|  |  |
| --- | --- |
|  |  |

**合肥工业大学**

**计算机与信息学院**

**项目设计报告**

**课 程：计算机网络**

**专业班级：计算机科学与技术18-2班**

**学 号：2018211958**

**学生姓名：孙淼**

目录

[一、项目实验要求 3](#_Toc57830350)

[1）项目实验目的 3](#_Toc57830351)

[2）项目实验内容 3](#_Toc57830352)

[3）项目实验设计 3](#_Toc57830353)

[二、开发环境与工具 3](#_Toc57830354)

[三、软件使用说明及测试 3](#_Toc57830355)

[四、关键问题及其解决方法 10](#_Toc57830356)

[1） 新建文件路径问题 10](#_Toc57830357)

[2）命名规范的重要性 12](#_Toc57830358)

[3）根据提示导入库的时候需要注意 13](#_Toc57830359)

[4）参数缺失报错 13](#_Toc57830360)

[五、附录（详细注释的源码） 13](#_Toc57830361)

[server.java 13](#_Toc57830362)

[client.java 15](#_Toc57830363)

[GBN.java 17](#_Toc57830364)

### 一、项目实验要求

1）项目实验目的

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。

2）项目实验内容

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。

3）项目实验设计

1.基于UDP实现的GBN协议，可以不进行差错检测，可以利用UDP协议差错检测；

2.自行设计数据帧的格式，应至少包含序列号Seq和数据两部分；

3.自行定义发送端序列号Seq比特数L以及发送窗口大小W；

4.为了模拟ACK丢失，可以利用模N运算，每N次模拟丢包，或者每N次模拟接收。因为只是模拟，这个操作既可以在发送端也可以在接收端，如果在发送端，则少发送数据包，在接收端则不发回ACK；

5.当设置服务器端发送窗口的大小为1时，GBN协议就是停-等协议。

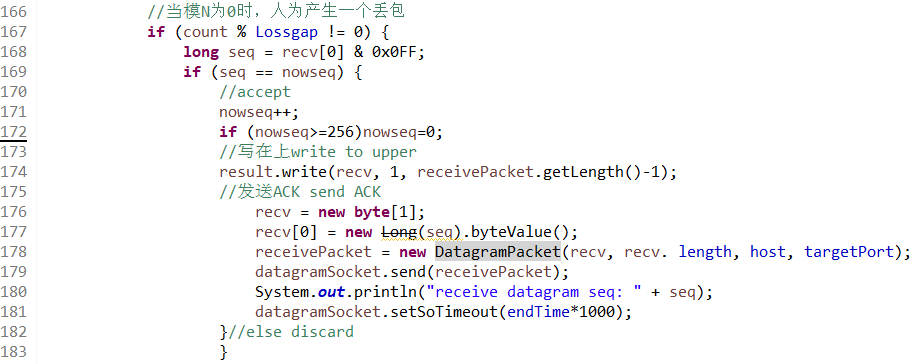
### 二、开发环境与工具

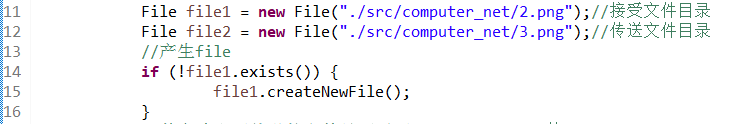
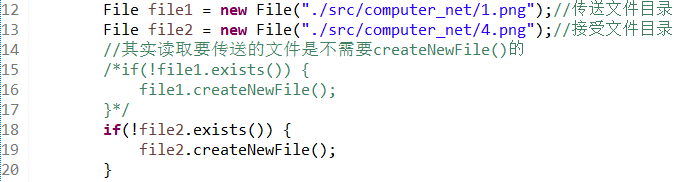
开发工具：Intel® Core(TM)i7-8750H CPU @ 2.20GHz 2.20GHz，windows 10

语言环境：java，eclipse-java-2019-06-R

### 三、软件使用说明及测试

对于实验内容一，若要只实现单向数据传输，只给服务器端send，客户receive即可；

对于实验内容二，为了模拟ACK丢失，我采用的是模N运算，每N次模拟丢包，或者每N次模拟接收。因为只是模拟，这个操作既可以在发送端也可以在接收端，，如果在发送端，则少发送数据包，在接收端则不发回ACK，我选择在接收端实现，也就是每隔N（我设置N=100）就不返回ACK，这部分具体实现代码如下：

对应实验内容三，此处展示改进后的GBN协议，也就是实现了双向数据传输的程序，具体实现在第五部分源码附录以及注释中可以看到，此处简单证明：

上面分别是服务器端和客户端的文件路径部分，对于服务器端，我选择读取3.png用于传向客户端，并且将从客户端接收到的文件1.png保存为2.png（双向传输）；对于客户端，我选择读取1.png用于传向服务器端，并且将从服务器端接收到的文件3.png保存为4.png（双向传输）。

对于实验设计二自行设计数据帧的格式，应至少包含序列号Seq和数据两部分，实际上在序列号前面还设计了base部分；

base与sendSeq序列号初始值相同但是作用不同，base用于超时重传，sendSeq单纯作为序列号，用于发送接收的显示。

根据参考资料，GBN一个分组的发送格式是:

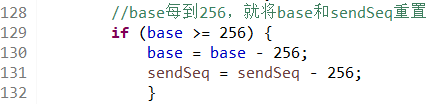
Base(1Byte) + seq(1Byte) + data(max 1024Byte)；

解释如下：GBN协议的传送流程是:从上层应用层获得到一个完整的数据报，将这个数据报进行拆分，因为在以太网中，数据帧的MTU为1500字节，所以UDP数据报的数据部分应小于1472字节(除去IP头部20字节与UDP头的8字节)

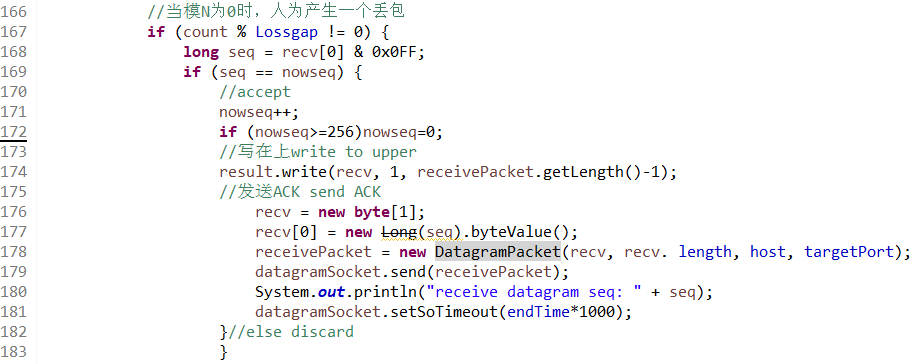
所以一个GBN数据帧最大传输的数据大小限制为1024B。需要注意的是：

对于发送方，起始base的值，窗口采用链表的数据结构存储每个发出去的数据包(因为后面超时需要重传，所以得按顺序能标示地存下来)进入一个循环，循环结束条件是所有需要传送的数据都已经发送完成,并且窗口中的分组都已经全部确认。在这个循环中,如果窗口内有空余,就开始发送分组,直到窗口被占满,计时器开始计时,之后进入接收ACK的状态,收到ACK之后，更新滑动窗口的位置,之后如果计时器超时，就将窗口内所有的分组全部重发一次。之后开始下一次循环。(构造数据包的时候只需要构造1字节的seq值放在头部)

对于接收方，记录一个base值,每成功接收--个数据帧,base+1,开始循环顺序接收数据帧,对于base不是目标值得数据帧直接丢弃，如果是符合要求的数据帧,就给发送方发送一个ACK=base的确认数据帧,直到发送方没有数据传来为止。

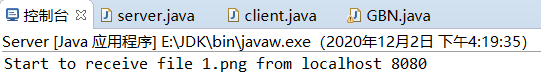
对于实验设计三，设计的序列号sendSeq为8位，对应最大序列号（10进制）为0-255，

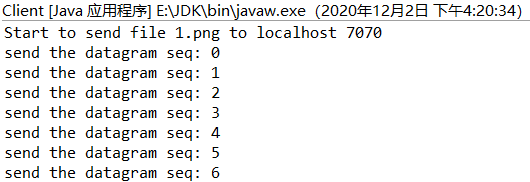
滑动窗口设为16，也是256的整数倍，注意此处若更改窗口大小为1，则GBN协议变为停等协议

对于实验设计四，为了模拟ACK丢失，我采用的是模N运算，每N次模拟丢包，或者每N次模拟接收。因为只是模拟，这个操作既可以在发送端也可以在接收端，，如果在发送端，则少发送数据包，在接收端则不发回ACK，我选择在接收端实现，也就是每隔N（我设置N=100）就不返回ACK，这部分具体实现代码如下：

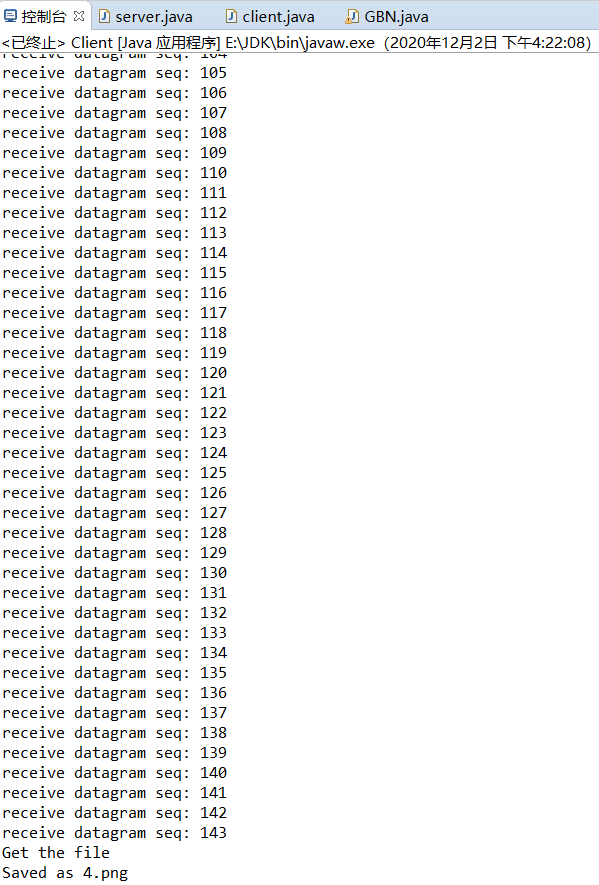
对于实验设计五，更改窗口大小为1，则GBN协议变为停等协议

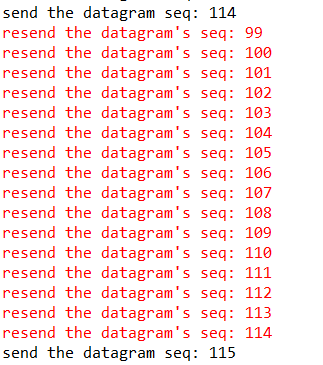
实际测试过程如下：

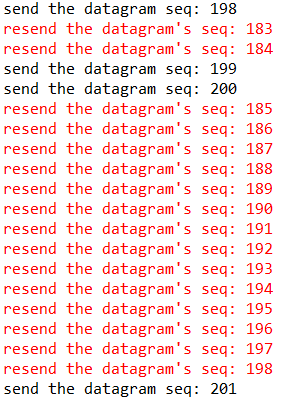
首先运行server.java，此时服务器已经做好从客户端8080接收文件1.png的准备，需要运行client建立连接。

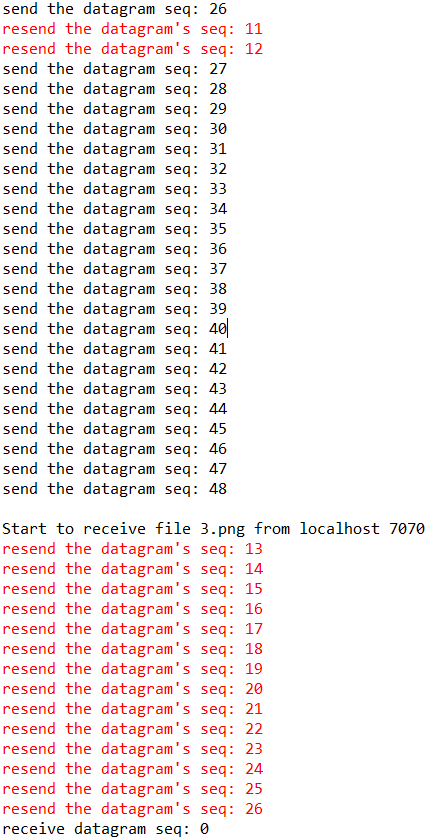
然后我们运行client.java，连接建立，此时客户端开始向服务器端7070传输文件1.png。

待到运行完成，我们可以得到如下结果

现在我们对其中的细节进行分析：

根据我们的模100接受方丢ACK包的设计，我们在传输过程中检查，果然在第99个序列号出发生了丢包（0-99刚好是100个），我们设计是假设接收方正在等待接收分组n，而分组n+ 1却已经到达了，于是，分组n+ 1被直接丢弃，所以发送方并不会出现在连续发送分组n,分组n+1之后，而分组n+ 1的ACK却比分组n的ACK更早到达发送方的情况。所以在sendSeq之后的包括第99数据包在内的连续16个（滑动窗口大小）的数据包都视作没有收到ACK而重传，情况如下：

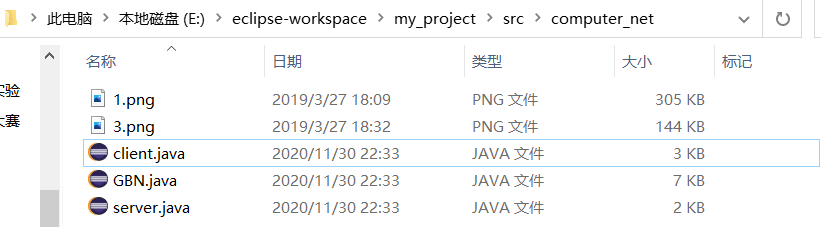
后面的丢包情况也都一样，情况如下面两张图所示，丢包的数量都是16

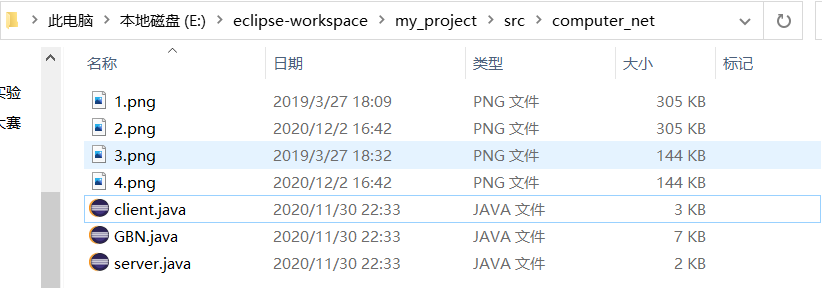
GBN中如果发送方的滑动窗口中，如果窗口内已经被发送但未收到确认的分组数目未达到窗口长度,就将窗口剩余的分组全部用来发送新构造好的数据,剩余未能发送的数据进行缓存。发送完窗口大小的数据分组后,开始等待接收从接收方发来的确定信息(ACK),GBN协议采取了累积确认，当发送方收到一个对分组n的ACK的时候，即表明接收方对于分组n以及分组n之前的分组全部都收到了。对于已经确认的分组,就将窗口滑动到未确认的分组位置(窗口又有空闲位置,可以发送剩余分组了),对于未确认的分组,如果计时器超时,就需要重新发送,直到收到接收方的ACK为止。对于超时的触发,GBN协议会将当前所有已发送但未被确认的分组重传，即如果当前窗口内都是已发送但未被确认的分组，一旦定时器发现窗口内的第-一个分组超时，则窗口内所有分组都要被重传。每次当发送方收到一个ACK的时候，定时器都会被重置。：

作为双向传输，client还完成了从服务器端7070接收文件3.png的任务，如下

并且成功将其存储为4.png，如下

需要注意的是，服务器端7070从客户端8080接收到的文件1.png被存储为2.png。

文件夹初始情况如下所示

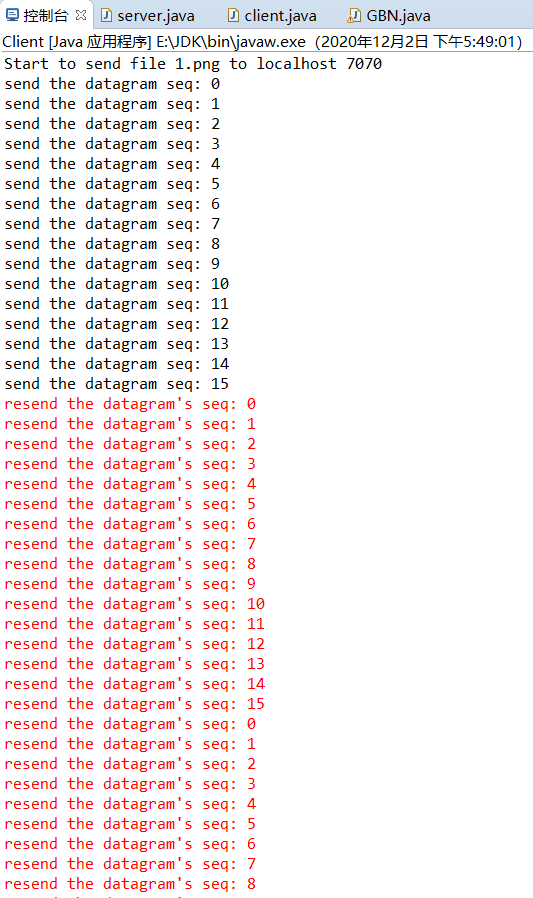
最后运行结束时，文件夹里情况如下所示

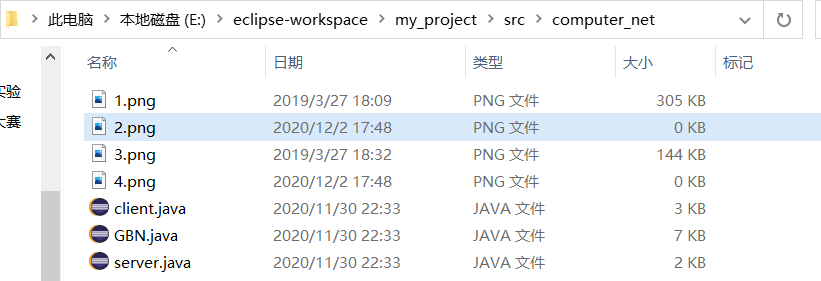
### 四、关键问题及其解决方法

本次实验总共耗时20小时左右，在实验的过程中，遇到了大大小小的问题，现在将部分难解决的问题总结如下：

1. 新建文件路径问题

在实验一开始，由于上次写java实现的socket还是在大二的时候，所以对是否需要建立file不太记得，导致一开始写的包含下面被注释掉的部分，于是运行的时候总是会发生如下的情况

也就是对16个初始数据包一直重传，自然接收方得到的文件2.png和4.png也都是0kb如下，原因是读取要传送的文件是不需要createNewFile()的，因为其已经建立的，但是语新建了路径，最终导致发送方发不出去，接收方文件一直收不到，最后发送方就一直重传。

2）命名规范的重要性

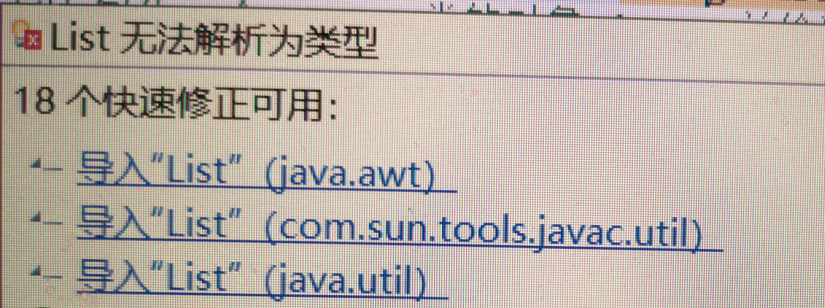
一直在上面的地方报错，错误说是不能对InputStream里面的非静态方法read进行动态引用，检查了一天，查询了无数资料，最后甚至都尝试将read方法进行重写，但是发现违反了Java的规定，于是解决不了。

最后发现是对一个变量命名时命名了大写，但是自己按照习惯在后面使用的时候使用的是自己习惯的小写，导致最后不统一导致的错误，这种细小的错误往往最难检查到，报错也会五花八门很难找到原因，也最耽误时间，再一次说明了我们在编写代码时需要注意细节。

3）根据提示导入库的时候需要注意

问题如下图，

在代码这部分使用了List数据结构

于是需要导入相应的包，提示如下

我刚开始没看仔细，直接选择import java.awt.List，结果就是一直报错：

类型 List 不是通用的;不能使用参数 <String> 将它参数化。

最后检查的时候才发现这个错误是import导入错的List导致的。

Java.awt.List 是一个界面控件，是重量级系统列表控件；

Java.util.List 是一种数据容器，是列表模式的数据容器；

两者区别很大，不能混用。

4）参数缺失报错

刚开始没注意，没有ownPort，导致服务器端和客户端都连接不到对方，一直超时重传错误，改正加入参数后就行了。

剩下的错误基本上都是对一些现有函数的调用不明确导致的错误，都没花费太多时间就解决了

### 五、附录（详细注释的源码）

server.java

**package** computer\_net;

**import** java.io.ByteArrayOutputStream;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**public** **class** server{

**public** **static** **void** main(String[] args) **throws** IOException, InterruptedException {

File file1 = **new** File("./src/computer\_net/2.png");//接受文件目录

File file2 = **new** File("./src/computer\_net/3.png");//传送文件目录

//产生file

**if** (!file1.exists()) {

file1.createNewFile();

}

//其实读取要传送的文件是不需要createNewFile()的

/\*if (!file2.exists()) {

file2.createNewFile();

}\*/

//GBN协议的服务器

GBN server = **new** GBN("localhost", 8080, 7070);//目标客户端端口为7070，本地服务器端口为8080

System.***out***.println("Start to receive file 1.png from "+"localhost "+ 8080);

**while** (**true**) {

ByteArrayOutputStream byteArrayOutputStream = server.receive();

**if** (byteArrayOutputStream.size() != 0) {

FileOutputStream fileOutputStream = **new** FileOutputStream(file1);

fileOutputStream.write(byteArrayOutputStream.toByteArray(), 0, byteArrayOutputStream.size());

fileOutputStream.close();

System.***out***.println("Get the file ");

System.***out***.println("Saved as 2.png");

fileOutputStream.close();

**break**;

}

Thread.*sleep*(50);

}

//捕获内存缓冲区的数据，转换成字节数组。

ByteArrayOutputStream byteArrayOutputStream = **new** ByteArrayOutputStream();

client.*CloneStream*(byteArrayOutputStream, **new** FileInputStream(file2));

System.***out***.println("\nStart to send file 3.png to" + "localhost" + 8080);

server.send(byteArrayOutputStream.toByteArray());

}

}

client.java

**package** computer\_net;

**import** java.io.ByteArrayOutputStream;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

**public** **class** client {

**public** **static** **void** main(String[] args) **throws** IOException, InterruptedException {

File file1 = **new** File("./src/computer\_net/1.png");//传送文件目录

File file2 = **new** File("./src/computer\_net/4.png");//接受文件目录

//其实读取要传送的文件是不需要createNewFile()的

/\*if(!file1.exists()) {

file1.createNewFile();

}\*/

**if**(!file2.exists()) {

file2.createNewFile();

}

ByteArrayOutputStream byteArrayOutputStream = **new** ByteArrayOutputStream();

*CloneStream*(byteArrayOutputStream, **new** FileInputStream(file1));

//协议的选择

GBN client = **new** GBN("localhost",7070,8080);//目标服务器端口8080，本地客户端端口7070

System.***out***.println("Start to send file 1.png to " + "localhost " + 7070);

client.send(byteArrayOutputStream.toByteArray());

System.***out***.println("\nStart to receive file 3.png from " + "localhost " + 7070);

**while** (**true**) {

byteArrayOutputStream = client.receive();

**if** (byteArrayOutputStream.size() != 0) {

FileOutputStream fileOutputStream = **new** FileOutputStream(file2);

fileOutputStream.write(byteArrayOutputStream.toByteArray(), 0, byteArrayOutputStream.size());

fileOutputStream.close();

System.***out***.println("Get the file ");

System.***out***.println("Saved as 4.png");

**break**;

}

Thread.*sleep*(50);

}

}

/\*\*

\* clone the input stream to a ByteArrayOutputstream object

\*

\* **@param** CloneResult the clone result of input stream

\* **@param** InputStream the input stream to be cloned

\* **@throws** IOException when read input stream, some exception occur

\*/

//将输入流转为字节数组对象

**static** **void** CloneStream(ByteArrayOutputStream CloneResult, InputStream InputStream) **throws** IOException {

//InputStream Inputstream

**byte**[] buffer = **new** **byte**[1024];

**int** length;

**while** ((length = InputStream.read(buffer)) != -1) {//？

CloneResult.write(buffer, 0, length);

}

CloneResult. flush();

}

}

GBN.java

**package** computer\_net;

//import java.awt.List;//导错包导致38行一直报错:类型 List 不是通用的；不能使用参数 <ByteArrayOutputStream> 将它参数化

**import** java.io.ByteArrayOutputStream;

**import** java.io.IOException;

**import** java.net.\*;

**import** java.net.DatagramPacket;

**import** java.net.DatagramSocket;

**import** java.net.InetAddress;

**import** java.net.SocketTimeoutException;

**import** java.net.UnknownHostException;

**import** java.util.List;

**import** java.util.LinkedList;

**class** GBN {

//获取到本机的InetAddress

**private** InetAddress host;

//目标端口和本机端口

**private** **int** targetPort, ownPort;

//若此处改窗口大小为1，则为停等协议

**private** **int** WindowSize = 16;

//最大发送时间，最大接收时间，一个数据包的超时时间

**private** **final** **int** sendMaxTime = 2,receiveMaxTime = 4,endTime=5;

//起始base值

**private** **long** base = 0;

//按照模N运算模拟丢包，N即Lossgap

**private** **final** **int** Lossgap = 100;

//对GBN中成员变量进行赋值A

GBN(String host, **int** targetPort, **int** ownPort) **throws** UnknownHostException {

//this关键字调用类中成员变量ownPort

**this**.ownPort = ownPort;

//确定主机的IP地址

**this**.host = InetAddress.*getByName*(host);

//this关键字调用类中成员变量targetPort

**this**.targetPort = targetPort;

}

/\*\*

\*向主机目的地址处传输数据相关功能

\*/

//send功能

**void** send(**byte**[] content) **throws** IOException {

//

**int** sendIndex = 0, length;

//自行设计数据包最大字节数

**final** **int** MAX\_LENGTH = 1024;

//用于UDP类DatagramSocket，表示接受或发送数据报的套接字，此处先建立套接字<DatagramSocket>

DatagramSocket datagramSocket = **new** DatagramSocket(ownPort);

//滑动窗口缓存，采用链表的数据结构，因为数据包需要按序标识并存下，这样才能实现重传

List<ByteArrayOutputStream> datagramBuffer = **new** LinkedList<>();

//实例化一个Integer类的对象timer，并给该对象赋值为0（计时器）

Integer timer = 0;

//自行设计序列号

**long** sendSeq = base;

**do** {

//直到窗口开始滑动

**while** (sendIndex < content.length && sendSeq < 256 && datagramBuffer.size()<WindowSize) {

//向滑窗缓存中加入byte型数据

datagramBuffer.add(**new** ByteArrayOutputStream());

//若数据部分字节数未超过限定的最大值，则不变；若超过，就取限定的最大值

length = content.length - sendIndex < MAX\_LENGTH ? content.length - sendIndex : MAX\_LENGTH;

//构造数据帧

ByteArrayOutputStream oneSend = **new** ByteArrayOutputStream();

//初始化byte数组

**byte**[] temp = **new** **byte**[1];

//

temp[0] = **new** ~~Long~~(sendSeq).byteValue();

//从temp中向oneSend写入从0开始处一字节的数据

oneSend.write(temp, 0, 1);

//从content中向oneSend写入从sendIndex开始处length字节的数据

oneSend.write(content, sendIndex, length);

//DatagramPacket表示存放数据的数据报，此句表示向host地址targetPort端口处发送oneSend.size()长度的oneSend.toByteArray()

DatagramPacket datagramPacket = **new** DatagramPacket(oneSend.toByteArray(), oneSend.size(), host, targetPort);

//从datagramSocket发送datagramPacket

datagramSocket.send(datagramPacket);

//向滑窗缓存中sendSeq-base的位置写入content中从sendIndex开始处length字节的数据

datagramBuffer.get((**int**) (sendSeq - base)).write(content, sendIndex, length);

//后移length，循环用

sendIndex += length;

//“输出的数据包序号为sendSeq”

System.***out***.println("send the datagram seq: " +sendSeq);

//序号+1，循环用

sendSeq++;

}

//倒计时函数实现1秒倒计时

datagramSocket.setSoTimeout(1000);

//创建一个接受ACK的数据包

DatagramPacket receivePacket;

**try**{//从base开始接收ACK receive ACKs for base

**while** (**true**) {

//1500字节的recv字节数组

**byte**[] recv = **new** **byte**[1500];

//存放recv.length长的recv的数据报

receivePacket = **new** DatagramPacket(recv, recv.length);

//套接字开始接收数据包，当接收到时，receivePacket填满为接收到的数据

datagramSocket.receive(receivePacket);

//与11111111按位与，结果化为int型赋予ack，这一步是为了实现接收方按序接收数组

**int** ack = (**int**) ((recv[0] & 0x0FF));

//如果接收到了第一个，就继续接收第二个，并顺便清空缓存

**if** (ack == base) {

base++;

datagramBuffer.remove(0);

**break**;

}

}

//超时功能

} **catch** (SocketTimeoutException e) {

timer++;

}

//如果超出设定的超时时间

**if** ( timer> **this**.sendMaxTime) {

// 重发所有没有收到ACK（超时）的数据报

**for** (**int** i = 0; i<datagramBuffer.size();i++) {

ByteArrayOutputStream resender = **new** ByteArrayOutputStream();

**byte**[] temp = **new** **byte**[1];

temp[0] = **new** ~~Long~~(i + base).byteValue();

resender.write(temp, 0, 1);

resender.write(datagramBuffer.get(i).toByteArray(), 0, datagramBuffer.get(i).size());

DatagramPacket datagramPacket = **new** DatagramPacket(resender.toByteArray(), resender.size(), host, targetPort);

datagramSocket.send(datagramPacket);

System.***err***.println("resend the datagram's seq: "+ (i + base));

}

//重置timer

timer = 0;

//base每到256，就将base和sendSeq重置

**if** (base >= 256) {

base = base - 256;

sendSeq = sendSeq - 256;

}

}

//直到数据全部传输完成

}**while** (sendIndex < content.length || datagramBuffer.size() !=0) ;

//关闭套接字

datagramSocket.close();

}

/\*\*

\* 接收主机目的地址处的数据相关功能

\*/

//receive

ByteArrayOutputStream receive() **throws** IOException {

//用于 used to simulate datagram loss

**int** count = 1,time=0;

**long** nowseq = 0;

//存储接收的 store the received content

ByteArrayOutputStream result = **new** ByteArrayOutputStream();

//接收数据报和发送ACK的UDP套接字 UDP socket to receive datagram and send ACKs

DatagramSocket datagramSocket = **new** DatagramSocket(ownPort);

//一个暂时的数据报 one temp datagram packet

DatagramPacket receivePacket;

//

datagramSocket.setSoTimeout(endTime\*1000);

**while** (**true**) {

//接收一个数据报并且返回ACK

**try** {

**byte**[] recv = **new** **byte**[1500];

receivePacket = **new** DatagramPacket(recv, recv.length, host, targetPort);

datagramSocket.receive(receivePacket);

//当模N为0时，人为产生一个丢包

**if** (count % Lossgap != 0) {

**long** seq = recv[0] & 0x0FF;

**if** (seq == nowseq) {

//accept

nowseq++;

**if** (nowseq>=256)nowseq=0;

//写在上write to upper

result.write(recv, 1, receivePacket.getLength()-1);

//发送ACK send ACK

recv = **new** **byte**[1];

recv[0] = **new** ~~Long~~(seq).byteValue();

receivePacket = **new** DatagramPacket(recv, recv. length, host, targetPort);

datagramSocket.send(receivePacket);

System.***out***.println("receive datagram seq: " + seq);

datagramSocket.setSoTimeout(endTime\*1000);

}//else discard

}

}**catch** (SocketTimeoutException e) {

**break**;

}

//datagramSocket.setSoTimeout(endTime);

count++;

}

datagramSocket.close();

**return** result;

}

}