendPort + "端口， Seq = " + i);
        }

    }catch (IOException e){
        e.printStackTrace();
    }

}
```

以下是GBN协议的接受方代码，同样的，其逻辑与FSM中是一致的。
当接收到的数据不是期望的时，发送方上一条ACK消息，并将数据包丢弃。
若收到期望的数据包，发送对应ACK并留下数据。



```
public void receive() throws IOException {
    int exceptedSeq =1;
    try {
        receiverDatagramSocket = new DatagramSocket(receivePort);
        while (true) {
            byte[] receivedData = new byte[4096];
            receiverDatagramPacket = new DatagramPacket(receivedData, receivedData.length);
            receiverDatagramSocket.receive(receiverDatagramPacket);
            //收到的数据
            String received = new String(receivedData,  offset: 0, receivedData.length);//offset是初始偏移
            //System.out.println(received);
            int seqIndex = received.indexOf("Seq = ");
            int ack = Integer.parseInt(received.substring( beginIndex: seqIndex + 6).trim());
            if(ack!=-1)System.out.println(hostName+"接收到的报文为：["+received.trim()+"]");
            if(ack==-1){
                System.out.println("本次传输结束");
                System.exit( status: 0);
                break;
            }
            //收到了预期的数据
            if (ack== exceptedSeq) {
                //发送ack
                sendAck(exceptedSeq);
                System.out.println(hostName + "已收到预期编号 期待的数据Seq = " + exceptedSeq);
                //期待值加1
                exceptedSeq++;
            } else {
                // 未收到预期的Seq
                System.out.println(hostName + "未收到预期编号 期待的数据Seq = " + exceptedSeq);
                //仍发送之前的ack
                sendAck(exceptedSeq - 1);
                System.out.println('\n');
            }
```

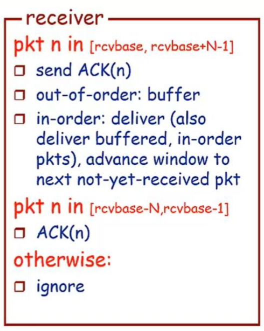
接下来实现SR协议，它和GBN协议是极为相似的，但是当超时重传时，只发送对应的未收到ACK的数据。



sender
data from above :
- if next available seq # in window, send pkt

timeout(n):
- resend pkt n, restart timer

ACK(n) in [sendbase,sendbase+N]:
- mark pkt n as received
- if n smallest unACKed pkt, advance window base to next unACKed seq #



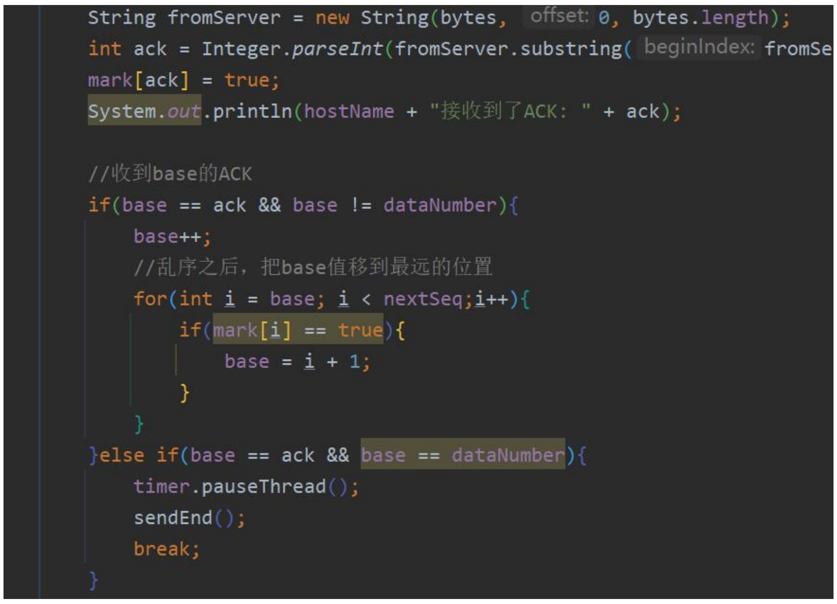
receiver
pkt n in [rcvbase, rcvbase+N-1]
- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

pkt n in [rcvbase-N,rcvbase-1]
- ACK(n)

otherwise:
- ignore

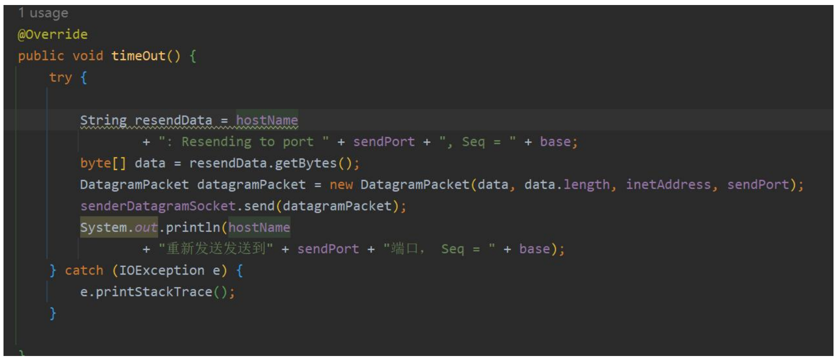
因为GBN和SR的相似性，这里只展示一些区别较大的代码。

在发送方中，收到ACK中将base移到最远的位置。



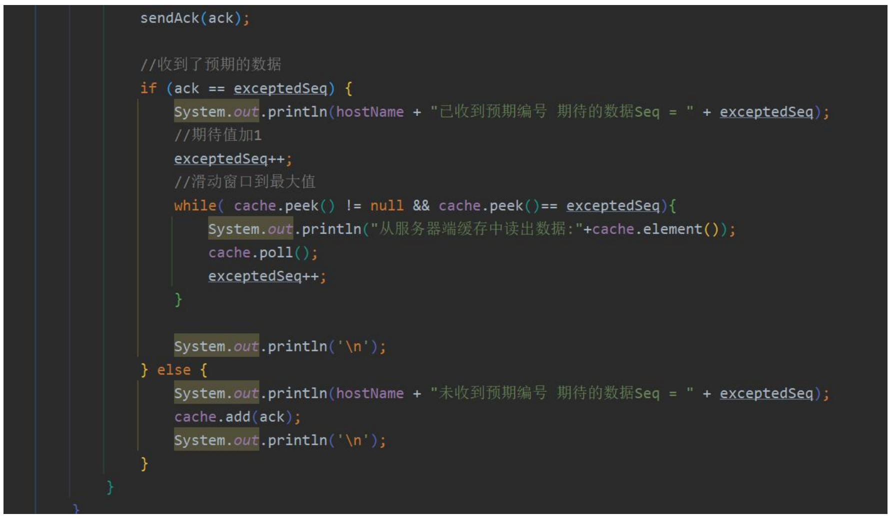
```
String fromServer = new String(bytes,  offset: 0, bytes.length);
int ack = Integer.parseInt(fromServer.substring( beginIndex: fromSe
mark[ack] = true;
System.out.println(hostName + "接收到了ACK: " + ack);

//收到base的ACK
if(base == ack && base != dataNumber){
    base++;
    //乱序之后，把base值移到最远的位置
    for(int i = base; i < nextSeq;i++){
        if(mark[i] == true){
            base = i + 1;
        }
    }
}else if(base == ack && base == dataNumber){
    timer.pauseThread();
    sendEnd();
    break;
}
```

超时重传时只重传对应数据。



```
1 usage
@Override
public void timeOut() {
    try {

        String resendData = hostName
                + ": Resending to port " + sendPort + ", Seq = " + base;
        byte[] data = resendData.getBytes();
        DatagramPacket datagramPacket = new DatagramPacket(data, data.length, inetAddress, sendPort);
        senderDatagramSocket.send(datagramPacket);
        System.out.println(hostName
                + "重新发送发送到" + sendPort + "端口，  Seq = " + base);
    } catch (IOException e) {
        e.printStackTrace();
    }
```

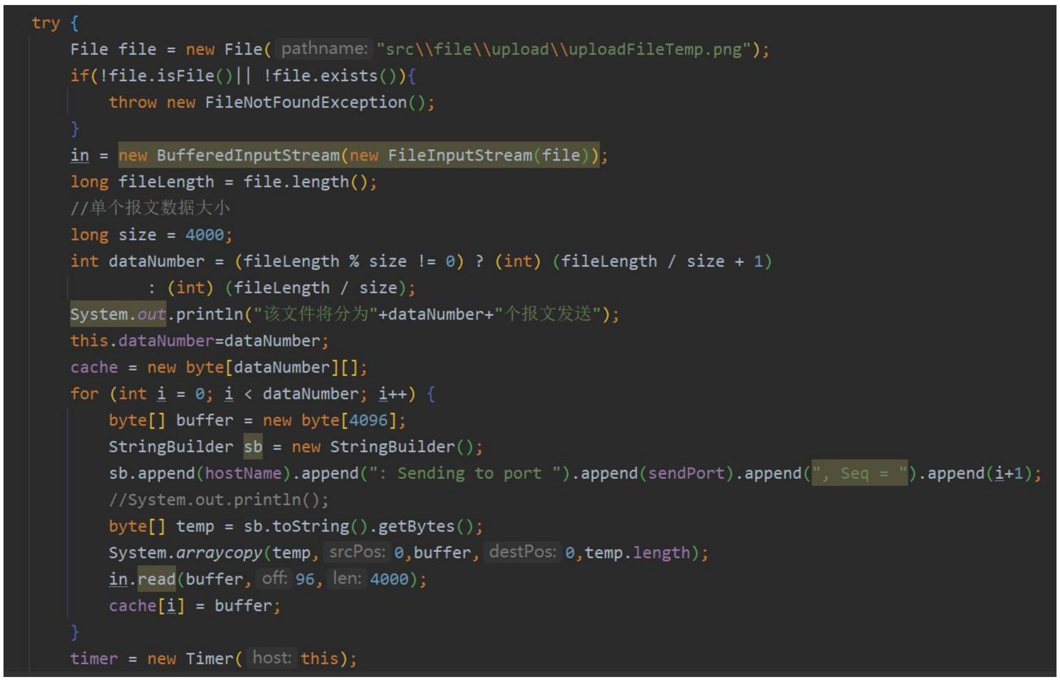
在接收方中，收到数据包时，滑动窗口到最大值（从缓存中读出数据）。



```
            sendAck(ack);

            //收到了预期的数据
            if (ack == exceptedSeq) {
                System.out.println(hostName + "已收到预期编号 期待的数据Seq = " + exceptedSeq);
                //期待值加1
                exceptedSeq++;
                //滑动窗口到最大值
                while( cache.peek() != null && cache.peek()== exceptedSeq){
                    System.out.println("从服务器端缓存中读出数据:"+cache.element());
                    cache.poll();
                    exceptedSeq++;
                }

                System.out.println('\n');
            } else {
                System.out.println(hostName + "未收到预期编号 期待的数据Seq = " + exceptedSeq);
                cache.add(ack);
                System.out.println('\n');
            }
        }
```

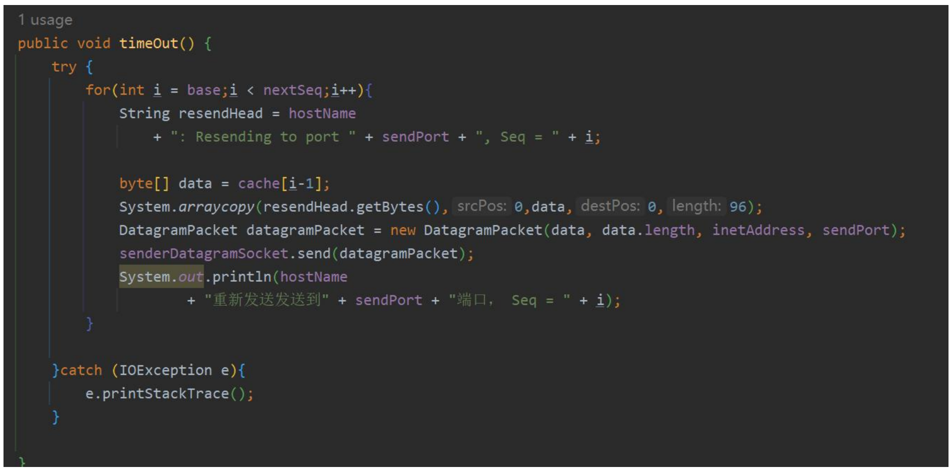
接下来是关于文件功能的实现，这里是基于GBN协议实现的，所以同样只展示不同的地方。

对与发送方，在发送数据前，需要读入文件并将其分割并封装到报文中。



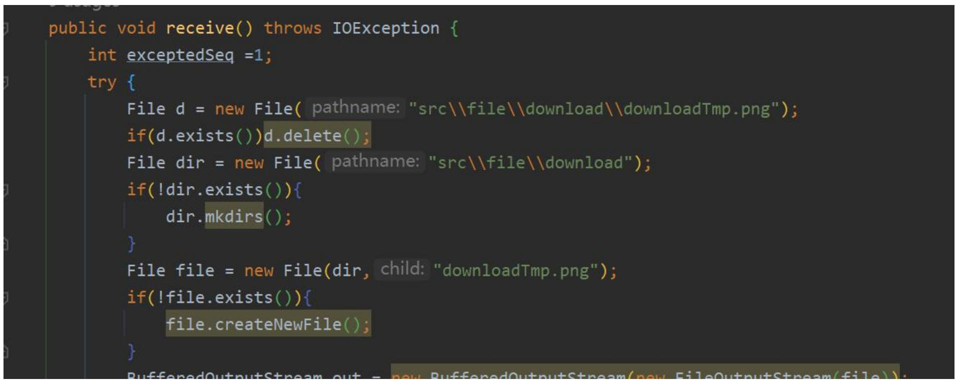
```
try {
    File file = new File( pathname: "src\\file\\upload\\uploadFileTemp.png");
    if(!file.isFile()|| !file.exists()){
        throw new FileNotFoundException();
    }
    in = new BufferedInputStream(new FileInputStream(file));
    long fileLength = file.length();
    //单个报文数据大小
    long size = 4000;
    int dataNumber = (fileLength % size != 0) ? (int) (fileLength / size + 1)
            : (int) (fileLength / size);
    System.out.println("该文件将分为"+dataNumber+"个报文发送");
    this.dataNumber=dataNumber;
    cache = new byte[dataNumber][];
    for (int i = 0; i < dataNumber; i++) {
        byte[] buffer = new byte[4096];
        StringBuilder sb = new StringBuilder();
        sb.append(hostName).append(": Sending to port ").append(sendPort).append(", Seq = ").append(i+1);
        //System.out.println();
        byte[] temp = sb.toString().getBytes();
        System.arraycopy(temp, srcPos: 0,buffer, destPos: 0,temp.length);
        in.read(buffer, off: 96, len: 4000);
        cache[i] = buffer;
    }
    timer = new Timer( host: this);
```

重发过程中，同样需要从缓存中读取数据并发送。



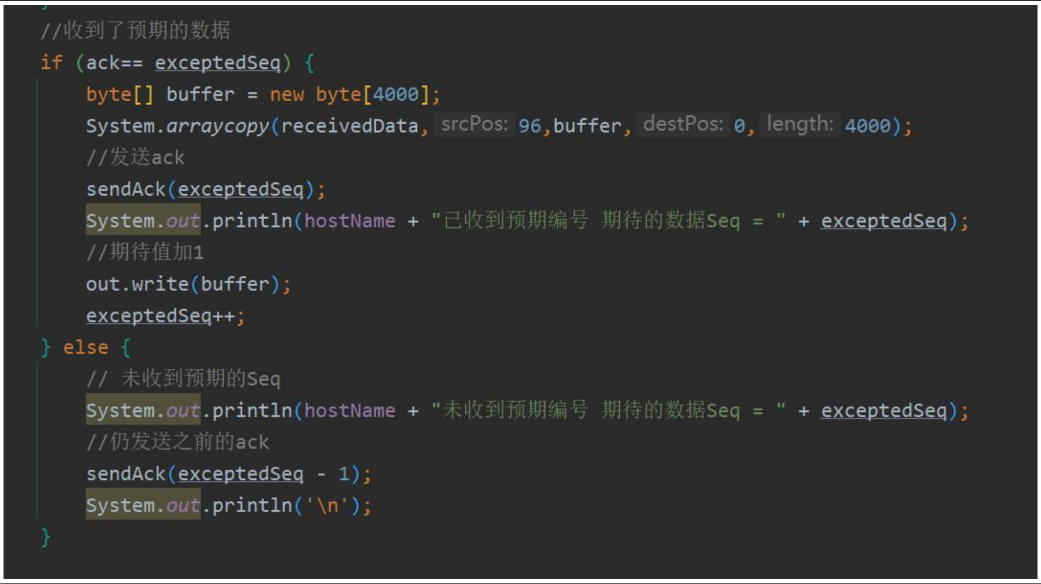
```
1 usage
public void timeOut() {
    try {
        for(int i = base;i < nextSeq;i++){
            String resendHead = hostName
                + ": Resending to port " + sendPort + ", Seq = " + i;

            byte[] data = cache[i-1];
            System.arraycopy(resendHead.getBytes(), srcPos: 0,data, destPos: 0, length: 96);
            DatagramPacket datagramPacket = new DatagramPacket(data, data.length, inetAddress, sendPort);
            senderDatagramSocket.send(datagramPacket);
            System.out.println(hostName
                    + "重新发送发送到" + sendPort + "端口， Seq = " + i);
        }

    }catch (IOException e){
        e.printStackTrace();
    }
}
```

对于接收方，在正式接受文件前需要打开一个文件输出流，



```
public void receive() throws IOException {
    int exceptedSeq =1;
    try {
        File d = new File( pathname: "src\\file\\download\\downloadTmp.png");
        if(d.exists())d.delete();
        File dir = new File( pathname: "src\\file\\download");
        if(!dir.exists()){
            dir.mkdirs();
        }
        File file = new File(dir, child: "downloadTmp.png");
        if(!file.exists()){
            file.createNewFile();
        }
        BufferedOutputStream out = new BufferedOutputStream(new FileOutputStream(file));
```

并在接受到正确序列号的分组时，将其中封装的数据写入到文件中。



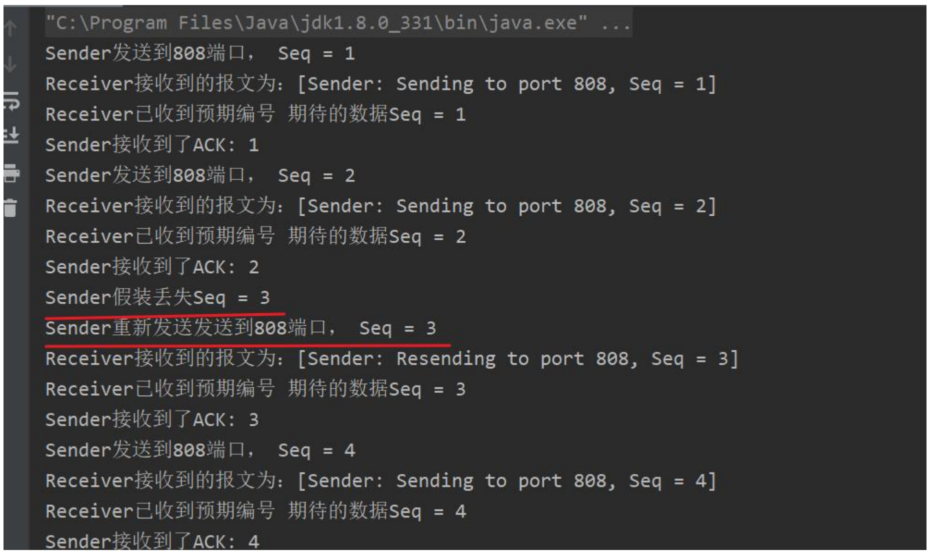
```
//收到了预期的数据
if (ack== exceptedSeq) {
    byte[] buffer = new byte[4000];
    System.arraycopy(receivedData, srcPos: 96,buffer, destPos: 0, length: 4000);
    //发送ack
    sendAck(exceptedSeq);
    System.out.println(hostName + "已收到预期编号 期待的数据Seq = " + exceptedSeq);
    //期待值加1
    out.write(buffer);
    exceptedSeq++;
} else {
    // 未收到预期的Seq
    System.out.println(hostName + "未收到预期编号 期待的数据Seq = " + exceptedSeq);
    //仍发送之前的ack
    sendAck(exceptedSeq - 1);
    System.out.println('\n');
}
```

实验结果：

停等协议实现结果：

只要将GBN协议的窗口大小设置为1即可

在发送过程中可以看到，发送方的数据丢失后，在等待一段时间未收到对应ACK后即会重传。



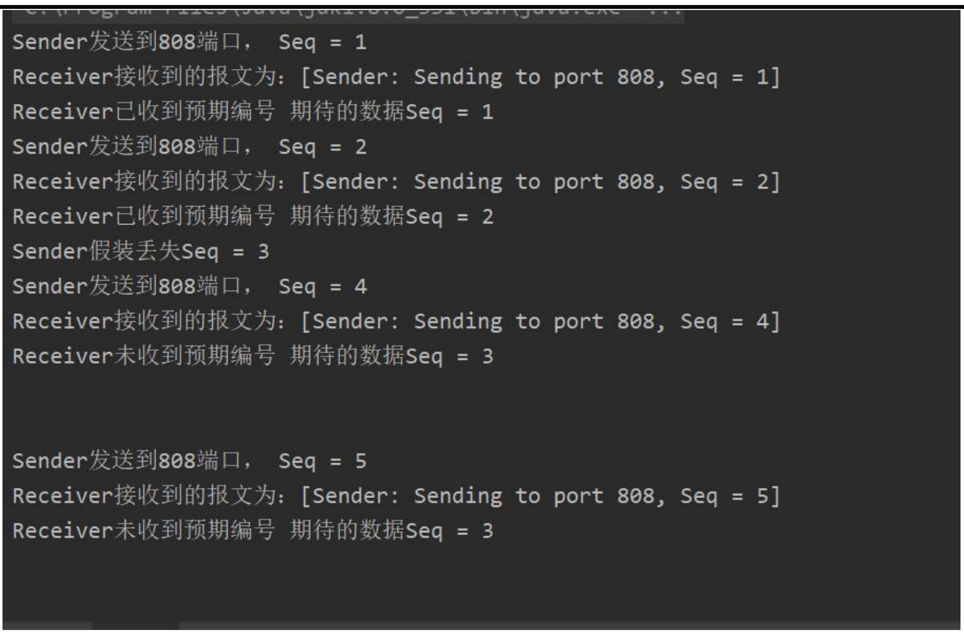
```
"C:\Program Files\Java\jdk1.8.0_331\bin\java.exe" ...
Sender发送到808端口，  Seq = 1
Receiver接收到的报文为: [Sender: Sending to port 808, Seq = 1]
Receiver已收到预期编号 期待的数据Seq = 1
Sender接收到了ACK: 1
Sender发送到808端口，  Seq = 2
Receiver接收到的报文为: [Sender: Sending to port 808, Seq = 2]
Receiver已收到预期编号 期待的数据Seq = 2
Sender接收到了ACK: 2
Sender假装丢失Seq = 3
Sender重新发送发送到808端口，  Seq = 3
Receiver接收到的报文为: [Sender: Resending to port 808, Seq = 3]
Receiver已收到预期编号 期待的数据Seq = 3
Sender接收到了ACK: 3
Sender发送到808端口，  Seq = 4
Receiver接收到的报文为: [Sender: Sending to port 808, Seq = 4]
Receiver已收到预期编号 期待的数据Seq = 4
Sender接收到了ACK: 4
```

GBN协议实现结果：

这里设定发送窗口大小为5，发送数据量为20.

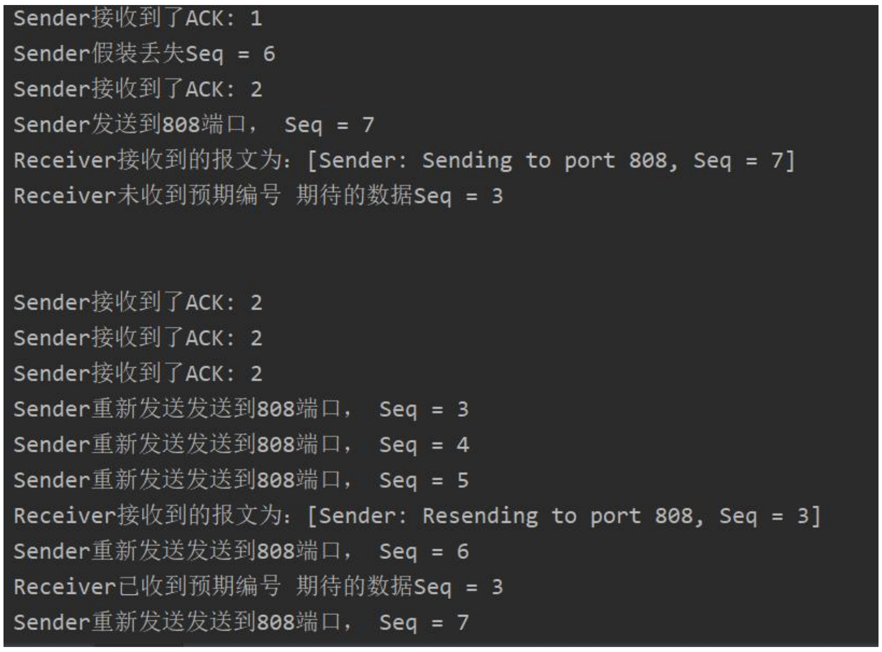
首先可以看到，发送方发送了1-5（其中3被模拟丢失）

且在接受方收到4-5后期待的数据为3



```
Sender发送到808端口， Seq = 1
Receiver接收到的报文为：[Sender: Sending to port 808, Seq = 1]
Receiver已收到预期编号 期待的数据Seq = 1
Sender发送到808端口， Seq = 2
Receiver接收到的报文为：[Sender: Sending to port 808, Seq = 2]
Receiver已收到预期编号 期待的数据Seq = 2
Sender假装丢失Seq = 3
Sender发送到808端口， Seq = 4
Receiver接收到的报文为：[Sender: Sending to port 808, Seq = 4]
Receiver未收到预期编号 期待的数据Seq = 3


Sender发送到808端口， Seq = 5
Receiver接收到的报文为：[Sender: Sending to port 808, Seq = 5]
Receiver未收到预期编号 期待的数据Seq = 3
```

然后当发送方接收到ACK1后窗口移动，发送了数据包6，之后收到了3个重复的ACK2，这是接收方在收到乱序数据后发送的ACK，之后发送方在超时后，重新发送了窗口中所有数据3-7。这与协议规定的是相同的。

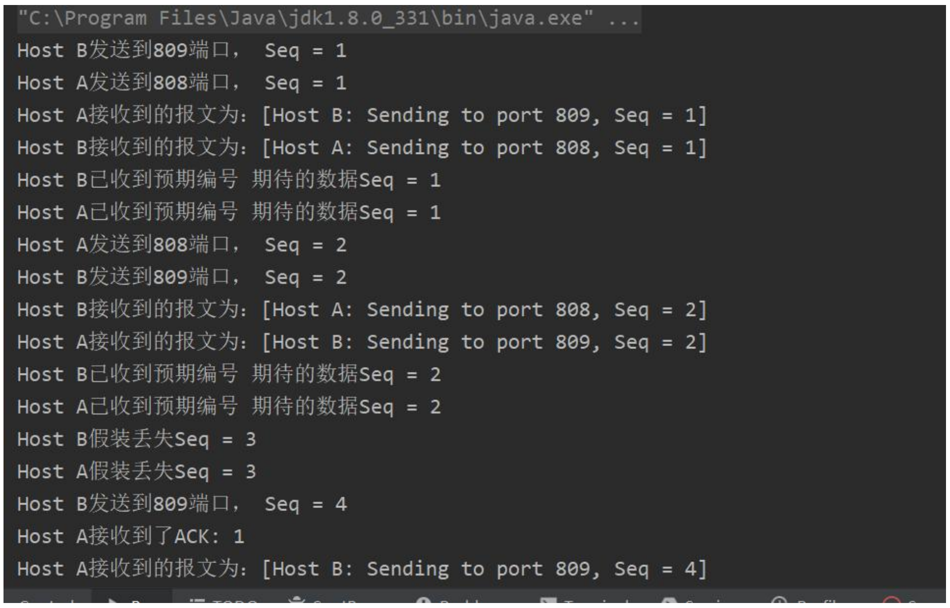


```
Sender接收到了ACK: 1
Sender假装丢失Seq = 6
Sender接收到了ACK: 2
Sender发送到808端口， Seq = 7
Receiver接收到的报文为：[Sender: Sending to port 808, Seq = 7]
Receiver未收到预期编号 期待的数据Seq = 3


Sender接收到了ACK: 2
Sender接收到了ACK: 2
Sender接收到了ACK: 2
Sender重新发送发送到808端口， Seq = 3
Sender重新发送发送到808端口， Seq = 4
Sender重新发送发送到808端口， Seq = 5
Receiver接收到的报文为：[Sender: Resending to port 808, Seq = 3]
Sender重新发送发送到808端口， Seq = 6
Receiver已收到预期编号 期待的数据Seq = 3
Sender重新发送发送到808端口， Seq = 7
```

双向数据传输：

前面的单项数据传输是用两个线程来是实现的，一个用来发送数据，一个用来接受数据，那么就可以用四个线程来实现双向数据传输。

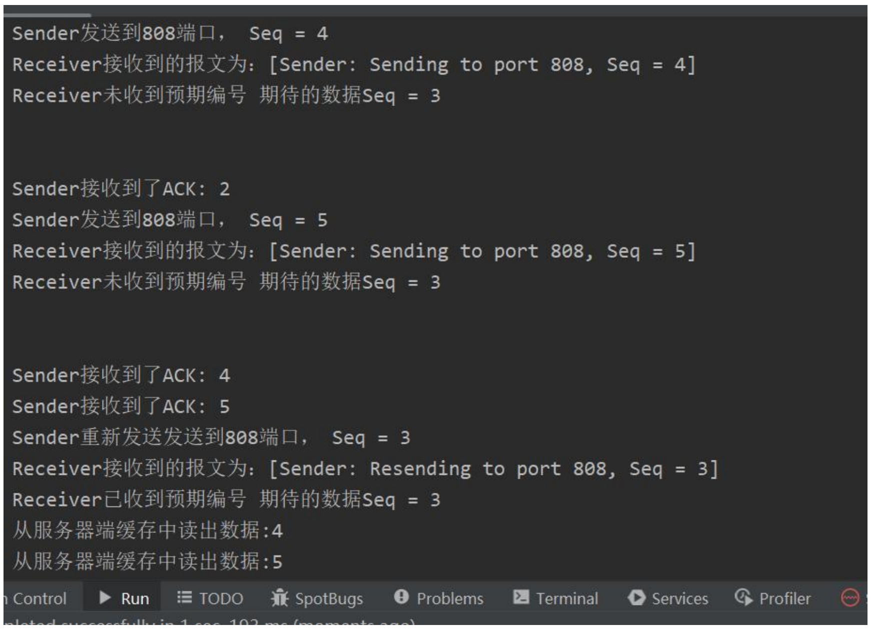
这里就不进行仔细讲解了，可以看到如图确实实现了双向数据传输。



```
"C:\Program Files\Java\jdk1.8.0_331\bin\java.exe" ...
Host B发送到809端口，  Seq = 1
Host A发送到808端口，  Seq = 1
Host A接收到的报文为：[Host B: Sending to port 809, Seq = 1]
Host B接收到的报文为：[Host A: Sending to port 808, Seq = 1]
Host B已收到预期编号 期待的数据Seq = 1
Host A已收到预期编号 期待的数据Seq = 1
Host A发送到808端口，  Seq = 2
Host B发送到809端口，  Seq = 2
Host B接收到的报文为：[Host A: Sending to port 808, Seq = 2]
Host A接收到的报文为：[Host B: Sending to port 809, Seq = 2]
Host B已收到预期编号 期待的数据Seq = 2
Host A已收到预期编号 期待的数据Seq = 2
Host B假装丢失Seq = 3
Host A假装丢失Seq = 3
Host B发送到809端口，  Seq = 4
Host A接收到了ACK: 1
Host A接收到的报文为：[Host B: Sending to port 809, Seq = 4]
```

SR协议实现

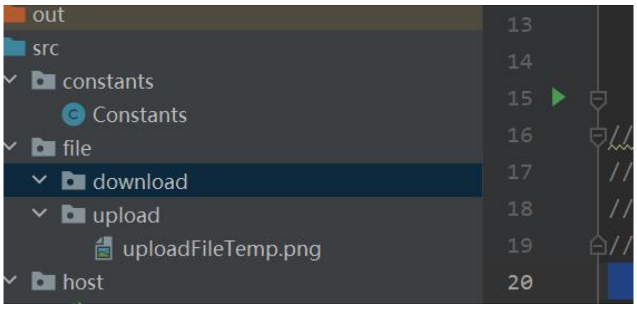
与GBN协议一样，这里也采用3的倍数模拟丢包，可以看到接受方在接收到4-5时，期待的数据还是3，当发送方重新发送了3后，接收方会从缓存中读取能够到达的最大窗口基数。



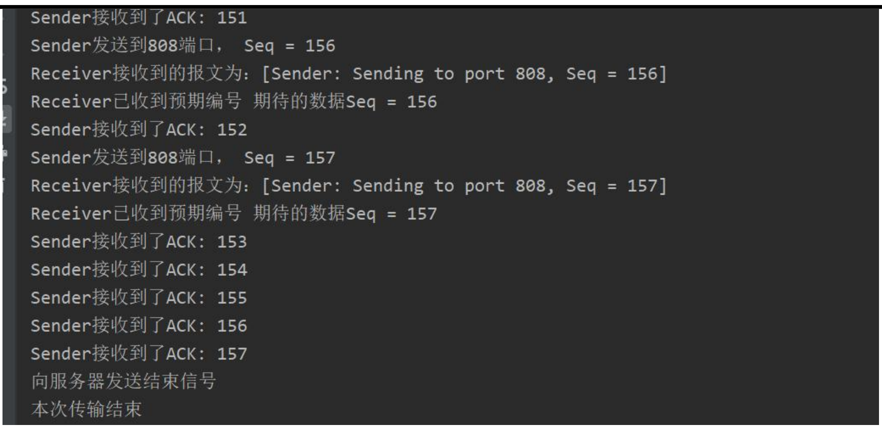
```
Sender发送到808端口，  Seq = 4
Receiver接收到的报文为：[Sender: Sending to port 808, Seq = 4]
Receiver未收到预期编号 期待的数据Seq = 3


Sender接收到了ACK: 2
Sender发送到808端口，  Seq = 5
Receiver接收到的报文为：[Sender: Sending to port 808, Seq = 5]
Receiver未收到预期编号 期待的数据Seq = 3


Sender接收到了ACK: 4
Sender接收到了ACK: 5
Sender重新发送发送到808端口，  Seq = 3
Receiver接收到的报文为：[Sender: Resending to port 808, Seq = 3]
Receiver已收到预期编号 期待的数据Seq = 3
从服务器端缓存中读出数据:4
从服务器端缓存中读出数据:5
```

文件传输实现：

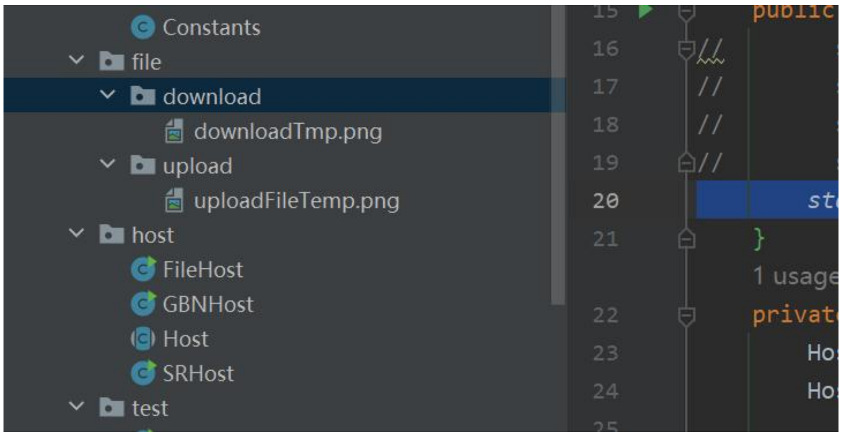
可以看到在运行前，目录download下是没有文件的。



在运行后可以看到多了一个文件且其内容与uploadFileTemp.png是相同的。



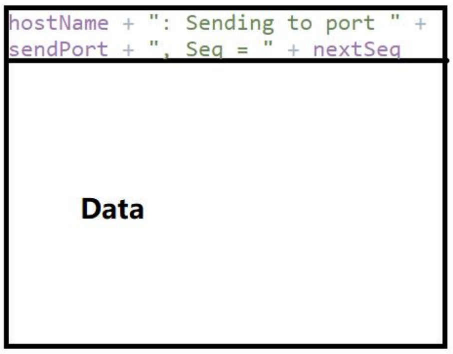
```
Sender接收到了ACK: 151
Sender发送到808端口， Seq = 156
Receiver接收到的报文为：[Sender: Sending to port 808, Seq = 156]
Receiver已收到预期编号 期待的数据Seq = 156
Sender接收到了ACK: 152
Sender发送到808端口， Seq = 157
Receiver接收到的报文为：[Sender: Sending to port 808, Seq = 157]
Receiver已收到预期编号 期待的数据Seq = 157
Sender接收到了ACK: 153
Sender接收到了ACK: 154
Sender接收到了ACK: 155
Sender接收到了ACK: 156
Sender接收到了ACK: 157
向服务器发送结束信号
本次传输结束
```



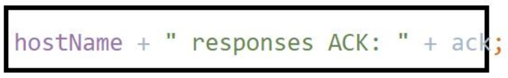
问题讨论：

在实验报告中要说明所设计 GBN 协议数据分组格式、确认分组格式、各个域作用程序实现的主要类（或函数）及其主要作用

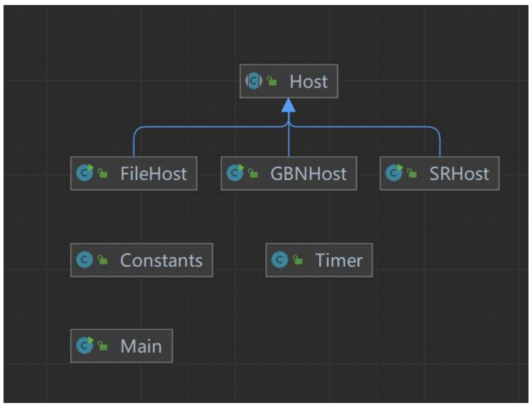
数据分组格式为


hostName + ": Sending to port " +
sendPort + ", Seq = " + nextSeq
Data

确认分组格式为



程序的主要类如下



Host类是一个抽象的类，作为所有主机类的父类，规定一些必要的接口，并方便Timer的实现。
GBNHost封装了GBN协议的发送方法和接受方法。
SRHost封装了SR协议的发送方法和接受方法。
FileHost改造了GBN协议的发送方法和接受方法，使其能够发送和接受文件。
Timer类是计时器类。
Constant作为公共常量类，存放一些常数。
Main为测试类，其中放一些测试用的方法。

心得体会：

深刻理解了GBN和SR协议的过程。
熟悉了Java中关于UDP的socket编程。
熟悉了java多线程编程，实现了一个计时器。

附录：程序源代码

Host.java

```
package host;

import util.timer.Timer;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.UnknownHostException;

/**
 * Host
 *
 * @author: lsxuan
 * @email: 1146887979@qq.com
 * @create: 2022-10-09 12:02
 */
```

```
public abstract class Host {
        protected String hostName;
        protected int sendPort;
        protected int receivePort;
        protected InetAddress inetAddress;
        protected DatagramSocket senderDatagramSocket;
        protected DatagramPacket senderDatagramPacket;
        protected DatagramSocket receiverDatagramSocket;
        protected DatagramPacket receiverDatagramPacket;
        protected int nextSeq;
        protected int base;
        protected int windowSize;
        protected int dataNumber;
        protected Timer timer;

        /**
         *  发送数据
         * @throws Exception
         */
        abstract public void send() throws Exception;
        /**
         *  超时数据重传
         */
        abstract public void timeOut();

        /**
         *  接受数据
         * @throws IOException
         */
        abstract public void receive() throws IOException ;

        /**
         *  发送 ack
         * @param ack
         * @throws IOException
         */
        abstract public void sendAck(int ack) throws IOException;

        /**
         *  发送结束信息
         * @throws IOException
         */
        abstract public void sendEnd() throws IOException;
```

```
    public Host(String hostName,int sendPort,int receivePort,int windowSize,int dataNumber) {
        try {
            this.hostName = hostName;
            inetAddress = InetAddress.getLocalHost();
            nextSeq=1;
            base=1;
            this.windowSize = windowSize;
            this.dataNumber=dataNumber;
            this.sendPort = sendPort;
            this.receivePort = receivePort;
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
    }
}
```

GBNHost.java

```
package host;

import util.timer.Timer;

import java.io.*;
import java.net.*;

/**
 * GBNHost
 */
public class GBNHost extends Host {


    public GBNHost(String name,int sendPort,int receivePort,int windowSize,int dataNumber){
        super(name,sendPort,receivePort,windowSize,dataNumber);
    }

    public void send() throws Exception {
        timer = new Timer(this);
        timer.start();
        senderDatagramSocket = new DatagramSocket();

        while(true){
            //向服务器端发送数据
            sendData();
            //从服务器端接受 ACK
            byte[] bytes = new byte[4096];
            senderDatagramPacket = new DatagramPacket(bytes, bytes.length);
```

```
                senderDatagramSocket.receive(senderDatagramPacket);
                String fromServer = new String(bytes, 0, bytes.length);
                // 解析出 ACK 编号
                int ack = Integer.parseInt(fromServer.substring(fromServer.indexOf("ACK: ") +
5).trim());
                System.out.println(hostName + "接收到了 ACK: " + ack);
                if(ack==dataNumber){
                    sendEnd();
                    timer.pauseThread();
                    break;
                }
                base = ack + 1;
                if(base == nextSeq){
                    //停止计时器
                    timer.clear();
                }else {
                    //开始计时器
                    timer.clear();
                }

            }

        }



        /**
         * 向服务器发送数据
         *
         * @throws Exception
         */
        private void sendData() throws Exception {

            while (nextSeq < base + windowSize && nextSeq <= dataNumber) {
                //不发编号为 3 的数据，模拟数据丢失
                if(nextSeq % 3 == 0) {
                    System.out.println(hostName + "假装丢失 Seq = " + nextSeq);
                    nextSeq++;
                    continue;
                }
                String sendData = hostName + ": Sending to port " + sendPort + ", Seq = " +
nextSeq;

                byte[] data = sendData.getBytes();
                DatagramPacket datagramPacket = new DatagramPacket(data, data.length,
inetAddress, sendPort);
                senderDatagramSocket.send(datagramPacket);
                System.out.println(hostName + "发送到" + sendPort + "端口，  Seq = " +
nextSeq);
                if(nextSeq == base){
                    //开始计时
                    timer.clear();
                }
                nextSeq++;
                try {
                    Thread.sleep(300);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }


        public void timeOut() {
            try {
                for(int i = base;i < nextSeq;i++){
                    String resendData = hostName
                            + ": Resending to port " + sendPort + ", Seq = " + i;

                    byte[] data = resendData.getBytes();
                    DatagramPacket datagramPacket = new DatagramPacket(data, data.length,
inetAddress, sendPort);
                    senderDatagramSocket.send(datagramPacket);
                    System.out.println(hostName
                            + "重新发送发送到" + sendPort + "端口，  Seq = " + i);
                }

            }catch (IOException e){
                e.printStackTrace();
            }

        }


        public void receive() throws IOException {
            int exceptedSeq =1;
            try {
                receiverDatagramSocket = new DatagramSocket(receivePort);
                while (true) {
                    byte[] receivedData = new byte[4096];
                    receiverDatagramPacket = new DatagramPacket(receivedData,
receivedData.length);
                    receiverDatagramSocket.receive(receiverDatagramPacket);
                    //收到的数据
                    String received = new String(receivedData, 0, receivedData.length);//offset 是
初始偏移量
                    //System.out.println(received);
                    int seqIndex = received.indexOf("Seq = ");
                    int ack = Integer.parseInt(received.substring(seqIndex + 6).trim());
                    if(ack!=-1)System.out.println(hostName+" 接 收 到 的 报 文 为 :
["+received.trim()+"]");
                    if(ack==-1){
                        System.out.println("本次传输结束");
                        System.exit(0);
                        break;
                    }
                    //收到了预期的数据
                    if (ack== exceptedSeq) {
                        //发送 ack
                        sendAck(exceptedSeq);
                        System.out.println(hostName + "已收到预期编号  期待的数据 Seq = "
+ exceptedSeq);
                        //期待值加 1
                        exceptedSeq++;
                    } else {
                        //  未收到预期的 Seq
                        System.out.println(hostName + "未收到预期编号  期待的数据 Seq = "
+ exceptedSeq);
                        //仍发送之前的 ack
                        sendAck(exceptedSeq - 1);
                        System.out.println('\n');
                    }
                }
            }catch(SocketException e){
                e.printStackTrace();
            }
        }

        @Override
        public void sendAck(int ack) throws IOException {
            String response = hostName + " responses ACK: " + ack;
            byte[] responseData = response.getBytes();
            InetAddress responseAddress = receiverDatagramPacket.getAddress();
```

```
            int responsePort = receiverDatagramPacket.getPort();
            receiverDatagramPacket                                    =                                    new
DatagramPacket(responseData,responseData.length,responseAddress,responsePort);
            receiverDatagramSocket.send(receiverDatagramPacket);
      }
      @Override
      public void sendEnd() throws IOException {
            inetAddress = InetAddress.getLocalHost();
            int end = -1;
            String clientData = hostName + ": Sending to port " + sendPort + ", Seq = " + end;
            System.out.println(hostName+"向服务器发送结束信号");

            byte[] data = clientData.getBytes();
            DatagramPacket datagramPacket = new DatagramPacket(data, data.length, inetAddress,
sendPort);
            senderDatagramSocket.send(datagramPacket);
      }
      public static void main(String[] args) throws InterruptedException {
            Host sender = new GBNHost("sender",33333,33334,1,20);
            Host receiver = new GBNHost("receiver",33334,33333,0,0);

            new Thread(new Runnable() {
                  @Override
                  public void run() {
                        try {
                              receiver.receive();
                        } catch (IOException e) {
                              throw new RuntimeException(e);
                        }
                  }
            }).start();
            Thread.sleep(1500);
            new Thread(new Runnable() {
                  @Override
                  public void run() {
                        try {
                              sender.send();
                        } catch (Exception e) {
                              throw new RuntimeException(e);
                        }
                  }
            }).start();
      }
}
```

SRHost.java

```
package host;

import util.timer.Timer;

import java.io.IOException;
import java.net.*;
import java.util.ArrayDeque;
import java.util.Queue;

public class SRHost extends Host {
    private boolean[] mark;
    //缓存队列
    private Queue<Integer> cache = new ArrayDeque<>();


    public SRHost(String name,int sendPort,int receivePort,int windowSize,int dataNumber) {
        super(name,sendPort,receivePort,windowSize,dataNumber);
    }

    @Override
    public void send() throws Exception {
        timer = new Timer(this);
        timer.start();
        senderDatagramSocket =new DatagramSocket();
        mark = new boolean[dataNumber +1];
        while(true){
            //向服务器端发送数据
            sendData();
            //从服务器端接受 ACK
            byte[] bytes = new byte[4096];
            senderDatagramPacket = new DatagramPacket(bytes, bytes.length);
            senderDatagramSocket.receive(senderDatagramPacket);

            String fromServer = new String(bytes, 0, bytes.length);
            int ack = Integer.parseInt(fromServer.substring(fromServer.indexOf("ACK: ") +
5).trim());
            mark[ack] = true;
            System.out.println(hostName + "接收到了 ACK: " + ack);

            //收到 base 的 ACK
            if(base == ack && base != dataNumber){
                base++;
                //乱序之后，把 base 值移到最远的位置
```

```
                    for(int i = base; i < nextSeq;i++){
                        if(mark[i] == true){
                            base = i + 1;
                        }
                    }
                }else if(base == ack && base == dataNumber){
                    timer.pauseThread();
                    sendEnd();
                    break;
                }

                if(base == nextSeq){
                    //停止计时器
                    timer.clear();
                }else {
                    //开始计时器
                    timer.clear();
                }
            }
        }
        /**
         * 向服务器发送数据
         *
         * @throws Exception
         */
        private void sendData() throws Exception {
            inetAddress = InetAddress.getLocalHost();
            while (nextSeq < base + windowSize && nextSeq <= dataNumber) {
                //不发编号为 3 的数据
                if(nextSeq == 3||nextSeq == 12) {
                    nextSeq++;
                    continue;
                }

                String sendData = hostName + ": Sending to port " + sendPort + ", Seq = " +
nextSeq;

                // 模拟发送分组
                byte[] data = sendData.getBytes();
                DatagramPacket datagramPacket = new DatagramPacket(data, data.length,
inetAddress, sendPort);
                senderDatagramSocket.send(datagramPacket);
                System.out.println(hostName + "发送到" + sendPort + "端口， Seq = " +
nextSeq);
```

```
            if(nextSeq == base){
                //开始计时
                //model.setTime(3);
                timer.clear();
            }
            nextSeq++;
            try {
                Thread.sleep(300);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public void timeOut() {
        try {

            String resendData = hostName
                    + ": Resending to port " + sendPort + ", Seq = " + base;
            byte[] data = resendData.getBytes();
            DatagramPacket datagramPacket = new DatagramPacket(data, data.length,
inetAddress, sendPort);
            senderDatagramSocket.send(datagramPacket);
            System.out.println(hostName
                    + "重新发送发送到" + sendPort + "端口，  Seq = " + base);
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
    /**
     * 向服务器发送结束信号
     */
    public void sendEnd() throws IOException {
        inetAddress = InetAddress.getLocalHost();
        int end = -1;
        String clientData = hostName + ": Sending to port " + sendPort + ", Seq = " + end;
        System.out.println("向服务器发送结束信号");

        byte[] data = clientData.getBytes();
        DatagramPacket datagramPacket = new DatagramPacket(data, data.length, inetAddress,
sendPort);
        senderDatagramSocket.send(datagramPacket);
```

```
	}
	@Override
	public void receive() throws IOException {
		int exceptedSeq =1;
		try {
			receiverDatagramSocket = new DatagramSocket(receivePort);
			while (true) {
				byte[] receivedData = new byte[4096];
				receiverDatagramPacket = new DatagramPacket(receivedData,
receivedData.length);
				receiverDatagramSocket.receive(receiverDatagramPacket);

				//收到的数据
				String received = new String(receivedData, 0, receivedData.length);//offset 是
初始偏移量
				//System.out.println(received);
				int seqIndex = received.indexOf("Seq = ");
				int ack = Integer.parseInt(received.substring(seqIndex + 6).trim());
				if(ack!=-1)System.out.println(hostName+" 接 收 到 的 报 文 为 :
["+received.trim()+"]");
				if(ack == -1){
					System.out.println("本次传输结束");
					System.exit(0);
					break;
				}else{
					sendAck(ack);

					//收到了预期的数据
					if (ack == exceptedSeq) {
						System.out.println(hostName + "已收到预期编号 期待的数据 Seq
= " + exceptedSeq);
						//期待值加 1
						exceptedSeq++;
						//滑动窗口到最大值
						while( cache.peek() != null && cache.peek()== exceptedSeq){
							System.out.println(" 从 服 务 器 端 缓 存 中 读 出 数
据:"+cache.element());
							cache.poll();
							exceptedSeq++;
						}

						System.out.println('\n');
					} else {
						System.out.println(hostName + "未收到预期编号 期待的数据 Seq
= " + exceptedSeq);
                        cache.add(ack);
                        System.out.println('\n');
                    }
                }
            }
        }catch(SocketException e){
            e.printStackTrace();
        }
    }
    //向客户端发送 ack
    public void sendAck(int ack) throws IOException {
        String response = hostName + " responses ACK: " + ack;
        byte[] responseData = response.getBytes();
        InetAddress responseAddress = receiverDatagramPacket.getAddress();
        int responsePort = receiverDatagramPacket.getPort();
        receiverDatagramPacket                              =                              new
DatagramPacket(responseData,responseData.length,responseAddress,responsePort);
        receiverDatagramSocket.send(receiverDatagramPacket);
    }
    public static void main(String[] args) throws InterruptedException {
        Host sender = new SRHost("sender",33333,33334,4,20);
        Host receiver = new SRHost("receiver",33334,33333,4,20);

        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    receiver.receive();
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
        }).start();
        Thread.sleep(1500);
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    sender.send();
                } catch (Exception e) {
                    throw new RuntimeException(e);
                }
            }
        }).start();
    }
}
```

SRHost.java

```
package host;

import util.timer.Timer;

import java.io.IOException;
import java.net.*;
import java.util.ArrayDeque;
import java.util.Queue;

public class SRHost extends Host {
    private boolean[] mark;
    //缓存队列
    private Queue<Integer> cache = new ArrayDeque<>();


    public SRHost(String name,int sendPort,int receivePort,int windowSize,int dataNumber) {
        super(name,sendPort,receivePort,windowSize,dataNumber);
    }

    @Override
    public void send() throws Exception {
        timer = new Timer(this);
        timer.start();
        senderDatagramSocket =new DatagramSocket();
        mark = new boolean[dataNumber +1];
        while(true){
            //向服务器端发送数据
            sendData();
            //从服务器端接受 ACK
            byte[] bytes = new byte[4096];
            senderDatagramPacket = new DatagramPacket(bytes, bytes.length);
            senderDatagramSocket.receive(senderDatagramPacket);

            String fromServer = new String(bytes, 0, bytes.length);
            int ack = Integer.parseInt(fromServer.substring(fromServer.indexOf("ACK: ") +
5).trim());
            mark[ack] = true;
            System.out.println(hostName + "接收到了 ACK: " + ack);

            //收到 base 的 ACK
            if(base == ack && base != dataNumber){
                base++;
                //乱序之后，把 base 值移到最远的位置
                for(int i = base; i < nextSeq;i++){
                    if(mark[i] == true){
                        base = i + 1;
                    }
                }
            }else if(base == ack && base == dataNumber){
                timer.pauseThread();
                sendEnd();
                break;
            }

            if(base == nextSeq){
                //停止计时器
                timer.clear();
            }else {
                //开始计时器
                timer.clear();
            }
        }
    }
    /**
     * 向服务器发送数据
     *
     * @throws Exception
     */
    private void sendData() throws Exception {
        inetAddress = InetAddress.getLocalHost();
        while (nextSeq < base + windowSize && nextSeq <= dataNumber) {
            //不发编号为 3 的数据
            if(nextSeq == 3||nextSeq == 12) {
                nextSeq++;
                continue;
            }

            String sendData = hostName + ": Sending to port " + sendPort + ", Seq = " +
nextSeq;

            // 模拟发送分组
            byte[] data = sendData.getBytes();
            DatagramPacket datagramPacket = new DatagramPacket(data, data.length,
inetAddress, sendPort);
```

```
            senderDatagramSocket.send(datagramPacket);
            System.out.println(hostName + "发送到" + sendPort + "端口，  Seq = " +
nextSeq);
            if(nextSeq == base){
                //开始计时
                //model.setTime(3);
                timer.clear();
            }
            nextSeq++;
            try {
                Thread.sleep(300);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public void timeOut() {
        try {

            String resendData = hostName
                    + ": Resending to port " + sendPort + ", Seq = " + base;
            byte[] data = resendData.getBytes();
            DatagramPacket datagramPacket = new DatagramPacket(data, data.length,
inetAddress, sendPort);
            senderDatagramSocket.send(datagramPacket);
            System.out.println(hostName
                    + "重新发送发送到" + sendPort + "端口，  Seq = " + base);
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
    /**
     * 向服务器发送结束信号
     */
    public void sendEnd() throws IOException {
        inetAddress = InetAddress.getLocalHost();
        int end = -1;
        String clientData = hostName + ": Sending to port " + sendPort + ", Seq = " + end;
        System.out.println("向服务器发送结束信号");

        byte[] data = clientData.getBytes();
```

```
            DatagramPacket datagramPacket = new DatagramPacket(data, data.length, inetAddress, sendPort);
            senderDatagramSocket.send(datagramPacket);
        }
        @Override
        public void receive() throws IOException {
            int exceptedSeq =1;
            try {
                receiverDatagramSocket = new DatagramSocket(receivePort);
                while (true) {
                    byte[] receivedData = new byte[4096];
                    receiverDatagramPacket = new DatagramPacket(receivedData, receivedData.length);
                    receiverDatagramSocket.receive(receiverDatagramPacket);

                    //收到的数据
                    String received = new String(receivedData, 0, receivedData.length);//offset 是初始偏移量
                    //System.out.println(received);
                    int seqIndex = received.indexOf("Seq = ");
                    int ack = Integer.parseInt(received.substring(seqIndex + 6).trim());
                    if(ack!=-1)System.out.println(hostName+" 接收到的报文为：["+received.trim()+"]");
                    if(ack == -1){
                        System.out.println("本次传输结束");
                        System.exit(0);
                        break;
                    }else{
                        sendAck(ack);

                        //收到了预期的数据
                        if (ack == exceptedSeq) {
                            System.out.println(hostName + "已收到预期编号 期待的数据 Seq = " + exceptedSeq);
                            //期待值加 1
                            exceptedSeq++;
                            //滑动窗口到最大值
                            while( cache.peek() != null && cache.peek()== exceptedSeq){
                                System.out.println("从服务器端缓存中读出数据:"+cache.element());
                                cache.poll();
                                exceptedSeq++;
                            }
                            System.out.println('\n');
                        } else {
                            System.out.println(hostName + "未收到预期编号 期待的数据 Seq
    = " + exceptedSeq);
                            cache.add(ack);
                            System.out.println('\n');
                        }
                    }
                }
            }catch(SocketException e){
                e.printStackTrace();
            }
        }
        //向客户端发送 ack
        public void sendAck(int ack) throws IOException {
            String response = hostName + " responses ACK: " + ack;
            byte[] responseData = response.getBytes();
            InetAddress responseAddress = receiverDatagramPacket.getAddress();
            int responsePort = receiverDatagramPacket.getPort();
            receiverDatagramPacket                    =                    new
    DatagramPacket(responseData,responseData.length,responseAddress,responsePort);
            receiverDatagramSocket.send(receiverDatagramPacket);
        }
        public static void main(String[] args) throws InterruptedException {
            Host sender = new SRHost("sender",33333,33334,4,20);
            Host receiver = new SRHost("receiver",33334,33333,4,20);

            new Thread(new Runnable() {
                @Override
                public void run() {
                    try {
                        receiver.receive();
                    } catch (IOException e) {
                        throw new RuntimeException(e);
                    }
                }
            }).start();
            Thread.sleep(1500);
            new Thread(new Runnable() {
                @Override
                public void run() {
                    try {
                        sender.send();
                    } catch (Exception e) {
                        throw new RuntimeException(e);
                    }
                }
            }).start();
        }
    }
```

Constants.java

```
package constants;

/**
 * TODO
 *
 * @author: lsxuan
 * @email: 1146887979@qq.com
 * @create: 2022-10-09 12:15
 */
public class Constants {
    public static final int DEFAULT_TIMEOUT = 3;

}
```

```
package util.timer;

import constants.Constants;
import host.Host;

/**
 *  计时器
 */
public class Timer extends Thread{
    private int timeout;
    private Host host;
    private final Object lock = new Object();
    static int min = 0, sec = 0;
    private boolean pause = false;//阻塞标志（默认关闭）

    public Timer(Host host) {
        this.timeout = Constants.DEFAULT_TIMEOUT;
        this.host = host;
    }
    public Timer(Host host,int timeout) {
        this.timeout=timeout;
        this.host = host;
```

```
}
public void pauseThread() {
        //System.out.println("计时器暂停");
        this.pause = true;
}//将阻塞信号开启

public void clear() {
        //System.out.println("计时器清零");
        min = 0;
        sec = 0;
        //show();
}

void onPause() {
        synchronized (lock) {
                try {
                        lock.wait();//阻塞线程方法
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
}//阻塞方法

public void resumeThread() {
        //System.out.println("计时器恢复");
        this.pause = false;
        synchronized (lock) {
                lock.notify();//恢复线程方法
        }
}//恢复方法

void show() {
        System.out.println("当前时间为：" + min + "：" + sec);
}

void mmshow() {
        int mm = (min * 60 + sec) * 100;
        System.out.print("毫秒示数为：" + mm + '\n');

}


public void run() {
        super.run();
```

```
            while (true) {
                if (pause) {

                    onPause();
                }
                try {
                    if(sec>timeout) {
                        //System.out.println("超时");
                        host.timeOut();
                        this.clear();
                    }
                    sec++;
                    //show();

                    if (sec == 60) {
                        sec = 0;
                        min++;
                    }

                    Thread.sleep(1000);

                } catch (Exception e) {
                    e.printStackTrace();
                    break;
                }
            }
    }//线程运行方法
}
```

Main.java

```
package test;

import host.FileHost;
import host.GBNHost;
import host.Host;
import host.SRHost;

import java.io.IOException;

public class Main {
    private static int host1Port = 808;        // host 1 占用端口

    private static int host2Port = 809;
    public static void main(String[] args) throws IOException {
//          startGBN();
//          startStopAndWait();
//          startDual();
//          startSR();
        startFile();
    }
    private static void startFile()throws IOException{
        Host sender = new FileHost("Sender",host1Port,host2Port,5);
        Host receive = new FileHost("Receiver",host2Port,host1Port,5);

        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    receive.receive();
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
        }).start();
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    sender.send();
                } catch (IOException e) {
                    throw new RuntimeException(e);
                } catch (Exception e) {
                    throw new RuntimeException(e);
                }
            }
        }).start();
    }
    private static void startSR() throws IOException {
        Host sender = new SRHost("Sender",host1Port,host2Port,3,25);
        Host receiver = new SRHost("Receiver",host2Port,host1Port,0,0);

        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    receiver.receive();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }).start();

        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    sender.send();
                } catch (IOException e) {
                    e.printStackTrace();
                } catch (Exception e) {
                    throw new RuntimeException(e);
                }
            }
        }).start();
    }

    private static void startStopAndWait() throws IOException {
        Host sender = new GBNHost("Sender",host1Port,host2Port,1,20);
        Host receiver = new GBNHost("Receiver",host2Port,host1Port,0,0);

        new Thread(() -> {
            try {
                receiver.receive();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }).start();

        new Thread(() -> {
            try {
                sender.send();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
        }).start();
    }

    private static void startGBN() throws IOException {
        Host sender = new GBNHost("Sender",host1Port,host2Port,5,20);
        Host receiver = new GBNHost("Receiver",host2Port,host1Port,0,0);

        new Thread(() -> {
            try {
                receiver.receive();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }).start();

        new Thread(() -> {
            try {
                sender.send();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
        }).start();
    }

    private static void startDual() throws IOException {

        GBNHost host1 = new GBNHost("Host A",host1Port,host2Port,3,20);
        GBNHost host2 = new GBNHost("Host B",host2Port,host1Port,4,30);

        // thread 2
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    host1.receive();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }).start();

        // thread 4
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    host2.receive();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }).start();

        // thread1
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    host1.send();
                } catch (IOException e) {
                    e.printStackTrace();
                } catch (Exception e) {
                    throw new RuntimeException(e);
                }
            }
        }).start();

        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    host2.send();
                } catch (IOException e) {
                    e.printStackTrace();
                } catch (Exception e) {
                    throw new RuntimeException(e);
                }
            }
        }).start();
    }
}
```