



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	可靠数据传输协议设计与实现					
姓名	马旭		院系	计算科学与技术学院		
班级	1603106		学号	1160300601		
任课教师	聂兰顺		指导教师	聂兰顺		
实验地点	格物楼 207		实验时间	2018.11.7		
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						

目录

1. 实验目的.....	3
2. 实验内容.....	3
3. 实验过程.....	3
3.1. GBN 协议数据格式	3
3.2. 确认分组格式.....	3
3.3 基本流程图.....	3
3.4 协议典型交互过程.....	5
3.5 数据分组丢失验证模拟方法.....	6
3.6 程序实现的主要类及主要函数.....	7
4. 实验结果.....	12
4.1. 基本功能实现:	12
4.2. 实现 GBN 协议:	12
4.3. 实现 SR 协议:	14
5. 问题讨论.....	15
6. 心得体会.....	16

1.实验目的

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。

2.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；（选作内容，加分项目，可以当堂完成或课下完成）
- 4) 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）

3.实验过程

3.1.GBN 协议数据格式

对一个字节数组进行封装，封装成一个类，如下：

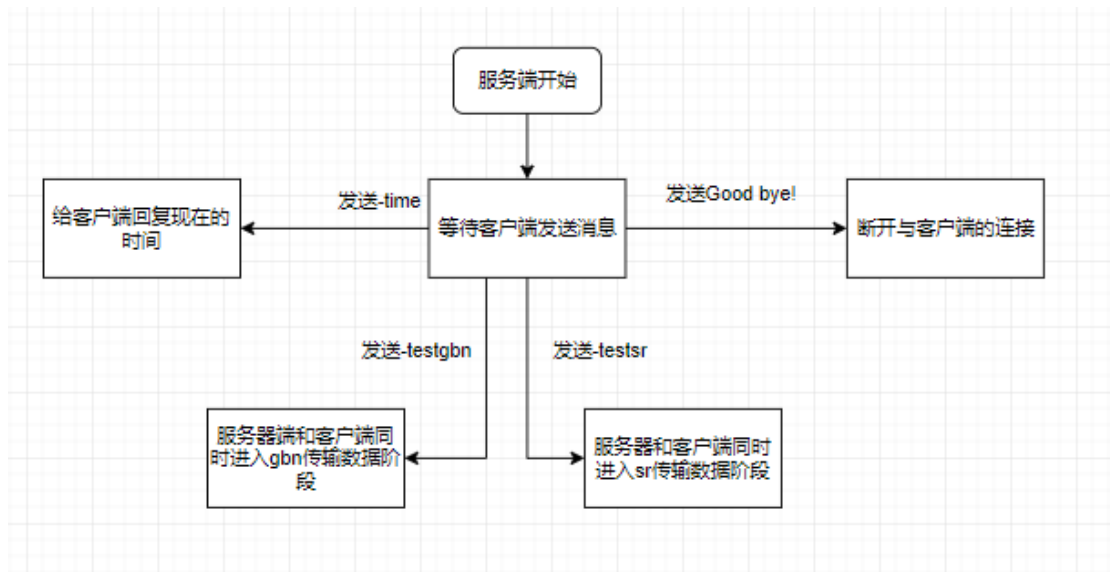
```
// 第一位字节为 seq,其余位为数据 data
private byte[] allData = new byte[1471];
```

3.2.确认分组格式

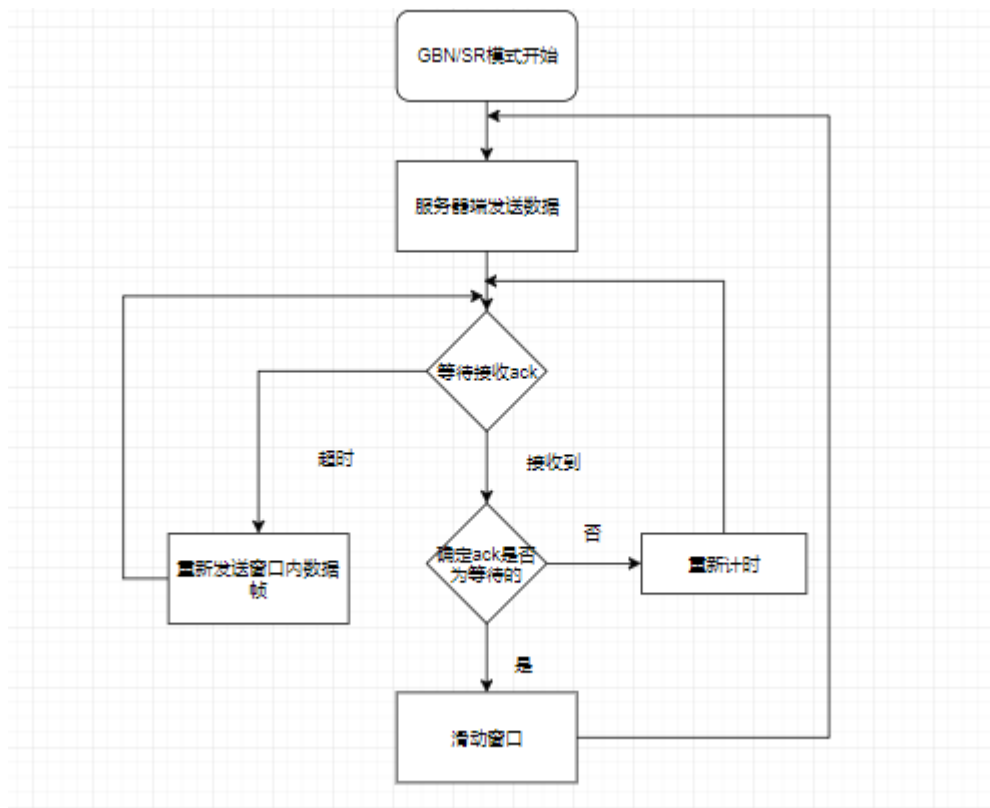
直接使用一个字节当作 ack 分组格式。

3.3 基本流程图

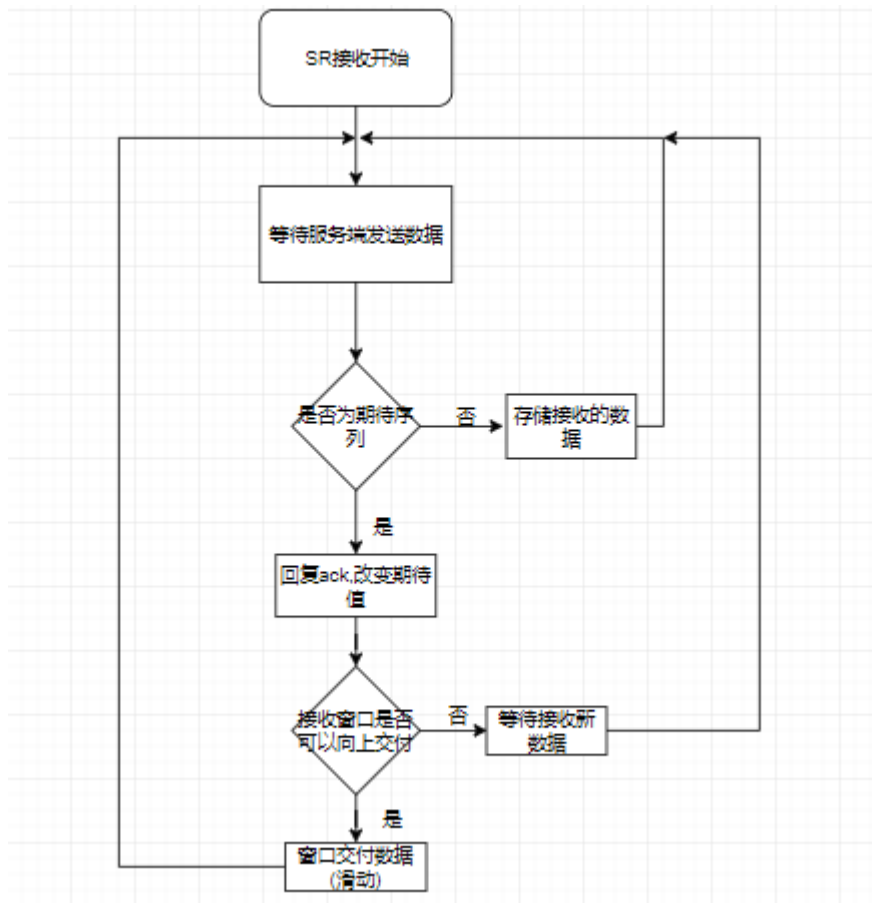
1. 服务器端大致流程



2. GBN/SR 传输具体流程

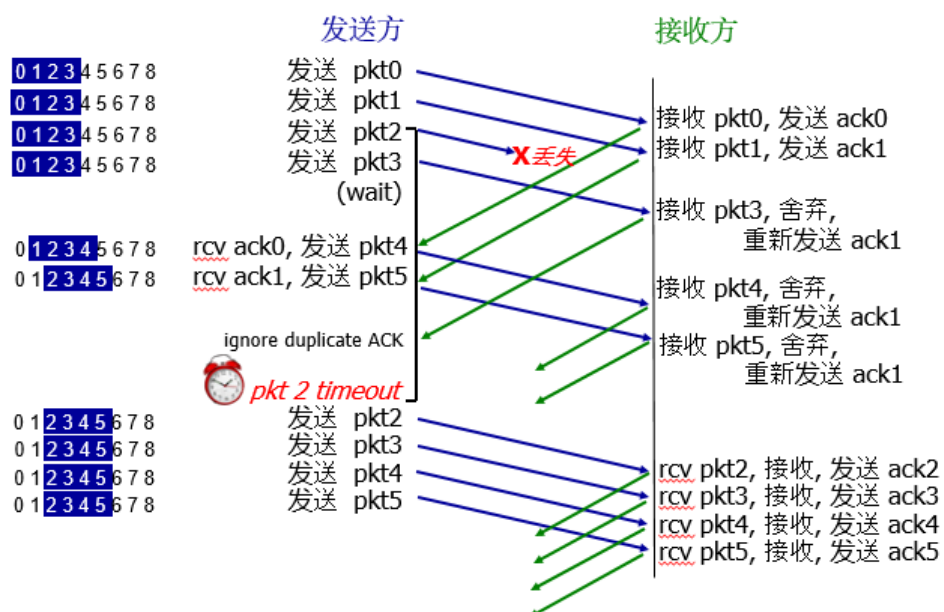


3. SR 具体接收流程

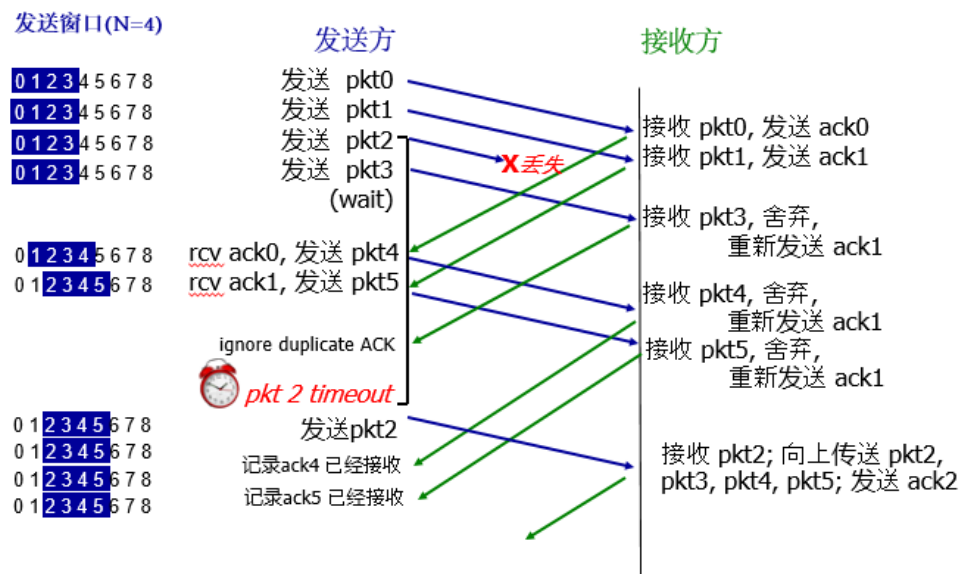


3.4 协议典型交互过程

1. GBN 协议



4. SR 协议



3.5 数据分组丢失验证模拟方法

1. 丢失模拟:

通过使用整除, 当发送数据的序号可以被某个数整除就实际不发送, 但是标记上标记发送。如下:

```
if (i % 3 == 1) {
    window.setNextseqnum(window.getNextseqnum() + 1);
    System.out.println("首次发送, 模拟第" + (i + window.getBase()) + "个数据丢失:"
        + frames[i + window.getBase()].getStrData());
    continue;
}
```

2. 验证丢失:

在接收端使用红色字体打印丢弃的分组, 使用白字打印接收的分组。如下:

```
if (expect == frame.getSeq()) {
    sendAck(frame.getSeq());
    System.out.println("收到分组:" + frame.getSeq() + "内容为:"
        + frame.getStrData() + ", 该分组为期待的分组");
    expect++;
} else {
    System.err.println("收到分组:" + frame.getSeq() + "内容为:"
        + frame.getStrData() + ", 该分组不是期待分组:" + expect);
    sendAck(expect - 1);
}
```

3.6 程序实现的主要类及主要函数

1. 发送窗口类:

模拟发送窗口，具有滑动方法等滑动窗口的动作。如下：

```
/**
 * 该类为发送窗口的类，用来模拟在使用GBN或者SR协议时发送方维护的窗口
 * 其中由一个byte[]数组用来记录数据，一个Wsize用来记录窗口的大小，一个base用来标注等待Ack确认的最小序号
 * 一个nextseqnum用来记录窗口中还未发送的最小序号在window窗口中的位置，即索引，而不是像base一样为序列号
 *
 * @author MaXU
 *
 */
public class SendWindow {
    int seq = StaticData.MAX_SEQ;

    private byte[] window; // 用来记录窗口；1表示发送且已经确认；其他情况为0

    private int Wsize = 0; // 用来记录记录窗口的大小
    private int base = 0; // 正在等待ack确认的最小的序号
    private int nextseqnum = 0; // 用来记录窗口可用还未发送的最小序号,但是在是现实时记录与base的相对距离

    public SendWindow(int size) {
        this.Wsize = size;
        this.window = new byte[size];
    }
}
```

5. 接收窗口类

一个模拟 SR 协议的接收窗口而创建的类，具有接收窗口的动作，如下：

```
public class ReceiveWindow {
    int maxSeq = StaticData.MAX_SEQ;

    private byte[] window;
    private int Wsize = 0;
    private int base = 0;

    public ReceiveWindow(int size) {
        this.Wsize = size;
        this.window = new byte[size];
    }
}
```

6. 服务器类 EchoServer

一个服务器类主要用来当作发送文件的服务器端。其主要具备下面几个函数：

- (1). 根据接收数据决定进行特定的处理，receive 函数

```

public void receive() throws IOException {
    String msg = new String(packet.getData(), 0, packet.getLength());
    if (msg.equals("-time")) {
        packet.setData(nowTime().getBytes());
        socket.send(packet);
    } else if (msg.equals("-quit")) {
        packet.setData(StaticData.SERVER_BYE.getBytes());
        socket.send(packet);
    } else if (msg.equals("-testgbn")) {
        this.inGBN = 1;
    } else if (msg.equals("-testsr")) {
        this.inGBN = 2;
    } else {
        packet.setData(msg.getBytes());
        socket.send(packet);
    }
}
}

```

(2). 进入 gbn 模式，用于处理对应的 gbn 请求，ingbn 函数。

```

//进入gbn模式
public void ingbn() throws IOException {
    this.frames = utils.geneFrame((byte) num);
    this.window = new SendWindow(wsize);

    timer.start();
    while (true) {
        //如果有可发送的数据就发送
        if (window.getNextseqnum() + window.getBase() != num) {
            sendData();
        }
        //接收客户端发来的数据
        DatagramPacket packet = new DatagramPacket(new byte[1471], 1471);
        socket.receive(packet);
        String msg = new String(packet.getData());
        byte b = msg.getBytes()[0];

        if (b == window.getBase()) {
            //如果是期待的ack就滑动窗口
            window.slipN(1);
            System.out.println("接收到的ack序号:" + b);
            System.out.println("滑动后,base:" + window.getBase() + ",nextseqnum:"
                + window.getNextseqnum());
            timer.setTime(0);
        } else {
            //否则重新计时
            timer.setTime(time);
        }
        if (b == num) {
            timer.interrupt();
            break;
        }
    }
}
}

```

(3). 进入 sr 模式，用于处理对应的 sr 请求，insr 函数。


```

//进入sr模式
public void insr() throws IOException {
    this.frames = utils.geneFrame((byte) num);
    this.window = new SendWindow(wsize);

    timer.start();
    while (true) {           //如果有可以发送的数据就发送
        if (window.getNextseqnum() + window.getBase() != num) {
            sendData();
        }
        //接收客户端发来的数据
        DatagramPacket packet = new DatagramPacket(new byte[1471], 1471);
        socket.receive(packet);
        String msg = new String(packet.getData());
        byte b = msg.getBytes()[0];

        window.setAckBySeq(b);
        if(window.canSlip()) {           //如果窗口可以滑动就滑动
            window.slip();
            System.out.println("滑动后,base:" + window.getBase() + ",nextseqnum:"
                + window.getNextseqnum());
            timer.setTime(time);
        }

        if (b == num) {
            timer.interrupt();
            break;
        }
    }
}

```

(4). 发送数据函数，用于将数据发送给客户端，sendData 函数。

```

//发送数据方法
public void sendData() throws IOException {
    for (int i = window.getNextseqnum(); i < window.getWsize()
        && i + window.getNextseqnum() < num; i++) {

        if (i % 3 == 1) {           //模拟数据丢失
            window.setNextseqnum(window.getNextseqnum() + 1);
            System.out.println("首次发送,模拟第" + (i + window.getBase()) + "个数据丢失:"
                + frames[i + window.getBase()].getStrData());
            continue;
        }
        window.setNextseqnum(window.getNextseqnum() + 1); //更新nextseqnum
        packet.setData(frames[i + window.getBase()].getAllData());
        socket.send(packet);

        System.out.println("首次发送,第" + (i + window.getBase()) + "个数据已经发送:"
            + frames[i + window.getBase()].getStrData());
    }
    timer.setTime(time);
}

```

(5). 超时处理函数，用于超时时对数据进行发送。timeout 函数：

```
//超时调用的方法
public void timeout() throws IOException {
    for (int i = 0; i < window.getNextseqnum(); i++) {
        if (inGBN == 2) { //如果是sr就不发送已经收到ack的数据
            if (window.getAckOfn(i) == 1) {
                continue;
            }
        }
        packet.setData(frames[i + window.getBase()].getAllData());
        socket.send(packet);
        System.out.println("重发, 第" + (window.getBase() + i) + "个数据已经重发");
    }
    timer.setTime(time);
}
}
```

7. 客户端类 EchoClient

一个客户端类用于模拟客户端用于接收数据。其主要方法如下：

(1). 根据接收数据决定进行特定的处理，receive 函数

```
public void receive() throws IOException {
    String msg = new String(packet.getData(), 0, packet.getLength());
    if (msg.equals("-time")) {
        packet.setData(nowTime().getBytes());
        socket.send(packet);
    } else if (msg.equals("-quit")) {
        packet.setData(StaticData.SERVER_BYE.getBytes());
        socket.send(packet);
    } else if (msg.equals("-testgbn")) {
        this.inGBN = 1;
    } else if (msg.equals("-testsr")) {
        this.inGBN = 2;
    } else {
        packet.setData(msg.getBytes());
        socket.send(packet);
    }
}
}
```

(2). 进入 gbn 模式，用于处理对应的 gbn 请求，testgbn 函数。

```
public void testgbn() throws IOException {
    while (true) {

        DatagramPacket inputPacket = new DatagramPacket(new byte[1471], 1471);

        socket.receive(inputPacket);
        UDPFrame frame = new UDPFrame();
        frame.setAllData(inputPacket.getData());
        if (expect == frame.getSeq()) { //如果是期待的数据就向上交付并发送确认ack
            sendAck(frame.getSeq());
            System.out.println("收到分组:" + frame.getSeq() + "内容为:"
                + frame.getStrData() + ",该分组为期待的分组");
            expect++;
        } else { //否则发送上一个接收的ack序号
            System.err.println("收到分组:" + frame.getSeq() + "内容为:"
                + frame.getStrData() + ",该分组不是期待分组:" + expect);
            sendAck(expect - 1);
        }
        if (expect == num) {
            break;
        }
    }
}
}
```

(3). 进入 sr 模式，用于处理对应的 sr 请求，insr 函数。

```
public void testSr() throws IOException {
    int sum = 0;
    while (true) {
        DatagramPacket inputPacket = new DatagramPacket(new byte[1471], 1471);

        socket.receive(inputPacket);
        UDPFrame frame = new UDPFrame();
        frame.setAllData(inputPacket.getData());
        this.window.setAck(frame.getSeq());
        sendAck(frame.getSeq()); //得到接收的数据的序号
        if(this.window.canRcv()) { //如果可以向上交付就交付
            int m = this.window.rcv();
            sum+=m;
            System.out.println("向上交付数据,序号为"+(this.window.getBase()-m)+"~"+(this.window.getBase()-1));
        }
        if(sum == this.num) {
            break;
        }
    }
}
```

(4). 回复 ack 函数，向发送方回复 ack

```
public void sendAck(int i) throws IOException {
    byte[] response = {(byte) i};
    DatagramPacket packet = new DatagramPacket(response, response.length,
        remoteIP, remotePort);
    socket.send(packet);
}
```

8. 计时器类 Timer

用于发送时的计时。

```

public class Timer extends Thread{
    public EchoServer server;
    int time = 0;
    public Timer(EchoServer server,int time) {
        this.server = server;
        this.time = time;
    }

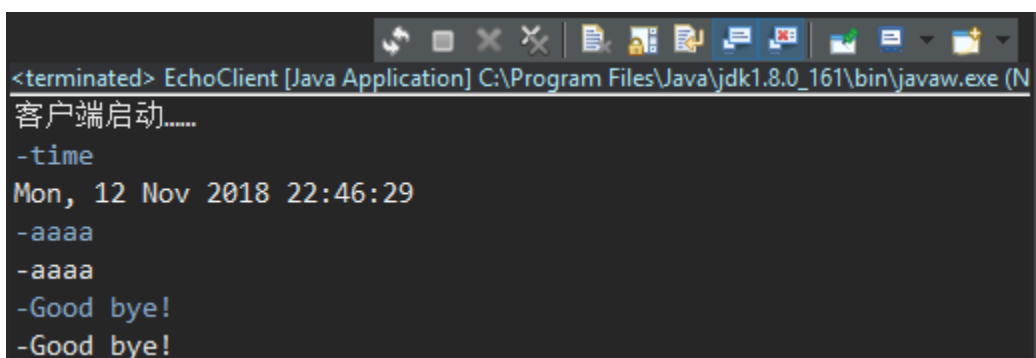
    @Override
    public void run() {
        do {
            if(time>0) {
                try {
                    Thread.sleep(time*1000);
                    server.timeout();
                    System.out.println("时间超时，已重新发送.....");
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }while(true);
    }

    public void setTime(int time) {
        this.time = time;
    }
}

```

4.实验结果

4.1.基本功能实现：



```

<terminated> EchoClient [Java Application] C:\Program Files\Java\jdk1.8.0_161\bin\javaw.exe (N
客户端启动.....
-time
Mon, 12 Nov 2018 22:46:29
-aaaa
-aaaa
-Good bye!
-Good bye!

```

4.2.实现 GBN 协议：

窗口大小：10

帧个数：15

运行结果如下：

```
客户端启动.....
-testgbn
收到分组:0内容为:the 0 data
收到分组:2内容为:the 2 data
收到分组:3内容为:the 3 data
收到分组:5内容为:the 5 data
收到分组:6内容为:the 6 data
收到分组:1内容为:the 1 data
收到分组:4内容为:the 4 data

收到分组:5内容为:the 5 data
收到分组:7内容为:the 7 data
收到分组:8内容为:the 8 data
收到分组:9内容为:the 9 data
收到分组:10内容为:the 10 data
收到分组:3内容为:the 3 data
收到分组:4内容为:the 4 data
收到分组:5内容为:the 5 data
收到分组:7内容为:the 7 data
收到分组:8内容为:the 8 data
收到分组:9内容为:the 9 data
收到分组:10内容为:the 10 data
收到分组:11内容为:the 11 data
收到分组:6内容为:the 6 data
收到分组:7内容为:the 7 data
收到分组:8内容为:the 8 data
收到分组:10内容为:the 10 data
收到分组:11内容为:the 11 data

收到分组:10内容为:the 10 data
收到分组:13内容为:the 13 data
收到分组:14内容为:the 14 data
收到分组:9内容为:the 9 data
收到分组:10内容为:the 10 data
收到分组:13内容为:the 13 data
收到分组:14内容为:the 14 data
收到分组:11内容为:the 11 data
收到分组:13内容为:the 13 data
收到分组:14内容为:the 14 data
收到分组:12内容为:the 12 data
收到分组:13内容为:the 13 data
收到分组:14内容为:the 14 data
```

其中红字表示被舍弃的数据，白字表示被正确接收的数据。

```
服务器启动.....
首次发送,第0个数据已经发送:the 0 data
首次发送,模拟第1个数据丢失:the 1 data
首次发送,第2个数据已经发送:the 2 data
首次发送,第3个数据已经发送:the 3 data
首次发送,模拟第4个数据丢失:the 4 data
首次发送,第5个数据已经发送:the 5 data
首次发送,第6个数据已经发送:the 6 data
首次发送,模拟第7个数据丢失:the 7 data
接收到的ack序号:0
滑动后,base:1,nextseqnum:7
首次发送,模拟第8个数据丢失:the 8 data
重发,第1个数据已经重发
接收到的ack序号:1
滑动后,base:2,nextseqnum:7
重发,第3个数据已经重发
重发,第4个数据已经重发
首次发送,模拟第9个数据丢失:the 9 data
重发,第5个数据已经重发
接收到的ack序号:2
滑动后,base:3,nextseqnum:7
重发,第7个数据已经重发
首次发送,模拟第10个数据丢失:the 10 data
重发,第8个数据已经重发
重发,第9个数据已经重发
重发,第10个数据已经重发
时间超时,已重新发送.....
重发,第11个数据已经重发
重发,第3个数据已经重发
重发,第4个数据已经重发
重发,第5个数据已经重发
接收到的ack序号:3
滑动后,base:4,nextseqnum:7
重发,第7个数据已经重发
重发,第8个数据已经重发
重发,第9个数据已经重发
首次发送,模拟第11个数据丢失:the 11 data
重发,第10个数据已经重发
接收到的ack序号:4
滑动后,base:5,nextseqnum:7
首次发送,模拟第12个数据丢失:the 12 data
接收到的ack序号:5
滑动后,base:6,nextseqnum:7
首次发送,模拟第13个数据丢失:the 13 data
重发,第13个数据已经重发
时间超时,已重新发送.....
重发,第6个数据已经重发
重发,第7个数据已经重发
重发,第8个数据已经重发
接收到的ack序号:6
滑动后,base:7,nextseqnum:7
重发,第10个数据已经重发
首次发送,模拟第14个数据丢失:the 14 data
接收到的ack序号:7
滑动后,base:8,nextseqnum:7
接收到的ack序号:8
```

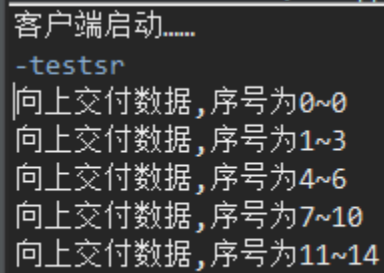
这是服务器端打印的发送数据。

4.3.实现 SR 协议:

窗口大小: 10

帧个数: 15

客户端接收数据情况如下:



```
客户端启动.....  
-testsr  
向上交付数据,序号为0~0  
向上交付数据,序号为1~3  
向上交付数据,序号为4~6  
向上交付数据,序号为7~10  
向上交付数据,序号为11~14
```

服务器端打印如下:

```
terminated. Enter server path/application, Ctrl+C to quit. Press any key to continue.
服务器启动.....
首次发送,第0个数据已经发送:the 0 data
首次发送,模拟第1个数据丢失:the 1 data
首次发送,第2个数据已经发送:the 2 data
首次发送,第3个数据已经发送:the 3 data
首次发送,模拟第4个数据丢失:the 4 data
首次发送,第5个数据已经发送:the 5 data
首次发送,第6个数据已经发送:the 6 data
首次发送,模拟第7个数据丢失:the 7 data
滑动后,base:1,nextseqnum:7
首次发送,模拟第8个数据丢失:the 8 data
重发,第1个数据已经重发
重发,第4个数据已经重发
滑动后,base:4,nextseqnum:5
重发,第8个数据已经重发
重发,第9个数据已经重发
时间超时,已重新发送.....
首次发送,第9个数据已经发送:the 9 data
首次发送,第10个数据已经发送:the 10 data
首次发送,模拟第11个数据丢失:the 11 data
滑动后,base:7,nextseqnum:5
首次发送,第12个数据已经发送:the 12 data
首次发送,第13个数据已经发送:the 13 data
首次发送,模拟第14个数据丢失:the 14 data
重发,第7个数据已经重发
滑动后,base:11,nextseqnum:4
重发,第14个数据已经重发
时间超时,已重新发送.....
重发 第11个数据已经重发
```

5.问题讨论

1. 在 SR 协议中,发送窗口大小和接收窗口大小的和要小于最大的序列号,否则在接收端会出现不知道具体接收的序号是不是需要接收,就会出现序号错乱的现象。
2. 在实验中,窗口中设置 ack 时,因为会出现延时到达的 ack 序号不在窗口中的情况,这样就会导致出现数组越界的错误。这样的话在设置 ack 时就需要综合考虑各种情况,尤其是 $base + size > max_seq$; 在进行判断时需要分段判断。
3. 计时器的使用,如果每个帧都使用一个计时器就会非常浪费线程,并且难以维护,所以就在整个文件传输中只使用了一个计时器。

6.心得体会

1. 通过实验对 GBN 协议和 SR 协议有了更加深入的了解。
2. 通过 Java 中的 UDP 编程有了一些了解。
3. 在写实验时首先要把需要的类写好，然后再将整体的框架写好，否则在实现主要功能时会因为个别需要的类没有写好而束手束脚。