滑动窗口协议仿真

实习类别	课程设计
学生姓名	徐晓亭
专　　业	网络工程
学　　号	150522136
指导教师	苏伟、闫飞
学　　院	计算机科学技术学院
时　　间	2017～2018 第一学期

<table>
<tr><td>课程设计题目</td><td colspan="5">滑动窗口协议仿真</td></tr>
<tr><td>姓名</td><td>徐晓亭</td><td>学号</td><td>150522136</td><td>班级</td><td>1505221</td></tr>
<tr><td>学院</td><td>计算机科学技术学院</td><td>专业</td><td colspan="3">网络工程</td></tr>
<tr><td colspan="2">指导教师</td><td colspan="4">闫飞、苏伟</td></tr>
<tr><td colspan="2">课程设计目的</td><td colspan="4">掌握滑动窗口协议的基本原理，并能够用所学计算机高级语言进行编程模拟</td></tr>
<tr><td colspan="2">课程设计所需环境</td><td colspan="4">开发环境： Eclipse 运行环境:Windows/Linux/Max OS X</td></tr>
<tr><td colspan="2">课程设计任务要求</td><td colspan="4">1．程序按照滑动窗口协议实现端对端的数据传送。包括协议的各种策略，如包丢失、停等应答、超时等都应有所仿真实现
2．显示数据传送过程中的各项具体数据。双方帧的个数变化，帧序号，发送和接受速度，暂停或重传提示等</td></tr>
<tr><td colspan="6">教研室审核意见：

教研室主任签字： 年 月 日</td></tr>
</table>

课程设计任务书

七. 心得体会
八. 参考文献

课程设计的主要内容

1.引言

早期的网络通信中，通信双方不会考虑网络的拥挤情况直接发送数据。由于大家不知道网络拥塞状况，一起发送数据，导致中间结点阻塞掉包，谁也发不了数据。在数据传输过程中，我们总是希望数据传输的更快一些，但如果发送方把数据发送的过快，接收方就可能来不及接收，这就造成数据的丢失。因此就有了滑动窗口机制来解决这些问题。早期我们使用的是 1Bit 滑动窗口协议，一次只发送一个帧，等收到 ACK 确认才发下一个帧，这样对信道的利用率太低了。因此提出了一种采用累积确认的连续 ARQ 协议，接收方不必对收到的帧逐个发送 ACK 确认，而是收到几个帧后，对按序到达的最后一个帧发送 ACK 确认。同 1Bit 滑动窗口协议相比，大大减少了 ACK 数量，并消除了延迟 ACK 对传输效率的影响。但是，这会产生一个新的问题，如果发送方发送了五个帧，而中间的第三个帧丢失了。这时接收方只能对前两个帧发出确认。发送方无法知道后面三个帧的下落，只好把后面的三个帧再重传一次，这就是回退 N 协议。为了解决这个问题，又提出了选择重传协议。当接收方发现某帧出错后，继续接受后面送来的正确的帧，只是不交付它们，存放在自己的缓冲区中，并且要求发送方重传出错的那一帧。一旦收到重传来的帧后，就可以将存于缓冲区中的其余帧一并按正确的顺序递交给主机。

2.基本原理

2.1 窗口机制

滑动窗口协议的基本原理就是在任意时刻，发送方都维持了一个连续的允许发送的帧的序号，称为发送窗口；同时，接收方也维持了一个连续的允许接收的帧的序号，称为接收窗口。发送窗口和接收窗口的序号的上下界不一定要一样，甚至大小也可以不同。不同的滑动窗口协议窗口大小一般不同。发送方窗口内的序号代表了那些已经被发送，但是还没有被确认的帧，或者

是那些可以被发送的帧。接受方为其窗口内的每一个序号保留了一个缓冲区。与每个缓冲区相关联的还有一位，用来指明该缓冲区是满的还是空的。

若从滑动窗口的观点来统一看待 1Bit 滑动窗口、后退 N 及选择重传三种协议，它们的差别仅在于各自窗口尺寸的大小不同而已。

1Bit 滑动窗口协议：发送窗口大小为 1，接收窗口大小也为 1。

后退 N 协议：发送窗口大小大于 1，接收窗口大小为 1。

选择重传协议：发送窗口大小大于 1,接收窗口大小也大于 1。

2.2 1Bit 滑动窗口协议

当发送窗口和接收窗口的大小固定为 1 时，滑动窗口协议退化为停等协议（Stop－And－Wait)。该协议规定发送方每发送一帧后就要停下来，等待接收方已正确接收的确认（Acknowledgement）返回后才能继续发送下一帧。由于接收方需要判断接收到的帧是新发的帧还是重新发送的帧，因此发送方要为每一个帧加一个序号。由于停等协议规定只有一帧完全发送成功后才能发送新的帧，因而只用 1Bit 来编号就够了。

2.3 后退 N 协议

由于停等协议要为每一个帧进行确认后才继续发送下一帧，大大降低了信道利用率，因此又提出了后退 N 协议。后退 N 协议中，发送方在发完一个数据帧后，不停下来等待应答帧，而是连续发送若干个数据帧，即使在连续发送过程中收到了接收方发来的应答帧，也可以继续发送。且发送方在每发送完一个数据帧时都要设置超时定时器。只要在所设置的超时时间内仍收到确认帧，就要重发相应的数据帧。如：当发送方发送了 N 个帧后，若发现该 N 帧的前一个帧在计时器超时后仍未返回其确认信息，则该帧被判为出错或丢失，此时发送方就不得不重新发送出错帧及其后的 N 帧。

从这里不难看出，后退 N 协议一方面因连续发送数据帧而提高了效率，但另一方面，在重传时又必须把原来已正确传送过的数据帧进行重传（仅因这些数据帧之前有一个数据帧出了错），这种做法又使传送效率降低。由此可见，若传输信道的传输质量很差因而误码率较大时，连续确认协议

不一定优于停止等待协议。此协议中的发送窗口的大小为 N，接收窗口仍是 1。

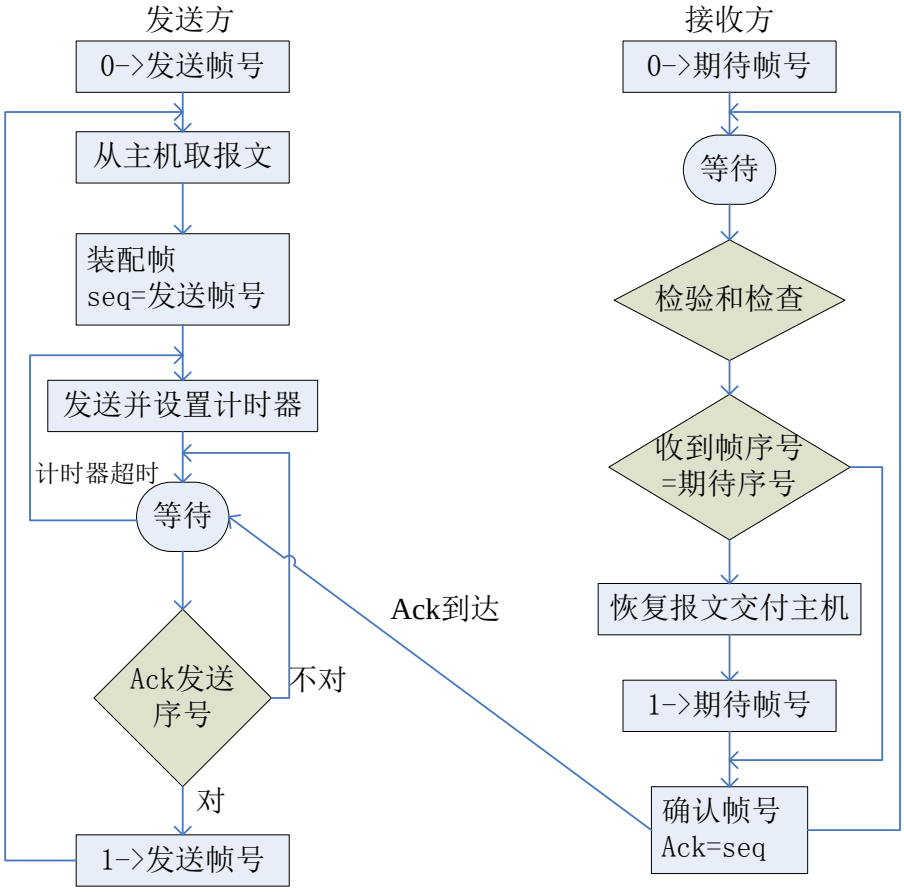


图 2-1 1Bit 滑动窗口协议示图

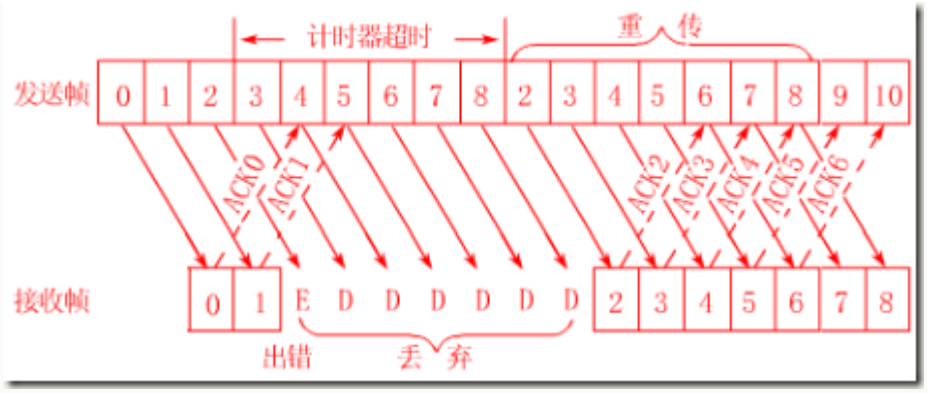


图 2-2 后退 N 协议示图

2.4 选择重传协议

在后退 N 协议中，接收方若发现错误帧就不再接收后续的帧，即使是正确到达的帧，这显然是一种浪费。另一种效率更高的策略是当接收方发现某帧出错后，其后继续送来的正确的帧虽然不能立即递交给接收方的高层，但接收方仍可收下来，存放在一个缓冲区中，同时要求发送方重新传送出错的那一帧。一旦收到重新传来的帧后，就可以原已存于缓冲区中的其余帧一并按正确的顺序递交高层。这种方法称为选择重发(SELECTICE REPEAT)。

2.5 流量控制

TCP 的特点之一是提供体积可变的滑动窗口机制，支持端到端的流量控制。TCP 的窗口以字节为单位进行调整，以适应接收方的处理能力。处理过程如下：

（1）TCP 连接阶段，双方协商窗口尺寸，同时接收方预留数据缓存区；

（2）发送方根据协商的结果，发送符合窗口尺寸的数据字节流，并等待对方的确认；

（3）发送方根据确认信息，改变窗口的尺寸，增加或者减少发送未得到确认的字节流中的字节数。调整过程包括：如果出现发送拥塞，发送窗口缩小为原来的一半，同时将超时重传的时间间隔扩大一倍。

（4）滑动窗口机制为端到端设备间的数据传输提供了可靠的流量控制机制。然而，它只能在源端设备和目的端设备起作用，当网络中间设备（例如路由器等）发生拥塞时，滑动窗口机制将不起作用。

3.需求分析

3.1 课程设计题目：滑动窗口协议仿真

3.2 开发环境：Eclipse Java 语言

3.3 运行环境：Windows/Linux/Max OS X

3.4 课程设计任务及要求：

（1）程序按照滑动窗口协议实现端对端的数据传送。包括协议的各种策略，如包丢失、停等应答、超时等都应有所仿真实现。

（2）显示数据传送过程中的各项具体数据。双方帧的个数变化，帧序号，发送和接受速度，暂停或重传提示等。

3.5 界面要求：

此次课程设计要求的所有功能应可视，使用语言为 Java，所以可以运行在所有系统之下，只要具有相应的 Java 运行环境。观察发送方（Sender）发送数据包到接收方（Receiver）时，界面应显示出双方帧个数的变化，帧序号，发送和接受速度，暂停或重传提示等。界面中必须动态显示数据帧的发送和接受情况，包括在相应的窗口详细显示相应的 ACK 和其他收发数据帧后发出的消息，以表明模拟协议的正确运作过程。在各种情况下，接受方和发送方窗口应实时显示帧的发送和接受情况，包括序号，时间戳，内容等。以及窗口的填充和清空情况。

3.6 网络接口要求：

两台机器或一台机器中多个线程模拟发送方，单个线程模拟接受方，接收数据的端口初始应为监听状态。发送方向接受方发起连接，成功后开始发送数据。

4.概要设计

4.1 主要类的定义如下：

基本帧：

```
public class Frame implements Serializable{
    /**
     * By Bill
     */
    private static final long serialVersionUID = 1L;
    private int kind;                //确认帧的类型: 1--->ack
    private int seq;                 //序号
    private int ack;                 //确认序号
    private char data;               //内容
    private int size;                //大小
    public Frame() {
    }
    public Frame(int seq, int ack, char data,int size) {
        this.seq = seq;
        this.ack = ack;
        this.data = data;
        this.size = size;
```

```
	}
}
```
多线程帧：
```
public class SupportedThread implements Runnable{
	private Frame f;
	private Frame frameInput;
	private ObjectInputStream input;
	private ObjectOutputStream output;
	private boolean success;
	public SupportedThread(Frame f) {
		this.f = f;
		this.success = false;
	}
	public void run() {
			Socket s = new Socket();
			try {
				s.connect(new InetSocketAddress("192.168.2.205", 9000),1000);
				output = new ObjectOutputStream(s.getOutputStream());
				System.out.println(this.f.getSeq());
				output.writeObject(this.f);
			} catch (IOException e1) {
				// TODO Auto-generated catch block
				e1.printStackTrace();
			}
			switch(randomNumber()) {
				case 0:
					/*
					 * 1.发送分组丢失,
					 * */
					System.out.println("发送分组丢失, 稍等之后开始重传. ");
					try {
						s.close();
					} catch (IOException e) {
						// TODO Auto-generated catch block
						e.printStackTrace();
					}
					break;
				case 1:
					/*
					 * 2.确认分组丢失,
					 * */
					System.out.println("确认分组丢失, 稍等之后开始重传. ");
					try {
						s.close();
```

```
                        } catch (IOException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        }
                        break;
                    case 2:
                        /*
                         * 3.确认分组延迟. 这些情况都会引起重传,
                         * */
                        System.out.println("确认分组延时, 稍等之后开始重传. ");
                        try {
                            s.close();
                        } catch (IOException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        }
                        break;
                    default:
                        /*
                         *  正常收到分组:1. 正常确认, 2. 重复确认
                         * */
                            try {
                                s.setSoTimeout(50);
                                input = new
ObjectInputStream(s.getInputStream());
                                frameInput = (Frame)input.readObject();
                                if(frameInput.getKind()==1) {
                                    System.out.println("正常收到确认. . . ");
                                    this.success = true;
                                } else {
                                    this.success = true;
                                    System.out.println("收到重复确认. . . ");
                                }
                            } catch(Exception e) {
//                                e.printStackTrace();
                                System.out.println("服务器端校验错误. . . ");
                        }
                        break;
                }
                try {
                    s.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
```

```
                }
        }
        ……
}
```
滑动窗口类：
```
public class Group {
        private SendQueue sq;
        private int sendNumber;
        private Frame[] frameGroup = new Frame[5];
        public Group(SendQueue sq) {
                this.sq = sq;
                this.sendNumber = 0;
        }
        public void initFrame() {
                for(int i=0; i<5; i++)
                {
                        this.frameGroup[i]=this.sq.remove();
                }
        }
        public void slideWindow() {
                initFrame();
                multiSend();
        }
        public void multiSend() {
                SupportedThread[] group = new SupportedThread[5];
                int number=0 ;                          //success 计数器
                int i=0;
                for(int j=0; j<5; j++) {
                        group[j]=new SupportedThread(frameGroup[j]);
                        group[j].run();
                }
                while(true) {
                        if(group[i%5].returnSuccess()) {
                                number++;
                        } else {
                                group[i%5].run();
                                this.sendNumber++;
                        }
                        if(number!=0&&number%5==0) {
                                break;
                        }
                        i++;
                }
                this.sendNumber = this.sendNumber+5;
```

```
    }
    public int getSendNumber() {
        return sendNumber;
    }
}
```

4.2 类的功能及类图

类名：Frame

功　能：基本帧类型，作为其他的基础类型

类名：SupportThread

功　能：支持多线程发送帧的类

类名：Group

功　能：实现滑动窗口的主要类型

类名：SendQueue

功　能：发送队列类

类名：Client

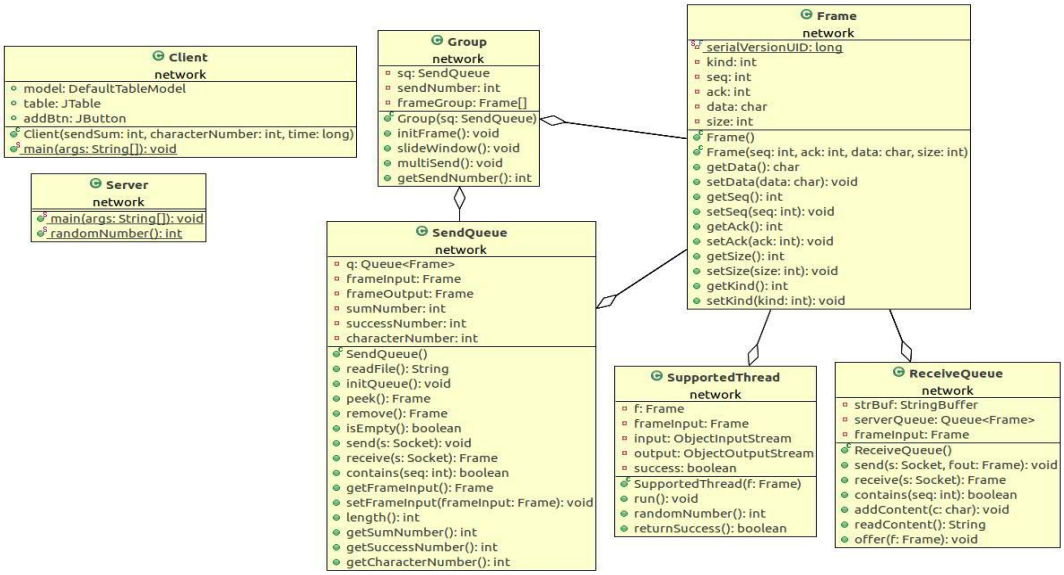
功　能：客户端发送主类

类名：ReceiveQueue

功　能：接受队列类

类名：Server

功　能：接受方主类



4.3 整体过程：

发送端：

（1）从文件中读取内容至主存；

（2）连接服务器端，设置超时计时器，这里是 100MS，超时则会进行异常处理，关闭 Socket，重发；

（3）Group 对象调用 SlideWindow 函数创建 5 个线程来发送数据帧，只有全部正常的发送才会进行下一组帧的发送。

（4）五个线程等待确认，正常确认的数据从缓存中删去，异常的进行重发。

（5）当发送队列中为空时，停止发送。最后弹出一个窗体，进行一些统计，包括:速率，时间，正确率等。

接收端：

（1）等待，接收数据帧；

（2）校验数据帧，假定产生随机结果，1%的概率校验错误；

（3）校验错误时，丢弃数据帧，其余什么也不做(这和真实情况是一样的)；

（4）如果出现接收超时（假定未收到发送方发送的数据帧），则当前线程阻塞，不给发送发任何回应；

（5）如果校验正确，首先判断是否是上一帧的重发。是上一帧的重发，则丢弃数据帧，并发送确认帧 ACK；是新的数据帧，则保存数据帧到当前接收窗口，并发送确认帧 ACK。

（6）送数据帧至主机，这里的主机是不断的重写一个文件，将收到的内容写入文件。

5. 源代码（主要）

5.1 多线程执行函数

```
public void multiSend() {
 SupportedThread[] group = new SupportedThread[5];
 int number=0 ;                    //success 计数器
 int i=0;
 for(int j=0; j<5; j++) {
```

```
                group[j]=new SupportedThread(frameGroup[j]);
                group[j].run();
            }
            while(true) {
                if(group[i%5].returnSuccess()) {
                    number++;
                } else {
                    group[i%5].run();
                    this.sendNumber++;
                }
                if(number!=0&&number%5==0) {
                    break;
                }
                i++;
            }
            this.sendNumber = this.sendNumber+5;
        }
```

5.2 多线程实现函数

```
          public void run() {
                Socket s = new Socket();
                try {
                    s.connect(new InetSocketAddress("192.168.2.205", 9000),1000);
                    output = new ObjectOutputStream(s.getOutputStream());
                    System.out.println(this.f.getSeq());
                    output.writeObject(this.f);
                } catch (IOException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }
                switch(randomNumber()) {
```

```
case 0:
        /*
         * 1.发送分组丢失,
         * */
        System.out.println("发送分组丢失, 稍等之后开始重传. ");
        try {
            s.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        break;
case 1:
        /*
         * 2.确认分组丢失,
         * */
        System.out.println("确认分组丢失, 稍等之后开始重传. ");
        try {
            s.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        break;
case 2:
        /*
         * 3.确认分组延迟. 这些情况都会引起重传,
         * */
        System.out.println("确认分组延时, 稍等之后开始重传. ");
        try {
```

```
                    s.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                break;
            default:
                /*
                 * 正常收到分组:1.正常确认,2.重复确认
                 * 接收有可能出现异常,因为客户端可能不会发送帧,所以
                 * */
                    try {
                        s.setSoTimeout(50);
                        input = new ObjectInputStream(s.getInputStream());
                        frameInput = (Frame)input.readObject();
                        if(frameInput.getKind()==1) {
                            System.out.println("正常收到确认...");
                            this.success = true;
                        } else {
                            this.success = true;
                            System.out.println("收到重复确认...");
                        }
                    } catch(Exception e) {
//                      e.printStackTrace();
                        System.out.println("服务器端校验错误...");
                    }
                    break;
            }
```

```
                try {
                    s.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
        }
```

5.3 Client 代码

```
         public class Client extends JFrame{
        public DefaultTableModel model = null;
        public JTable table = null;

        public JButton addBtn = null;
        public Client(int sendSum, int characterNumber,long time)
        {
            super("TableDemo");
            setTitle("传输结果统计");
            String[][] datas = {};
            String[] titles = { "统计指标", "统计结果" };
            model = new DefaultTableModel(datas, titles);
            table = new JTable(model);

            model.addRow(new String[] {"传输总帧数",Integer.toString(sendSum)});
            model.addRow(new String[] {"一次性传输成功的个数
",Integer.toString(characterNumber)});
            model.addRow(new String[] {"失败的个数
",Integer.toString(sendSum-characterNumber)});
            model.addRow(new String[] {"传输数据总大小:",new
DecimalFormat("0.00").format((float)(characterNumber/1024.0)) +"KB"});
```

```
            model.addRow(new String[] {"正确率", new
DecimalFormat("0.00%").format((float)characterNumber/sendSum)});
            model.addRow(new String[] {"时间",Long.toString(time)});
            model.addRow(new String[] {"速率",new
DecimalFormat("0.00").format((float)characterNumber/(float)time * 1000) +
"KB/S"});
            add(new JScrollPane(table));
            setSize(400, 300);
            setLocationRelativeTo(null);
            setDefaultCloseOperation(EXIT_ON_CLOSE);
            setVisible(true);

        }
        public static void main(String[] args) throws IOException {
            long start = System.currentTimeMillis();
            long end;
            SendQueue sq = new SendQueue();
            sq.initQueue();
            Group g = new Group(sq);
            while(!sq.isEmpty())
            {
                g.slideWindow();
            }
            end = System.currentTimeMillis();
            new Client(g.getSendNumber(),sq.getCharacterNumber(),end-start);
        }
}
```

5.4 Server 代码

```
package network;
import java.net.*;
```

```
import java.util.*;
import java.io.*;

public class Server {
	public static void main(String[] args) throws Exception{
		ServerSocket server = new ServerSocket(9000);
		ReceiveQueue rq = new ReceiveQueue();
		Socket s;
		while(true){
			s = server.accept();
			Frame f = new Frame();
			f=rq.receive(s);
			switch(randomNumber()) {
				case 0:
					/*
					 * 1.分组出错,
					 * */
					System.out.println("校验出错, 直接丢弃. . . ");
					break;
				default:
					/*
					 * 2. 帧正常且正常接收
					 * */
					if(rq.contains(f.getSeq())) {
						System.out.println("收到重复帧. . . ");
					} else {
						System.out.println("收到正常帧. . . ");
						rq.offer(f);
						rq.addContent(f.getData());
						f.setKind(1);
```

```
                    rq.send(s, f);
                }
                break;
            }
            FileWriter fw = new FileWriter("./src/network/copy.txt");
            fw.write(rq.readContent() + "\n");
            fw.close();
        }
    }
    public static int randomNumber() {
        Random r = new Random();
        int number = r.nextInt(1000);
        return number%100;
    }
}
```

6.调试与操作说明

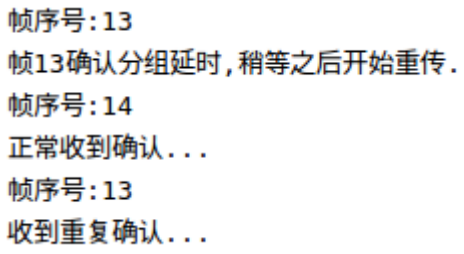
6.1 基本操作：先运行接受端，在运行发送端，发送端开始发送，知道内容发送完毕。

```
帧序号:0
正常收到确认...
帧序号:1
正常收到确认...
帧序号:2
正常收到确认...
帧序号:3
正常收到确认...
帧序号:4
正常收到确认...
帧序号:5
正常收到确认...
```

```
序号0收到正常帧...
序号1收到正常帧...
序号2收到正常帧...
序号3收到正常帧...
序号4收到正常帧...
序号5收到正常帧...
序号6收到正常帧...
序号7收到正常帧...
```

图 6-1 发送端　　图 6-2 接收端

6.2 确认分组延时：确认分组延时，发送端稍后重传数据帧，接受端仍发送确认帧。



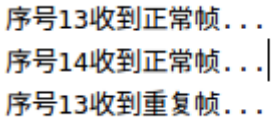
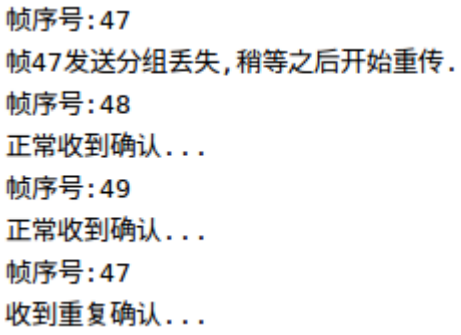


图 6-3 发送端　　图 6-4 接收端

6.3 发送分组丢失：发送分组丢失，接收端没有接收到任何信息，发送端定时重传。



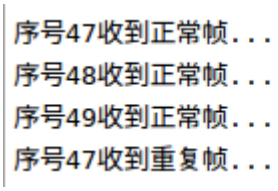


图 6-5 发送端　　图 6-6 接收端

6.4 帧错误：发送数据帧出错，接收端直接丢弃，发送端定时重传。

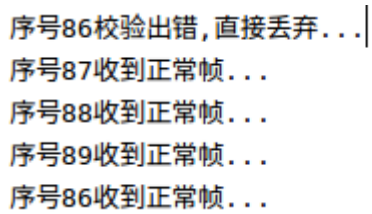
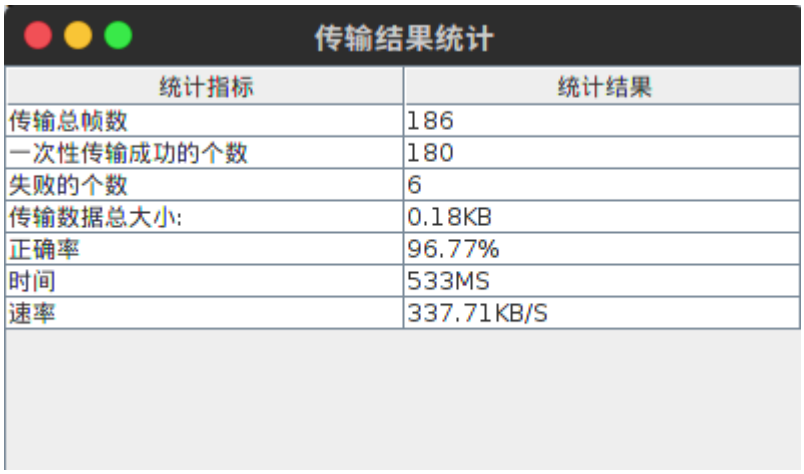


图 6-7 发送端　　图 6-8 接收端

6.5 相关指标统计



7.心得体会

通过这次课程设计，我学习到很多方面知识，如：软件开发流程，计算机网络，代码编写能力的提升，系统设计能力方面的提升等。

软件开发流程：需求分析→概要设计→详细设计→编码→测试→软件交付→验收→维护。

整个过程中，我感触最深的是设计方面。我使用的是 Java 语言来写的，自然需要分析和设计需要哪些是主类，哪些是辅助类，哪些是基类，哪些是子类。这涉及到横向设计和纵向设计。横向设计是指类的属性和方法，比如说：某一个类的具体功能，超时重传这个归属发送还是接收方的问题，确认丢失这种情况是属于发送方还是接收方的问题等等。纵向设计就是类的继承，比如说：帧是基类，发送队列是子类。这些都是需要考虑的问题，最后达成的目标是每一个类功能明确，划分合理，这样在编写代码这一环节可以非常轻松。

回顾起此课程设计，至今我仍感慨颇多，从理论到实践，在这段日子里，可以说得是苦多于甜，但是可以学到很多很多的东西，同时不仅可以巩固了以前所学过的知识，而且学到了很多在书本上所没有学到过的知识。通过这次课程设计使我懂得了理论与实际相结合是很重要的，只有理论知识是远远不够的，只有把所学的理论知识与实践相结合起来，从理论中得出结论，才能真正为社会服务，从而提高自己的实际动手能力和独立思考的能力。在设计的过程中遇到问题，可以说得是困难重重，但可喜的是最终都得到了解决。

8.参考文献

[1] Andrew S. Tandenbaum. Computer Networks(3rdEDITION)[M]. Prentice Hall ,1996.

[2] 胡晓峰,等编. 计算机网络原理[M]. 长沙. 国防科技大学出版社. 1997.

[3] 《计算机网络》（第 5 版）——电子工业出版社 谢希仁 编著

[4] 陈洛资,等. 计算机网络软件设计、开发与编程[M] .北京:科学出版社. 1994.

[5] 高传善,等. 计算机网络教程[m] . 上海:复旦大学出版社. 1996.

9.附录

评语： 评阅教师签名：　　年　　月　　日	
成　绩	