

# 哈尔滨工业大学

## <<计算机网络>>

### 实验报告

(2019 年度秋季学期)

姓名：	陈鋆
学号：	1170300513
学院：	计算机科学与技术学院
教师：	聂兰顺

## 实验二 可靠数据传输的设计与实现

### 一、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。

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。

### 二、实验内容

#### 1.可靠数据传输协议-停等协议的设计与实现

- 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。

#### 2.可靠数据传输协议-GBN 协议的设计与实现

- 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；
- 4) 将所设计的 GBN 协议改进为 SR 协议。

### 三、实验过程及结果

#### GBN 协议：

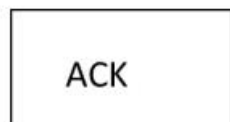
##### GBN 协议数据分组格式：

Seq	Data
-----	------

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等于 1472 字节。

本实验中将数据帧的大小恰好设为 1472 字节，其中 Seq 段对应着序列号，大小为 3 个 Byte，Data 段大小恒为 1469Byte，故所以 UDP 数据报的数据部分应等于 1472 字节。

##### 确认分组格式：



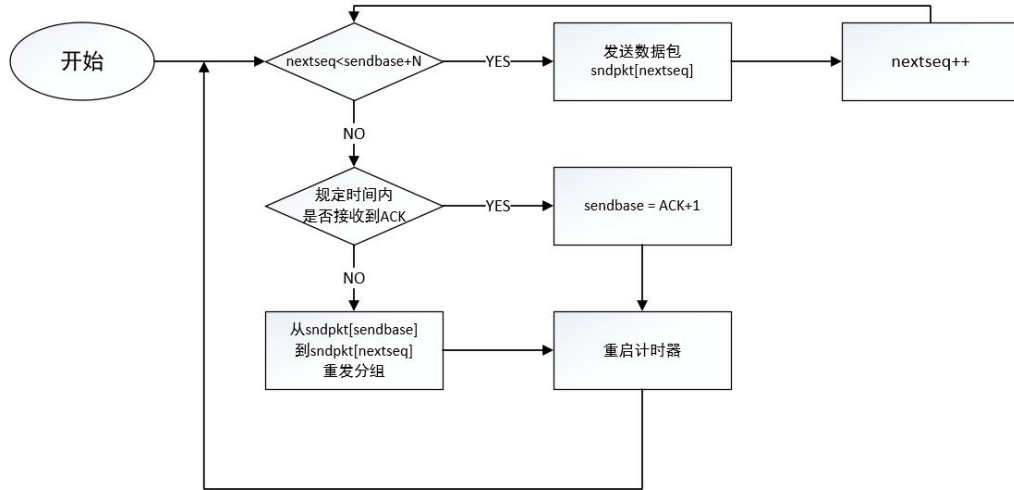
由于是从服务器端到客户端的单向数据传输，因此 ACK 数据帧不包含任何数据，只需要将 ACK 发送给服务器端即可。

ACK 字段用于表示序列号，大小设定为 10 个字节。

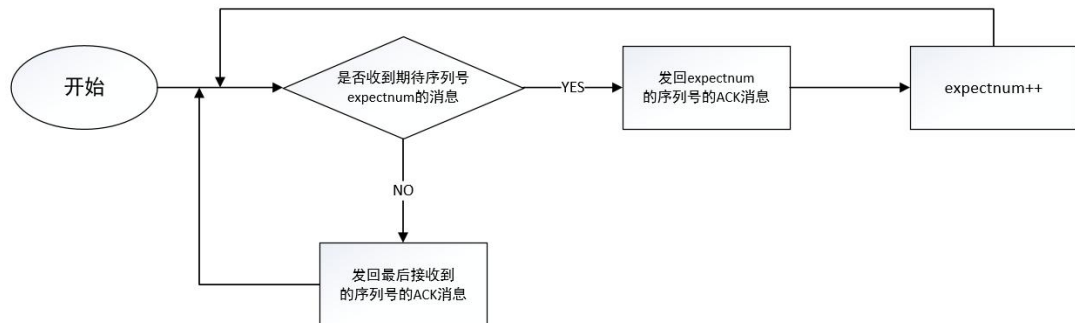
```
public class ACK {  
    protected int ACK;  
    protected String ack;  
    protected byte[] ackByte;  
  
    public ACK(int ACK) {  
        this.ACK=ACK;  
        ack=String.valueOf(ACK);  
        ackByte=ack.getBytes();  
    }  
}
```

### 协议两端程序流程图：

发送端：



接收端：



### 协议典型交互过程：

设窗口大小为 N

send\_base 是已发送未确认 ACK 的分组

nextsequnm 是下一个要发送的数据包

expectseq 是期望收到的数据包序号

**发送端：初始化** send\_base=1,nextseqnum=1

1. 发送方发送数据过程

(1) 如果有条件  $\text{nextseqnum} < \text{send\_base} + N$  成立，则发送数据包  $\text{Data}[\text{nextseqnum}]$ ，并使  $\text{seqnum}++$ 。

(2) 重启计时器。

2. 发送方接收到 ACK

(1) 得知获得的 ACK 的序列号  $\text{ack}$ 。

(2)  $\text{send\_base}=\text{ack}+1$ 。

(3) 如果  $\text{send\_base}$  与  $\text{nextseqnum}$  相等，关闭计时器。否则重启计时器。

3. 发送方的计时器超时

(1) 重发分组从  $\text{Data}[\text{send\_base}]$  到  $\text{Data}[\text{nextseqnum}-1]$

(2) 重启计时器

**接收端：初始化** expectseq=1

接收方接收到数据包：

提取数据包的序列号，判断是否等于 **expectseq**。如果相等，则发送回该序列号的 ACK 报文；如果不等，则发送上一次受到的序列号的 ACK 报文。

**数据分组丢失验证模拟方法：**

在发送数据包时，将数据包的序列号取模，如序号为  $x$ ，如果  $x\%5==0$ ，则不发送该数据包。

```

if (nextseqnum % 5 != 0)
{
    SenderSocket.send(SendDataPacket);
    System.out.println("发送分组:" + nextseqnum%seqnum+";发送的是第"+nextseqnum+"个包");
}
else
{
    System.out.println("模拟分组" + nextseqnum%seqnum + "丢失"+"丢失的是第"+nextseqnum+"个包");
}

```

在接收 ACK 处，将 ACK 的序列号也取模，如序号为 xa，如果  $xa \% 6 == 0$ ，则不接收该 ACK。

```

if (ack % 6 != 0)
{
    System.out.println("接收到ACK" + ack);
    int m;
    //越序号列的情况.
    if (((send_base%seqnum)>ack)&&((nextseqnum/seqnum)>(send_base/seqnum))&&(ack<=((send_base+N)%N)))
    {
        m=send_base/seqnum*seqnum+ack+seqnum+1;
    }
    else
    {
        m=send_base/seqnum*seqnum+ack+1;
    }
    send_base = Math.max(send_base, m);
}
else
{
    System.out.println("模拟ACK" + ack + "丢失");
}

```

### 程序实现的主要类（或函数）及其主要作用：

**TimeBegin() 函数：**开始计时或者重新计时，还包含者重发的功能，超时时间为 2s。利用 ScheduledExecutorService 类来进行计时，一旦超时，则重发分组从 Data[send\_base]到 Data[nextseqnum-1]，并重新启动计时器。

```

public void TimeBegin()
{
    TimerTask task=new TimerTask()
    {
        @Override
        public void run()
        {
            //UDP数据包每次能够传输的最大长度 = MTU(1500B) - IP头(20B) -UDP头 (8B) = 1472Bytes
            //没有信息发送时就返回。
            if (send_base>= Math.ceil(B.length / 1469)) //取大于等于的最近整数
            {
                executor.shutdown();
                return;
            }
        }
    }
}

```

```

try
{
    //重新开始计时并发送

    for (int i = send_base; i < nextseqnum; i++)
    {
        byte[] tempb = GetPart(i);
        String temp = new String(tempb);
        String s = new String(i%seqnum + ":" + temp);
        String s;
        if(i%seqnum < 10)
        {
            s = new String("0" + i%seqnum + ":" + temp);
        }
        else
        {
            s = new String(i%seqnum + ":" + temp);
        }
        byte[] data = s.getBytes();
        DatagramPacket SenderPacket = new DatagramPacket(data, data.length, InetAddress.getLocalHost(), SendDataPort);
        //接收端地址也是本地
        SenderSocket.send(SenderPacket);
        System.out.println("重发分组:" + i%seqnum+";重发的是第"+i+"个包");
        //TimeBegin();
    }
}

```

**TimeReset() 函数与 TimeEnd() 函数：**负责重置计时器（接收到 ACK 后的一个动作）与关闭计时器。

```

/**
 * 结束计时
 */
public void TimeEnd()
{
    if (flag)
    {
        executor.shutdown();
        flag = false;
    }
}

/**
 * 重置计时器
 */
public void TimeReset()
{
    if (send_base == nextseqnum)
    {
        TimeEnd();
    }
    else
    {
        TimeBegin();
    }
}

```

**ReadFilebyLines 函数与 GetPart 函数：**一个负责读取文件并将其转为比特数组，另一个则负责将比特数组按规定的数据段大小（1469B）进行分割以便于装入 GBN 数据分组中。

```

public static void ReadFilebyLines(String filename)
{
    filename = "src\\lab2\\"+filename;
    File file = new File(filename);
    BufferedReader reader = null;
    try
    {
        reader = new BufferedReader(new FileReader(file));
        String tempstr = null;
        while ((tempstr = reader.readLine()) != null)
        {
            filestr += tempstr + "\r\n";
        }
        reader.close();
        //将文件转为比特存储于B中
        B = filestr.getBytes();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
    finally
    {
        if (reader != null)
        {
            try
            {
                reader.close();
            }
            catch (IOException e1)
            {
            }
        }
    }
}

```

```

public byte[] GetPart(int nextseqnum)
{
    //切割得到数据段
    byte[] temp = new byte[1469];
    for (int i = 0; i < 1469; i++)
    {
        if (nextseqnum * 1469 + i >= B.length)
        {
            break;
        }
        temp[i] = B[nextseqnum * 1469 + i];
    }
    return temp;
}

```

**Send()**：发送端的核心函数，负责将文件传输完的整个过程。在其内通过 while 循环，在发送窗口有空闲时不断发送数据包，并接收 ACK 报文，直至确认全文发送完毕后关闭计时器并退出。

```

public void Send()
{
    ReadFilebyLines("data.txt");
    team=(int) Math.ceil(B.length/1469);
    try
    {
        SenderSocket = new DatagramSocket();
        ReceiveAckSocket = new DatagramSocket(ReceiveAckPort);
        while (true)
        {
            SendtoReciver();
            ReceiveACK();
            if (send_base >= team)
            {
                break;
            }
        }
        //数据传输完毕后不要忘了关闭计时器
        executor.shutdown();
        System.out.println("Send Over");
    }
    catch (Exception e) {
    }
}

```

**SendtoReciver()** 函数：发送过程的核心函数，模拟一次发送窗口有空闲时的发送。其中包括了判断数据是否发送完的步骤、调用 GetPart 函数取得数据分块并装配序列号组成 GBN 协议数据分组并发送、模拟数据包丢失、调用 TimeBegin



函数开始计时的过程。

```
public void SendtoReceiver()
{
    try
    {
        //以窗口长度为判定界限.
        while (nextseqnum < send_base + N)
        {
            if (send_base >= team || nextseqnum >= team)
            {
                break;
            }
            byte[] tempb = GetPart(nextseqnum);
            String temp = new String(tempb);
            //给数据段添加上标记序列号的首部
            String s;
            if (nextseqnum % seqnum < 10)
            {
                s = new String("0" + nextseqnum % seqnum + ":" + temp);
            }
            else
            {
                s = new String(nextseqnum % seqnum + ":" + temp);
            }
            byte[] data = s.getBytes();
            SendDataPacket = new DatagramPacket(data, data.length, InetAddress.getLocalHost(), SendDataPort);
            // 模拟数据包丢失
            if (nextseqnum % 5 != 0)
            {
                SenderSocket.send(SendDataPacket);

                System.out.println("发送分组:" + nextseqnum % seqnum + ";发送的是第" + nextseqnum + "个包");
            }
            else
            {
                System.out.println("模拟分组" + nextseqnum % seqnum + "丢失" + "丢失的是第" + nextseqnum + "个包");
            }
            if (send_base == nextseqnum) //重启计时器
            {
                TimeBegin();
            }
            nextseqnum++;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

**ReceiveACK()**：用于处理接收 ACK 信息过程的函数。包含了提取 ACK 序列号、模拟 ACK 丢包等过程。

```

public void ReceiveACK() throws InterruptedException
{
    try {
        //发送完数据的情况
        if (send_base >= team)
        {
            return;
        }
    }
    //ACK消息的大小为10bytes.
    byte[] bytes = new byte[10];
    ReceiverAckPacket = new DatagramPacket(bytes, bytes.length);
    ReceiverAckSocket.receive(ReceiverAckPacket);
    String ackString = new String(bytes, 0, bytes.length);
    String acknum = new String();
    for (int i = 0; i < ackString.length(); i++)
    {
        //取出ACK信息中的ACK序列号
        if (ackString.charAt(i) >= '0' && ackString.charAt(i) <= '9')
        {
            acknum += ackString.charAt(i);
        }
        else
        {
            break;
        }
    }
    int ack = Integer.parseInt(acknum);
    // 模拟ACK丢包
    if (ack % 6 != 0)

```

**Receive() 函数：**接收端的核心函数，负责不断接收数据包，并提取中其中的序列号，再将其与期待收到的序列号进行对比，如果符合就发送相应的 ACK 信息，否则发送上一次接收到的序列号的 ACK 信息。

```

public void Receive() {
    try {
        ReceiverSocket = new DatagramSocket(ReceiveDataPort);
        while (true)
        {
            byte[] data = new byte[1472];

            DatagramPacket packet = new DatagramPacket(data, data.length);
            ReceiverSocket.receive(packet);
            byte[] d = packet.getData();
            String message = new String(d);
            String num = new String();
            //取出该包的序列号
            for (int i = 0; i < message.length(); i++)
            {
                if (message.charAt(i) <= '9' && message.charAt(i) >= '0')
                {
                    num = num + message.charAt(i);
                } else {
                    break;
                }
            }
            // 进行累积确认，不是想要的序号段的话直接丢弃
            if (expectedSeqNum == Integer.valueOf(num))
            {
                int ack = expectedSeqNum;
                SendACK(ack);
                expectedSeqNum = (expectedSeqNum + 1) % seqnum;
                last = ack;
            }
        }
    }
}

```

**SendACK 函数：**负责发回 ACK

```
public void SendACK(int ack) {  
    try  
    {  
        ACK ACK = new ACK(ack);  
        SendAckPacket = new DatagramPacket(ACK.ackByte, ACK.ackByte.length, InetAddress.getLocalHost(), SendAckPort);  
        ReceiverSocket.send(SendAckPacket);  
        System.out.println("发回ACK" + ack);  
    } catch (Exception e)  
    {}  
}
```

### 实验验证结果:

#### 模拟分组丢失:

模拟分组0丢失 丢失的是第0个包

#### 模拟 ACK 信息丢失:

模拟ACK0丢失

#### 发送后计时器到时的重发:

模拟分组0丢失 丢失的是第0个包  
发送分组:1;发送的是第1个包  
发送分组:2;发送的是第2个包  
发送分组:3;发送的是第3个包  
发送分组:4;发送的是第4个包  
模拟分组5丢失 丢失的是第5个包  
发送分组:6;发送的是第6个包  
发送分组:7;发送的是第7个包  
重发分组:0;重发的是第0个包  
重发分组:1;重发的是第1个包  
重发分组:2;重发的是第2个包  
重发分组:3;重发的是第3个包  
重发分组:4;重发的是第4个包  
重发分组:5;重发的是第5个包  
重发分组:6;重发的是第6个包  
重发分组:7;重发的是第7个包

#### 接收到 ACK 后的窗口滑动:

```
重发分组:0;重发的是第0个包
重发分组:1;重发的是第1个包
重发分组:2;重发的是第2个包
重发分组:3;重发的是第3个包
重发分组:4;重发的是第4个包
重发分组:5;重发的是第5个包
重发分组:6;重发的是第6个包
重发分组:7;重发的是第7个包
模拟ACK0丢失
接收到ACK1
发送分组:8;发送的是第8个包
发送分组:9;发送的是第9个包
```

未接收到期待接收到的序列号的情况（包 10 丢失而后续未丢失）：

```
发回ACK9
发回ACK9
发回ACK9
发回ACK9
发回ACK9
发回ACK9
发回ACK9
发回ACK10

发送分组:8;发送的是第8个包
发送分组:9;发送的是第9个包
接收到ACK2
模拟分组10丢失 丢失的是第10个包
接收到ACK3
发送分组:11;发送的是第11个包
接收到ACK4
发送分组:12;发送的是第12个包
接收到ACK5
发送分组:13;发送的是第13个包
模拟ACK6丢失
接收到ACK7
发送分组:14;发送的是第14个包
模拟分组15丢失 丢失的是第15个包
接收到ACK8
发送分组:0;发送的是第16个包
接收到ACK9
发送分组:1;发送的是第17个包
接收到ACK9
接收到ACK9
接收到ACK9
接收到ACK9
接收到ACK9
接收到ACK9
重发分组:10;重发的是第10个包
```

发送完毕：

接收到ACK1  
接收到ACK2  
重发分组:2;重发的是第18个包  
重发分组:3;重发的是第19个包  
接收到ACK3  
Send Over

双工:

发回ACK12  
发回ACK13  
发回ACK14  
发回ACK15  
发回ACK0  
发回ACK1  
发回ACK2  
发回ACK3  
重发分组:0;重发的是第0个包  
模拟ACK0丢失  
重发分组:1;重发的是第1个包  
接收到ACK1  
发送分组:8;发送的是第8个包  
重发分组:2;重发的是第2个包  
发送分组:9;发送的是第9个包  
接收到ACK2  
模拟分组10丢失 丢失的是第10个包  
接收到ACK2  
重发分组:3;重发的是第3个包  
接收到ACK2  
接收到ACK3

详细注释源程序:

```
public class GBNS implements Runnable{

    private static int seqnum=16;

    private static int N = 8;

    public GBNS() {
        super();
    }

    @Override
    public void run() {

        Send();

    }

    public static void main(String[] args) {
        new Thread(new GBNS()).start();
    }
}
```

```

}

// 发送数据部分
private static int SendDataPort = 10241; //接收方端口号
private static int ReceiveAckPort = 10240; //发送方端口号
private static DatagramSocket SenderSocket;
private static DatagramPacket SendDataPacket;
private static DatagramSocket ReceiveAckSocket;
private static DatagramPacket ReceiverAckPacket;
private static int send_base = 0;
private static int nextseqnum = 0;
private static boolean flag = false; //是否开始计时的标志
private static int timeout = 2; //超时时间定为 2s
private static String filestr = new String();
private static byte[] B;
private static int team;

private ScheduledExecutorService executor;

/**
 * 开始计时或者重新计时，超时时间为 2s
 */
public void TimeBegin()
{
    TimerTask task = new TimerTask()
    {
        @Override
        public void run()
        {
            //UDP 数据包每次能够传输的最大长度 = MTU(1500B) - IP 头(20B) - UDP
            //头(8B) = 1472Bytes
            //没有信息发送时就返回。
            if (send_base >= Math.ceil(B.length / 1469) - 1) //取大于等于的最
            近整数
            {
                executor.shutdown();
                return;
            }
            try
            {
                //重新开始计时并发送

```

```

        for (int i = send_base; i < nextseqnum; i++)
        {
            byte[] tempb = GetPart(i);
            String temp = new String(tempb);
//            String s = new String(i%seqnum + ":" + temp);
            String s;
            if(i%seqnum < 10)
            {
                s = new String("0" + i%seqnum + ":" + temp);
            }
            else
            {
                s = new String(i%seqnum + ":" + temp);
            }

            byte[] data = s.getBytes();
            DatagramPacket SenderPacket = new DatagramPacket(data,
data.length, InetAddress.getLocalHost(),SendDataPort);
            //接收端地址也是本地
            SenderSocket.send(SenderPacket);
            System.out.println("重发分组:" + i%seqnum+";重发的是第
"+i+"个包");

            TimeBegin();
        }

    } catch (Exception e) {
    }
}

};

if (!flag) {
    flag = true;
} else {
    executor.shutdown();
}
executor=Executors.newSingleThreadScheduledExecutor();
executor.scheduleWithFixedDelay(task, timeout, timeout,
TimeUnit.SECONDS);

}

/**
 * 结束计时
 */

```

```
public void TimeEnd()
{
    if (flag)
    {
        executor.shutdown();
        flag = false;
    }
}

/**
 * 重置计时器
 */
public void TimeReset()
{
    if (send_base == nextseqnum)
    {
        TimeEnd();
    }
    else
    {
        TimeBegin();
    }
}

/**
 * 读取文件，存入 filestr 和字节数组 B 中
 *
 * @param filename 文件名
 */
public static void ReadFilebyLines(String filename)
{
    filename = "src\\lab2\\"+filename;
    File file = new File(filename);
    BufferedReader reader = null;
    try
    {
        reader = new BufferedReader(new FileReader(file));
        String tempstr = null;
        while ((tempstr = reader.readLine()) != null)
        {
            filestr += tempstr + "\r\n";
        }
        reader.close();
        //将文件转为比特存储于 B 中
    }
}
```



```
        B = filestr.getBytes();

    } catch (IOException e)
    {
        e.printStackTrace();
    } finally
    {
        if (reader != null)
        {
            try
            {
                reader.close();
            } catch (IOException e1)
            {
            }
        }
    }
}

/**
 * 根据 nextseqnum 获得要传输的字节切片(分块传输)
 *
 * @param nextseqnum 要传输的分组序号
 *
 * @return 字节数组, 要传输的字节
 */
public byte[] GetPart(int nextseqnum)
{
    //切割得到数据段
    byte[] temp = new byte[1469];
    for (int i = 0; i < 1469; i++)
    {
        if (nextseqnum * 1469 + i >= B.length)
        {
            break;
        }
        temp[i] = B[nextseqnum * 1469 + i];
    }
    return temp;
}

/**
 * 发送数据
 */
```

```
public void Send()
{
    ReadFilebyLines("data.txt");
    team=(int) Math.ceil(B.length/1469);
    try
    {
        SenderSocket = new DatagramSocket();
        ReceiveAckSocket = new DatagramSocket(ReceiveAckPort);
        while (true)
        {
            SendtoReciver();
            ReceiveACK();
            if (send_base >= team)
            {
                break;
            }
        }
        //数据传输完毕后不要忘了关闭计时器
        executor.shutdown();
        System.out.println("发送结束.");
    } catch (Exception e) {
    }
}

/**
 * 发送数据给接收方
 */
public void SendtoReciver()
{
    try
    {
        //以窗口长度为判定界限。
        while (nextseqnum < send_base + N)
        {
            if (send_base >= team || nextseqnum >= team)
            {
                break;
            }
            byte[] tempb = GetPart(nextseqnum);
            String temp = new String(tempb);
            //给数据段添加上标记序列号的首部
            String s;
            if(nextseqnum%seqnum < 10)
            {
```

```

        s = new String("0" + nextseqnum%seqnum + ":" + temp);
    }
    else
    {
        s = new String(nextseqnum%seqnum + ":" + temp);
    }
    byte[] data = s.getBytes();
    SendDataPacket = new DatagramPacket(data, data.length,
InetAddress.getLocalHost(), SendDataPort);
    // 模拟数据包丢失
    if (nextseqnum % 5 != 0)
    {
        SenderSocket.send(SendDataPacket);
        System.out.println("发送分组:" + nextseqnum%seqnum+";发送
的是第"+nextseqnum+"个包");
    }
    else
    {
        System.out.println("模拟分组" + nextseqnum%seqnum + "丢失
"+" 丢失的是第"+nextseqnum+"个包");
    }
    if (send_base == nextseqnum) //重启计时器
    {
        TimeBegin();
    }
    nextseqnum++;

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

/**
 * 接收 ACK
 * @throws InterruptedException
 */
public void ReceiveACK() throws InterruptedException
{
    try {
        //发送完数据的情况
        if (send_base>=team)
        {
            return;

```

```

    }
    //ACK 消息的大小为 10bytes.
    byte[] bytes = new byte[10];
    ReceiverAckPacket = new DatagramPacket(bytes, bytes.length);
    ReceiveAckSocket.receive(ReceiverAckPacket);
    String ackString = new String(bytes, 0, bytes.length);
    String acknum = new String();
    for (int i = 0; i < ackString.length(); i++)
    {
        //取出 ACK 信息中的 ACK 序列号
        if (ackString.charAt(i) >= '0' && ackString.charAt(i) <= '9')
        {
            acknum += ackString.charAt(i);
        }
        else
        {
            break;
        }
    }
    int ack = Integer.parseInt(acknum);
    // 模拟 ACK 丢包
    if (ack % 6 != 0)
    {
        System.out.println("接收到 ACK" + ack);
        int m;
        //越序号列的情况.
        if
        (((send_base%seqnum)>ack)&&((nextseqnum/seqnum)>(send_base/seqnum))&&(ack<=
        ((send_base+N)%N)))
        {
            m=send_base/seqnum*seqnum+ack+seqnum+1;
        }
        else
        {
            m=send_base/seqnum*seqnum+ack+1;
        }
        send_base = Math.max(send_base, m);
    }
    else
    {
        System.out.println("模拟 ACK" + ack + "丢失");
    }
    TimeReset();
} catch (IOException e) {

```

```
    }  
}  
  
}  
  
public class GBNR implements Runnable{  
    private static int N = 8;  
  
    public GBNR() {  
        super();  
    }  
    @Override  
    public void run() {  
  
        Receive();  
  
    }  
  
    public static void main(String[] args) {  
        new Thread(new GBNR()).start();  
    }  
  
    // 发送数据部分  
    private static int seqnum=16;  
    // 接收数据部分  
    private static int SendAckPort = 10240;  
    private static int ReceiveDataPort = 10241;  
    private static DatagramSocket ReciverSocket;  
    private static DatagramPacket SendAckPacket;  
    private static int expectedSeqNum = 0;  
  
    private static int last=-1;  
    /**  
    * 接收数据并发回 ACK  
    */  
    public void Receive() {  
        try {  
            ReciverSocket = new DatagramSocket(ReceiveDataPort);  
            while (true)  
            {  
                byte[] data= new byte[1472];  

```

```
DatagramPacket packet = new DatagramPacket(data, data.length);
ReceiverSocket.receive(packet);
byte[] d = packet.getData();
String message = new String(d);
String num = new String();
//取出该包的序列号
for (int i = 0; i < message.length(); i++)
{
    if (message.charAt(i) <= '9' && message.charAt(i) >= '0')
    {
        num = num + message.charAt(i);
    } else {
        break;
    }
}
// 进行累积确认, 不是想要的序号段的话直接丢弃
if (expectedSeqNum == Integer.valueOf(num))
{
    int ack = expectedSeqNum;
    SendACK(ack);
    expectedSeqNum = (expectedSeqNum + 1)%seqnum;
    last=ack;
}
else
{
    if (last>=0)
    {
        SendACK(last);
    }
}
} catch (Exception e) {
}
}

/**
 * 发回 ACK
 *
 * @param ack 发回的 ACK 序号, 为 0 到 N-1
 */
public void SendACK(int ack) {
    try
    {
```

```
    ACK ACK = new ACK(ack);
    SendAckPacket = new DatagramPacket(ACK.ackByte, ACK.ackByte.length,
    InetAddress.getLocalHost(), SendAckPort);
    ReciverSocket.send(SendAckPacket);
    System.out.println("发回 ACK" + ack);
} catch (Exception e)
{
}
}
}
```

## 停-等协议：

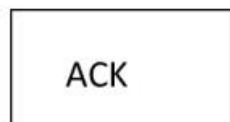
### 停-等协议数据分组格式：



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等于 1472 字节。

本实验中将数据帧的大小恰好设为 1472 字节，其中 Seq 段对应着序列号，大小为 3 个 Byte，Data 段大小恒为 1469Byte，故所以 UDP 数据报的数据部分应等于 1472 字节。

### 确认分组格式：



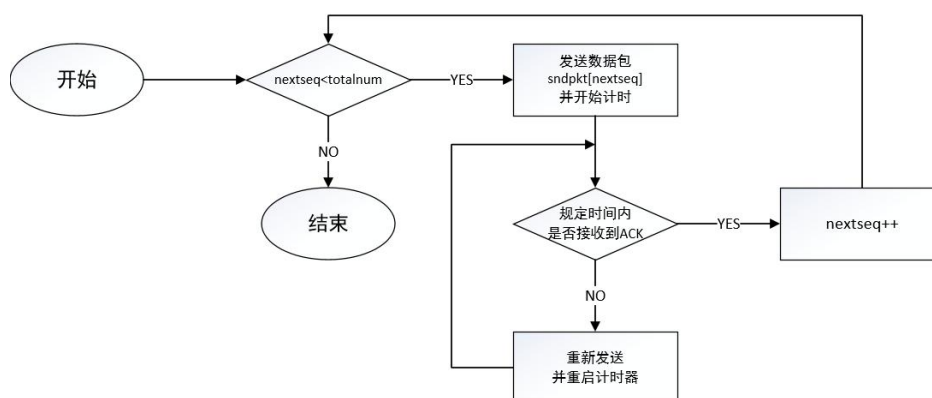
由于是从服务器端到客户端的单向数据传输，因此 ACK 数据帧不包含任何数据，只需要将 ACK 发送给服务器端即可。

ACK 字段用于表示序列号，大小设定为 10 个字节。

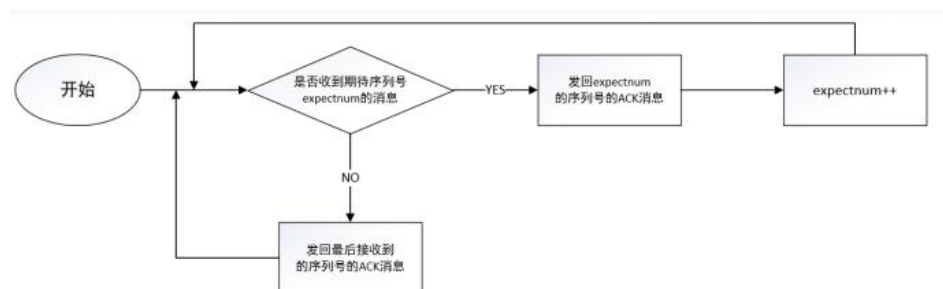
```
public class ACK {  
    protected int ACK;  
    protected String ack;  
    protected byte[] ackByte;  
  
    public ACK(int ACK) {  
        this.ACK=ACK;  
        ack=String.valueOf(ACK);  
        ackByte=ack.getBytes();  
    }  
}
```

### 协议两端程序流程图：

#### 发送端：



#### 接收端：



协议典型交互过程：（在 GBN 协议代码上略加改进得到的停-等协议代码）

send\_base 是已发送未确认 ACK 的分组



nextseqnum 是下一个要发送的数据包

expectseq 是期望收到的数据包序号

**发送端：初始化 send\_base=1,nextseqnum=1**

1. 发送方发送数据过程

(1) 如果有条件  $\text{nextseqnum} < \text{send\_base} + 1$  成立，则发送数据包  $\text{Data}[\text{nextseqnum}]$ ，并使  $\text{seqnum}++$ 。

(2) 重启计时器。

2. 发送方接收到 ACK

(1) 得知获得的 ACK 的序列号  $\text{ack}$ 。

(2) 如果  $\text{send\_base}$  与  $\text{ack}$  相等，关闭计时器。否则重新发送并重启计时器。

3. 发送方的计时器超时

(1) 重发分组  $\text{Data}[\text{send\_base}]$

(2) 重启计时器

**接收端：初始化 expectseq=1**

接收方接收到数据包：

提取数据包的序列号，判断是否等于  $\text{expectseq}$ 。如果相等，则发送回该序列号的 ACK 报文；如果不等，则发送上一次受到的序列号的 ACK 报文。

**数据分组丢失验证模拟方法：**

在发送数据包时，将数据包的序列号取模，如序号为  $x$ ，如果  $x\%5==0$ ，则不发送该数据包。

```
if (nextseqnum % 5 != 0)
{
    SenderSocket.send(SendDataPacket);
    System.out.println("发送分组:" + nextseqnum%seqnum+";发送的是第"+nextseqnum+"个包");
}
else
{
    System.out.println("模拟分组" + nextseqnum%seqnum + "丢失"+"丢失的是第"+nextseqnum+"个包");
}
```

在接收 ACK 处，将 ACK 的序列号也取模，如序号为 xa，如果  $xa \% 6 == 0$ ，则不接收该 ACK。（较 GBN 有所改进，仅丢失一次 ACK 信息，否则停等协议无法继续）

```

if (ack % 6 == 0 && times == 1)
{
    System.out.println("模拟ACK" + ack + "丢失");
    times++;
}
else
{
    System.out.println("接收到"
        + "ACK" + ack);
    int m;
    //越序号列的情况.
    if (((send_base%seqnum)>ack)&&((nextseqnum/seqnum)>(send_base/seqnum))&&(ack<=((send_base+N)%N)))
    {
        m=send_base/seqnum*seqnum+ack+seqnum+1;
    }
    else
    {
        m=send_base/seqnum*seqnum+ack+1;
    }
    send_base = Math.max(send_base, m);
}

```

**程序实现的主要函数及其主要作用（仅提供与 GBN 代码不同的部分）：**

**TimeBegin() 函数：**每次超时后只发送一个数据报文。

```

String s;
if(send_base%seqnum < 10)
{
    s = new String("0" + send_base%seqnum + ":" + temp);
}
else
{
    s = new String(send_base%seqnum + ":" + temp);
}
byte[] data = s.getBytes();
DatagramPacket SenderPacket = new DatagramPacket(data, data.length, InetAddress.getLocalHost(),SendDataPort);
//接收端地址也是本地
SenderSocket.send(SenderPacket);

```

**Receive() 函数：**模拟丢失 ACK 信息的方式有所改变：

```

if (ack % 6 == 0 && times == 1)
{
    System.out.println("模拟ACK" + ack + "丢失");
    times++;
}
else
{
    System.out.println("接收到"
        + "ACK" + ack);
    int m;
    //越序号列的情况.
    if (((send_base%seqnum)>ack)&&((nextseqnum/seqnum)>(send_base/seqnum))&&(ack<=((send_base+N)%N)))
    {
        m=send_base/seqnum*seqnum+ack+seqnum+1;
    }
    else
    {
        m=send_base/seqnum*seqnum+ack+1;
    }
    send_base = Math.max(send_base, m);
}

```

## 实验验证结果：



### 单独发送：

模拟分组0丢失 丢失的是第0个包  
重发分组:0 重发的是第0个包  
模拟ACK0丢失  
重发分组:0 重发的是第0个包  
接收到ACK0  
发送分组:1 发送的是第1个包  
接收到ACK1  
发送分组:2 发送的是第2个包  
接收到ACK2  
发送分组:3 发送的是第3个包  
接收到ACK3  
发送分组:4 发送的是第4个包  
接收到ACK4  
模拟分组5丢失 丢失的是第5个包  
重发分组:5 重发的是第5个包  
接收到ACK5  
发送分组:6 发送的是第6个包  
接收到ACK6  
发送分组:7 发送的是第7个包  
接收到ACK7  
发送分组:8 发送的是第8个包  
接收到ACK8  
发送分组:9 发送的是第9个包  
接收到ACK9  
模拟分组10丢失 丢失的是第10个包  
重发分组:10 重发的是第10个包  
接收到ACK10  
发送分组:11 发送的是第11个包  
接收到ACK11  
发送分组:12 发送的是第12个包  
接收到ACK12  
发送分组:13 发送的是第13个包  
接收到ACK13  
发送分组:14 发送的是第14个包  
接收到ACK14  
模拟分组15丢失 丢失的是第15个包  
重发分组:15 重发的是第15个包  
接收到ACK15  
发送分组:0 发送的是第16个包  
接收到ACK0  
发送分组:1 发送的是第17个包  
接收到ACK1

### 双工：

模拟分组0丢失 丢失的是第0个包  
 发回ACK0  
 重发分组:0 重发的是第0个包  
 模拟ACK0丢失  
 发回ACK0  
 发回ACK1  
 发回ACK2  
 发回ACK3  
 发回ACK4  
 重发分组:0 重发的是第0个包  
 接收到ACK0  
 发送分组:1 发送的是第1个包  
 接收到ACK1  
 发送分组:2 发送的是第2个包  
 接收到ACK2  
 发送分组:3 发送的是第3个包  
 接收到ACK3  
 发送分组:4 发送的是第4个包  
 接收到ACK4  
 模拟分组5丢失 丢失的是第5个包  
 发回ACK5  
 发回ACK6  
 发回ACK7  
 发回ACK8  
 发回ACK9

文件传输:

 data.txt  
 receivedata.txt

详细注释源程序:

```

public class StopWaitS implements Runnable{

    private static int seqnum=16;

    private static int N = 1;

    public StopWaitS() {
        super();
    }

    @Override
    public void run() {

        Send();

    }

    public static void main(String[] args) {
        new Thread(new StopWaitS()).start();
    }
  
```

```

}

// 发送数据部分
private static int SendDataPort = 10241; //接收方端口号
private static int ReceiveAckPort = 10240; //发送方端口号
private static DatagramSocket SenderSocket;
private static DatagramPacket SendDataPacket;
private static DatagramSocket ReceiveAckSocket;
private static DatagramPacket ReceiverAckPacket;
private static int send_base = 0;
private static int nextseqnum = 0;
private static boolean flag = false; //是否开始计时的标志
private static int timeout = 2; //超时时间定为 5s
private static String filestr = new String();
private static byte[] B;
private static int team;
private static int times = 1;
private ScheduledExecutorService executor;

/**
 * 开始计时或者重新计时，超时时间为 5s
 */
public void TimeBegin()
{
    TimerTask task = new TimerTask()
    {
        @Override
        public void run()
        {
            //UDP 数据包每次能够传输的最大长度 = MTU(1500B) - IP 头(20B) - UDP
            //头(8B) = 1472Bytes
            //没有信息发送时就返回。
            if (send_base >= Math.ceil(B.length / 1469) - 1) //取大于等于的最
            近整数
            {
                executor.shutdown();
                return;
            }
            try
            {
                //重新开始计时并发送
                byte[] tempb = GetPart(send_base);
                String temp = new String(tempb);

```

```

//          String s = new String(send_base%seqnum + ":" + temp);
          String s;
          if(send_base%seqnum < 10)
          {
              s = new String("0" + send_base%seqnum + ":" + temp);
          }
          else
          {
              s = new String(send_base%seqnum + ":" + temp);
          }

          byte[] data = s.getBytes();
          DatagramPacket SenderPacket = new DatagramPacket(data,
data.length, InetAddress.getLocalHost(),SendDataPort);
          //接收端地址也是本地
          SenderSocket.send(SenderPacket);
          System.out.println("重发分组:" + send_base%seqnum+" 重
发的是第"+send_base+"个包");
          TimeBegin();
          } catch (Exception e) {
          }
      }
  };

  if (!flag) {
      flag = true;
  } else {
      executor.shutdown();
  }
  executor=Executors.newSingleThreadScheduledExecutor();
  executor.scheduleWithFixedDelay(task, timeout, timeout,
TimeUnit.SECONDS);

}

/**
 * 结束计时
 */
public void TimeEnd()
{
    if (flag)
    {
        executor.shutdown();
        flag = false;
    }
}

```

```
}

/**
 * 重置计时器
 */
public void TimeReset()
{
    if (send_base == nextseqnum)
    {
        TimeEnd();
    }
    else
    {
        TimeBegin();
    }
}

/**
 * 读取文件，存入 filestr 和字节数组 B 中
 *
 * @param filename
 */
public static void ReadFilebyLines(String filename)
{
    filename = "src\\lab2\\"+filename;
    File file = new File(filename);
    BufferedReader reader = null;
    try
    {
        reader = new BufferedReader(new FileReader(file));
        String tempstr = null;
        while ((tempstr = reader.readLine()) != null)
        {
            filestr += tempstr + "\r\n";
        }
        reader.close();
        B = filestr.getBytes();
    } catch (IOException e)
    {
        e.printStackTrace();
    } finally
    {
        if (reader != null)
    }
```

```
        {
            try
            {
                reader.close();
            } catch (IOException e1)
            {
            }
        }
    }
}

/**
 * 根据 nextseqnum 获得要传输的字节切片(分块传输)
 *
 * @param nextseqnum
 *         要传输的分组序号
 * @return 字节数组, 要传输的字节
 */
public byte[] GetPart(int nextseqnum)
{
    byte[] temp = new byte[1469];
    for (int i = 0; i < 1469; i++)
    {
        if (nextseqnum * 1469 + i >= B.length)
        {
            break;
        }
        temp[i] = B[nextseqnum * 1469 + i];
    }
    return temp;
}

/**
 * 发送数据
 */
public void Send()
{
    ReadFilebyLines("data.txt");
    team=(int) Math.ceil(B.length/1469);
    try
    {
        SenderSocket = new DatagramSocket();
        ReceiveAckSocket = new DatagramSocket(ReceiveAckPort);
        while (true)
```



```

        {
            SendtoReciver();
            ReceiveACK();
            if (send_base >= team)
            {
                break;
            }
        }
        executor.shutdown();
        System.out.println("发送结束.");
    } catch (Exception e) {
    }
}

/**
 * 发送数据给接收方
 */
public void SendtoReciver()
{
    try
    {
        //以窗口长度为判定界限.
        while (nextseqnum < send_base + N)
        {
            if (send_base >= team || nextseqnum >= team)
            {
                break;
            }
            byte[] tempb = GetPart(nextseqnum);
            String temp = new String(tempb);
            //给数据段添加上标记序号的首部
            String s = new String(nextseqnum%seqnum + ":" + temp);
            String s;
            if(nextseqnum%seqnum < 10)
            {
                s = new String("0" + nextseqnum%seqnum + ":" + temp);
            }
            else
            {
                s = new String(nextseqnum%seqnum + ":" + temp);
            }

            byte[] data = s.getBytes();
            SendDataPacket = new DatagramPacket(data, data.length,
InetAddress.getLocalHost(), SendDataPort);

```

```

        // 模拟数据包丢失
        if (nextseqnum % 5 != 0)
        {
            SenderSocket.send(SendDataPacket);
            System.out.println("发送分组:" + nextseqnum%seqnum+" 发送
的是第"+nextseqnum+"个包");
        }
        else
        {
            System.out.println("模拟分组" + nextseqnum%seqnum + "丢失
"+" 丢失的是第"+nextseqnum+"个包");
        }
        if (send_base == nextseqnum)
        {
            TimeBegin();
        }
        nextseqnum++;
    }
} catch (Exception e) {
    e.printStackTrace();
}
}

/**
 * 接收 ACK
 * @throws InterruptedException
 */
public void ReceiveACK() throws InterruptedException
{
    try {
        //发送完数据的情况
        if (send_base>=team)
        {
            return;
        }
        //ACK 消息的大小为 10bytes.
        byte[] bytes = new byte[10];
        ReceiverAckPacket = new DatagramPacket(bytes, bytes.length);
        ReceiveAckSocket.receive(ReceiverAckPacket);
        String ackString = new String(bytes, 0, bytes.length);
        String acknum = new String();
        for (int i = 0; i < ackString.length(); i++)
        {

```

```

        //取出 ACK 信息中的完成 ACK 序列号
        if (ackString.charAt(i) >= '0' && ackString.charAt(i) <= '9')
        {
            acknum += ackString.charAt(i);
        }
        else
        {
            break;
        }
    }
    int ack = Integer.parseInt(acknum);
    if (ack % 6 == 0 && times == 1)
    {
        System.out.println("模拟 ACK" + ack + "丢失");
        times++;
    }
    else
    {
        System.out.println("接收到"
            + "ACK" + ack);
        int m;
        //越序号列的情况.
        if
        (((send_base%seqnum)>ack)&&((nextseqnum/seqnum)>(send_base/seqnum))&&(ack<=
        ((send_base+N)%N)))
        {
            m=send_base/seqnum*seqnum+ack+seqnum+1;
        }
        else
        {
            m=send_base/seqnum*seqnum+ack+1;
        }
        send_base = Math.max(send_base, m);
    }

    TimeReset();
} catch (IOException e) {
}
}

}

public class StopWaitR implements Runnable{
    private static int N = 1;

```

```
public StopWaitR() {
    super();
}

@Override
public void run() {

    Receive();

}

public static void main(String[] args) {
    new Thread(new StopWaitR()).start();
}

// 接收数据部分
private static int SendAckPort = 10240;
private static int ReceiveDataPort = 10241;
private static DatagramSocket ReciverSocket;
private static DatagramPacket SendAckPacket;
private static int expectedSeqNum = 0;
private static int seqnum=16;
private static int last=-1;

/**
 * 接收数据并发回 ACK
 */
public void Receive() {
    try {
        ReciverSocket = new DatagramSocket(ReceiveDataPort);
        while (true)
        {
            byte[] data= new byte[1472];
            //      if(expectedSeqNum <= 9)
            //          data = new byte[1471];
            //      else
            //          data = new byte[1472];
            DatagramPacket packet = new DatagramPacket(data, data.length);
            ReciverSocket.receive(packet);
            byte[] d = packet.getData();
            String message = new String(d);
```

```

String num = new String();
for (int i = 0; i < message.length(); i++)
{
    if (message.charAt(i) <= '9' && message.charAt(i) >= '0')
    {
        num = num + message.charAt(i);
    } else {
        break;
    }
}
// 进行累积确认，不是想要的序号段的话直接丢弃
if (expectedSeqNum == Integer.valueOf(num))
{
    int ack = expectedSeqNum;
    SendACK(ack);
    expectedSeqNum = (expectedSeqNum + 1)%seqnum;
    Last=ack;
}
else
{
    if (Last>=0)
    {
        SendACK(Last);
    }
}
} catch (Exception e) {
}
}

/**
 * 发回 ACK
 *
 * @param ack
 *      发回的 ACK 序号，为 0 到 N-1
 */
public void SendACK(int ack) {
    try {
        ACK ACK = new ACK(ack);
        SendAckPacket = new DatagramPacket(ACK.ackByte, ACK.ackByte.length,
        InetAddress.getLocalHost(), SendAckPort);
        ReceiverSocket.send(SendAckPacket);
        System.out.println("发回 ACK" + ack);
    } catch (Exception e) {

```

```
}  
}  
  
}
```

## SR 协议：

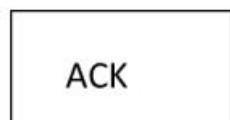
### SR 协议数据分组格式：



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等于 1472 字节。

本实验中将数据帧的大小恰好设为 1472 字节，其中 Seq 段对应着序列号，大小为 3 个 Byte，Data 段大小恒为 1469Byte，故所以 UDP 数据报的数据部分应等于 1472 字节。

### 确认分组格式：



由于是从服务器端到客户端的单向数据传输，因此 ACK 数据帧不包含任何数据，只需要将 ACK 发送给服务器端即可。

ACK 字段用于表示序列号，大小设定为 10 个字节。

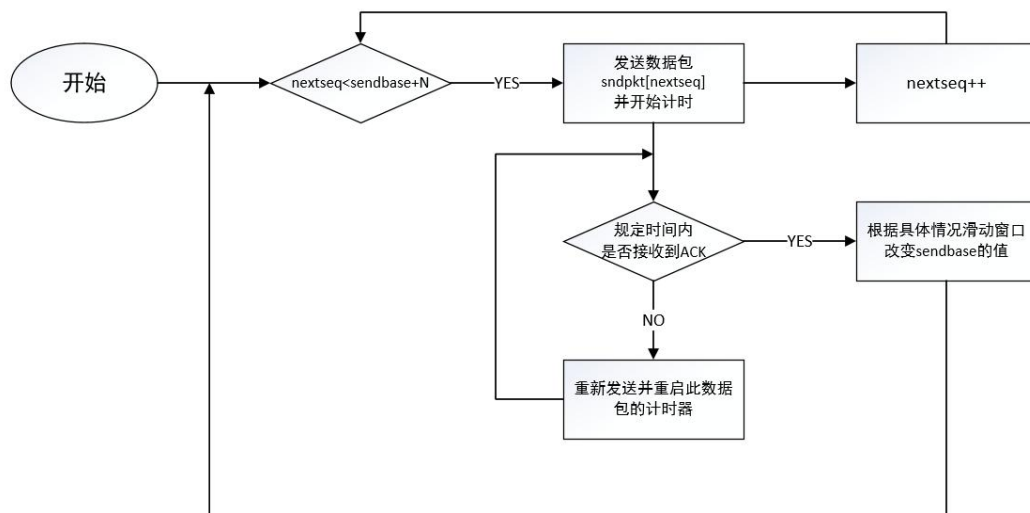
```

public class ACK {
    protected int ACK;
    protected String ack;
    protected byte[] ackByte;

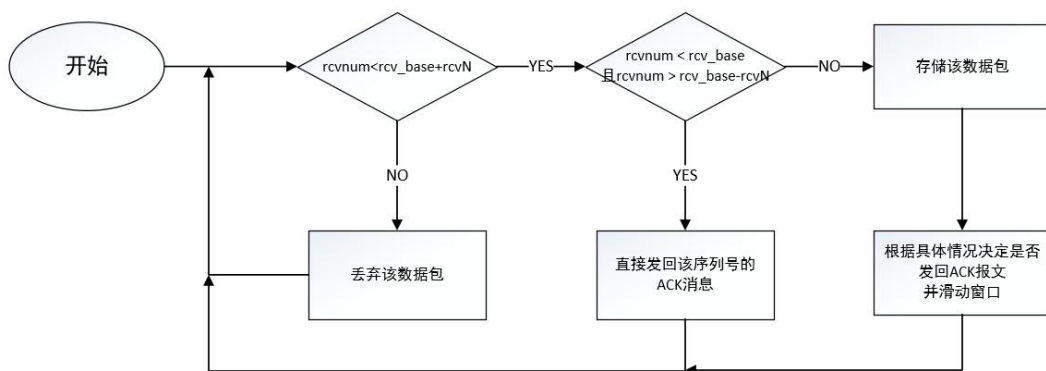
    public ACK(int ACK) {
        this.ACK=ACK;
        ack=String.valueOf(ACK);
        ackByte=ack.getBytes();
    }
}
    
```

### 协议两端程序流程图：

#### 发送端：



#### 接收端：



### 协议典型交互过程:

设发送方窗口大小为  $N$

设接收方窗口大小为  $revN$

$send\_base$  是已发送未确认 ACK 的分组

$nextseq$  是下一个要发送的数据包

$rev\_base$  代表接收窗口的第一个分组

发送端: 初始化  $send\_base=1, nextseqnum=1$

#### 1. 发送方发送数据过程

(1) 如果有条件  $nextseqnum < base+N$ , 则发送数据包  $Data[nextseqnum]$ , 并使  $seqnum++$

(2) 对  $Data[nextseqnum]$  启动一个计时器

#### 2. 发送方接收到 ACK

(1) 得知获得的 ACK 的序号, 如获得  $ACK\_ack$

(2) 结束  $send\_base$  到  $ack$  间所有的计时器

(3)  $send\_base=ack+1$

#### 3. 发送方的计时器 $k$ 超时

(1) 重发分组  $Data[x]$

(2) 重启计时器  $k$

接收端: 初始化  $expectseq=1$

接收方接收到数据包:

提取数据包的  $seq$ 。如果  $rev\_base \leq seq < rev\_base+N$ , 发送 ACK 信息报文; 如果  $seq\_base$  等于  $seq$ , 则滑动窗口; 如果  $rev\_base-N \leq seq < rev\_base$ , 发送 ACK 信息报文; 都不满足, 则不做任何处理

### 数据分组丢失验证模拟方法:

在发送数据包时, 将数据包的序列号取模, 如序号为  $x$ , 如果  $x \% 5 = 0$ , 则不发送该数据包。



```

if (nextseqnum % 5 != 0)
{
    SenderSocket.send(SendDataPacket);
    System.out.println("发送分组:" + nextseqnum%seqnum+";发送的是第"+nextseqnum+"个包");
}
else
{
    System.out.println("模拟分组" + nextseqnum%seqnum + "丢失"+" 丢失的是第"+nextseqnum+"个包");
}

```

在接收 ACK 处，将 ACK 的序列号也取模，如序号为 xa，如果  $xa \% 6 == 0$ ，则不接收该 ACK。（较 GBN 有所改进，仅丢失一次 ACK 信息，否则 SR 协议无法继续）

```

if (ack % 6 == 0 && times == 1)
{
    System.out.println("模拟ACK" + ack + "丢失");
    times++;
}
else
{
    System.out.println("接收到ACK" + ack%seqnum);
    int a = ack, b = ack + seqnum;
    //找到ACK信息代表的包号，通过包号建立起两个窗口序号的关系
    while (!(send_base >= a && send_base <= b))
    {
        a += seqnum;
        b += seqnum;
    }
    //判断是否接收到重复ACK（send_base之前的ACK不进行处理）
    if (b - send_base > send_base - a)
    {
        ack = a;
    }
    else
    {
        ack = b;
    }
}
// System.out.println("ack=" + ack);

```

程序实现的主要函数及其主要作用（仅提供与 GBN 代码不同的部分）：

TimeBegin(int q, int x) 函数：多个线程为每个在发送方窗口内已发送的序列号计时

```

public void TimeBegin(int q, int x) {
    TimerTask task = new TimerTask() {
        @Override
        public void run() {
            if (send_base >= Math.ceil(B.length / 1469)-1) {
                return;
            }
            try {
                byte[] tempb = GetPart(x);
                String temp = new String(tempb);
                String s;
                if(nextseqnum%seqnum < 10)
                {
                    s = new String("0" + x%seqnum + ":" + temp);
                }
                else
                {
                    s = new String(x%seqnum + ":" + temp);
                }
                byte[] data = s.getBytes();
                DatagramPacket SenderPacket = new DatagramPacket(data, data.length, InetAddress.getLocalHost(), SendDataPort);
                SenderSocket.send(SenderPacket);
                System.out.println("重发分组:" + x % seqnum + ";重发的是第" + x + "个包");
            } catch (Exception e) {
            }
        }
    };
    if (!flags[q]) {
        flags[q] = true;
    }
}

```

TimeEnd(int q, int x): 指定某一个计时器暂停。

```
public void TimeEnd(int q, int x) {
    if (flags[q]) {
        flags[q] = false;
        executors[q].shutdown();//终止计时器
    }
}
```

ReceiveACK() 函数: 添加了暂存 ACK 信息与窗口滑动的功能

```
System.out.println("接收到ACK" + ack%seqnum);
int a = ack, b = ack + seqnum;
//找到ACK信息代表的包号, 通过包号建立起两个窗口序号的关系
while (!(send_base >= a && send_base <= b))
{
    a += seqnum;
    b += seqnum;
}
//判断是否接收到重复ACK (send_base之前的ACK不进行处理)
if (b - send_base > send_base - a)
{
    ack = a;
}
else
{
    ack = b;
}
System.out.println("ack=" + ack);
if (ack >= send_base && ack < send_base + N)
{
    ackarray[ack - send_base] = true;//表示收到ack。
    TimeEnd(ack - send_base, ack);
    if (ack == send_base) //窗口滑动
    {
        ackarray[0] = true;
        int cnt = 0;
        for (int i = 0; i < N; i++)
        {
            if (ackarray[i])
            {
                cnt++;
            }
            else
            {
                break;
            }
        }
        System.out.println("cnt=" + cnt);
        for (int i = 0; i < N - cnt; i++)
        {
            ackarray[i] = ackarray[i + cnt];
            flags[i] = flags[i + cnt];
            executors[i] = executors[i + cnt];
        }
        for (int i = N - cnt; i < N; i++)
        {
            ackarray[i] = false;
            executors[i] = null;
            flags[i] = false;
        }
        send_base = send_base + cnt;
    }
}
```

Receive() 函数: 添加了暂存 ACK 信息与窗口滑动的功能

```
int a=rev_num,b=rev_num+seqnum;
while (!(rev_base>=a&&rev_base<=b))
{
    a+=seqnum;
    b+=seqnum;
}
if (b-rev_base>rev_base-a)
{
    rev_num=a;
}
else
{
    rev_num=b;
}
if (rev_num>=rev_base&&rev_num<rev_base+RecN)
{
    SendACK(rev_num);
    if (rev_num==rev_base) //窗口滑动
    {
        ack[0]=true;
        int cnt=0;
        for (int i=0;i<RecN;i++)
        {
            if (ack[i])
            {
                cnt++;
            }
            else
            {
                break;
            }
        }
        for (int i=0;i<RecN-cnt;i++)
        {
            ack[i]=ack[i+cnt];
        }
        for (int i=RecN-cnt;i<RecN;i++)
        {
            ack[i]=false;
        }
        rev_base=rev_base+cnt;
    }
    else
    {
        ack[rev_num-rev_base]=true;//记录已收到文件
    }
}
else if (rev_num<rev_base&&rev_num>=rev_base-RecN) //在此范围内一律发送ACK
{
    SendACK(rev_num);
}
```

实验验证结果:

发送方超时重发:

模拟分组0丢失;丢失的是第0个包  
发送分组:1;发送的是第1个包  
发送分组:2;发送的是第2个包  
发送分组:3;发送的是第3个包  
发送分组:4;发送的是第4个包  
模拟分组5丢失;丢失的是第5个包  
发送分组:6;发送的是第6个包  
发送分组:7;发送的是第7个包  
重发分组:7;重发的是第7个包  
重发分组:4;重发的是第4个包  
重发分组:6;重发的是第6个包  
重发分组:5;重发的是第5个包  
重发分组:1;重发的是第1个包  
重发分组:2;重发的是第2个包  
重发分组:0;重发的是第0个包  
重发分组:3;重发的是第3个包

#### 发送方窗口滑动:

接收到ACK4  
接收到ACK1  
接收到ACK2  
接收到ACK3  
接收到ACK0---先前ACK, 不处理  
窗口向后滑动5

#### 接收方窗口滑动:

发回ACK4  
发回ACK1  
发回ACK2  
发回ACK3  
发回ACK0  
窗口向后滑动5

#### 详细注释源程序:

```
public class SRS implements Runnable{

    private static int seqnum = 16;

    public SRS() {
        super();
    }

    @Override
```

```

public void run()
{
    Send();
}

public static void main(String[] args) {
    new Thread(new SRS()).start();
    // new Thread(new SRS(1)).start();
}

// 发送数据部分
private static int N = 8;
private static int SendDataPort = 10241;
private static int ReceiveAckPort = 10240;
private static DatagramSocket SenderSocket;
private static DatagramPacket SendDataPacket;
private static DatagramSocket ReceiveAckSocket;
private static DatagramPacket ReceiverAckPacket;
private static int send_base = 0;
private static int nextseqnum = 0;
private static int timeout = 4;
    private static int times = 1;
private static String filestr = new String();
private static byte[] B;
private static int team;
private static boolean[] ackarray = new boolean[N]; //判断每个序列号是否被
接收方接收到了的标志数组
private static boolean[] flags = new boolean[N]; //判断每个序列号的计时器是
否开启的标志数组
private static ScheduledExecutorService[] executors = new
ScheduledExecutorService[N];

/**
 * 开始计时或者重新计时，超时时间为 2s
 */
public void TimeBegin(int q, int x) {
    TimerTask task = new TimerTask() {
        @Override
        public void run() {
            if (send_base >= Math.ceil(B.length / 1469)-1) {
                return;
            }
            try {
                byte[] tempb = GetPart(x);

```

```

        String temp = new String(tempb);
        String s;
        if(nextseqnum%seqnum < 10)
        {
            s = new String("0" + x%seqnum + ":" + temp);
        }
        else
        {
            s = new String(x%seqnum + ":" + temp);
        }

        byte[] data = s.getBytes();
        DatagramPacket SenderPacket = new DatagramPacket(data,
data.length, InetAddress.getLocalHost(),SendDataPort);
        SenderSocket.send(SenderPacket);
        System.out.println("重发分组:" + x % seqnum + ";重发的是第"
+ x + "个包");
    } catch (Exception e) {
    }
}

};
if (!flags[q]) {
    flags[q] = true;
} else {
    executors[q].shutdown();
}
executors[q] = Executors.newSingleThreadScheduledExecutor();
executors[q].scheduleWithFixedDelay(task, timeout, timeout,
TimeUnit.SECONDS);
}

/**
 * 结束计时
 */
public void TimeEnd(int q, int x) {
    if (flags[q]) {
        flags[q] = false;
        executors[q].shutdown();//终止计时器
    }
}

/**
 * 读取文件，存入 filestring 和字节数组 B 中
 *
 * @param filename

```

```
*/  
public static void ReadFilebyLines(String filename) {  
    filename = "src\\lab2\\"+filename;  
    File file = new File(filename);  
    BufferedReader reader = null;  
    try {  
        reader = new BufferedReader(new FileReader(file));  
        String tempstr = null;  
        while ((tempstr = reader.readLine()) != null) {  
            filestr = filestr + tempstr + "\r\n";  
        }  
        reader.close();  
        //将所有数据转化为 byte 存到 B 数组中  
        B = filestr.getBytes();  
    } catch (IOException e) {  
        e.printStackTrace();  
    } finally {  
        if (reader != null) {  
            try {  
                reader.close();  
            } catch (IOException e1) {  
            }  
        }  
    }  
}  
  
/**  
 * 根据 nextseqnum 获得要传输的字节  
 *  
 * @param nextseqnum 要传输的分组序号  
 *  
 * @return 字节数组，要传输的字节  
 */  
public byte[] GetPart(int nextseqnum) {  
    byte[] temp = new byte[1469];  
    for (int i = 0; i < 1469; i++) {  
        if (nextseqnum * 1469 + i >= B.length) {  
            break;  
        }  
        temp[i] = B[nextseqnum * 1469 + i];  
    }  
    return temp;  
}
```

```
/**
 * 发送数据
 */
public void Send() {
    ReadFilebyLines("data.txt");
    team = (int) Math.ceil(B.length / 1469);
    try {
        SenderSocket = new DatagramSocket();
        ReceiveAckSocket = new DatagramSocket(ReceiveAckPort);
        while (true) {
            SendToReciver();
            ReceiveACK();
//            System.out.println("send_base=" + send_base);
            if (send_base >= team) {
                break;
            }
        }
        //传输完毕后不要忘了关定时器
        for(int i = 0 ; i < N ; i++)
        {
            executors[i].shutdown();
        }
        System.out.println("发送结束.");
    } catch (Exception e) {
        System.out.println("发送结束.");
    }
}

/**
 * 发送数据给接收方
 */
public void SendToReciver() {
    try {
        while (nextseqnum < send_base + N) {
            if (send_base >= team || nextseqnum >= team) {
                break;
            }
            byte[] tempb = GetPart(nextseqnum);
            String temp = new String(tempb);
            String s;
            if(nextseqnum%seqnum < 10)
            {
                s = new String("0" + nextseqnum%seqnum + ":" + temp);
            }
        }
    }
}
```



```

        else
        {
            s = new String(nextseqnum%seqnum + ":" + temp);
        }

        byte[] data = s.getBytes();
        SendDataPacket = new DatagramPacket(data, data.length,
InetAddress.getLocalHost(), SendDataPort);
        // 模拟数据包丢失
        if (nextseqnum % 5 != 0) {
            SenderSocket.send(SendDataPacket);
            System.out.println("发送分组:" + nextseqnum % seqnum + ";
发送的是第" + nextseqnum + "个包");
        } else {
            System.out.println("模拟分组" + nextseqnum % seqnum + "丢
失" + ";丢失的是第" + nextseqnum + "个包");
        }
        //      System.out.println("nextseqnum=" + nextseqnum);
        //      System.out.println("send_base=" + send_base);
        TimeBegin(nextseqnum - send_base, nextseqnum);
        nextseqnum++;

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

/**
 * 接收 ACK
 *
 * @throws InterruptedException
 */
public void ReceiveACK() throws InterruptedException {
    try
    {
        if (send_base >= team)
        {
            return;
        }
        byte[] bytes = new byte[10];
        ReceiverAckPacket = new DatagramPacket(bytes, bytes.length);
        ReceiveAckSocket.receive(ReceiverAckPacket);
        String ackString = new String(bytes, 0, bytes.length);
        String acknum = new String();
    }
}

```

```

    for (int i = 0; i < ackString.length(); i++)
    {
        //取出 ACK 信息中的完成 ACK 序列号
        if (ackString.charAt(i) >= '0' && ackString.charAt(i) <= '9')
        {
            acknum += ackString.charAt(i);
        }
        else
        {
            break;
        }
    }
    int ack = Integer.parseInt(acknum);
    if ( times == 1 && ack == 15)
    {
        System.out.println("模拟 ACK" + ack + "丢失");
        times++;
    }
    else
    {
        System.out.print("接收到 ACK" + ack%seqnum);
        int a = ack, b = ack + seqnum;
        //找到 ACK 信息代表的包号，通过包号建立起两个窗口序号的关系
        while (!(send_base >= a && send_base <= b))
        {
            a += seqnum;
            b += seqnum;
        }
        //判断是否接收到重复 ACK (send_base 之前的 ACK 不进行处理)
        if (b - send_base > send_base - a)
        {
            ack = a;
            System.out.println("----先前 ACK，不处理");
        }
        else
        {
            ack = b;
            System.out.println();
        }
    }
    // System.out.println("ack=" + ack);
    if (ack >= send_base && ack < send_base + N)
    {
        ackarray[ack - send_base] = true; //表示收到 ack。
        TimeEnd(ack - send_base, ack);
    }

```

```
        if (ack == send_base) //窗口滑动
        {
            ackarray[0] = true;
            int cnt = 0;
            for (int i = 0; i < N; i++)
            {
                if (ackarray[i])
                {
                    cnt++;
                }
                else
                {
                    break;
                }
            }

            for (int i = 0; i < N - cnt; i++)
            {
                ackarray[i] = ackarray[i + cnt];
                flags[i] = flags[i + cnt];
                executors[i] = executors[i + cnt];
            }
            for (int i = N - cnt; i < N; i++)
            {
                ackarray[i] = false;
                executors[i] = null;
                flags[i] = false;
            }
            System.out.println("窗口向后滑动" + cnt);
            send_base = send_base + cnt;
        }
    }
} catch (IOException e) {
}

}

}
```

```
public class SRR implements Runnable{

    public SRR() {
        super();
    }

    @Override
    public void run() {

        Receive();

    }

    public static void main(String[] args) {
        new Thread(new SRR()).start();
    }

    private static int seqnum=16;

    // 接收数据部分
    private static int SendAckPort = 10240;
    private static int ReceiveDataPort = 10241;
    private static DatagramSocket ReciverSocket;
    private static DatagramPacket SendAckPacket;
    private static int RecN=5;
    private static int N = 8;
    private static int rev_base=0;
    private static boolean ack[]=new boolean[RecN];

    /**
     * 接收数据并发回 ACK
     */
    public void Receive() {
        try {
            ReciverSocket = new DatagramSocket(ReceiveDataPort);
            while (true)
            {
                byte[] data = new byte[1472];
                DatagramPacket packet = new DatagramPacket(data, data.length);
                ReciverSocket.receive(packet);
                byte[] d = packet.getData();
```

```
String message = new String(d);
String num = new String();
for (int i = 0; i < message.length(); i++)
{
    if (message.charAt(i) <= '9' && message.charAt(i) >= '0')
    {
        num = num + message.charAt(i);
    }
    else
    {
        break;
    }
}
int rev_num=Integer.valueOf(num);
//找到包号，通过包号建立起两个窗口序号的关系
int a=rev_num,b=rev_num+seqnum;
while (!(rev_base>=a&&rev_base<=b))
{
    a+=seqnum;
    b+=seqnum;
}
if (b-rev_base>rev_base-a)
{
    rev_num=a;
}
else
{
    rev_num=b;
}
if (rev_num>=rev_base&&rev_num<rev_base+RecN)
{
    SendACK(rev_num);
    if (rev_num==rev_base) //窗口滑动
    {
        ack[0]=true;
        int cnt=0;
        for (int i=0;i<RecN;i++)
        {
            if (ack[i])
            {
                cnt++;
            }
            else
            {

```

```

        break;
    }
}
for (int i=0;i<RecN-cnt;i++)
{
    ack[i]=ack[i+cnt];
}
for (int i=RecN-cnt;i<RecN;i++)
{
    ack[i]=false;
}
rev_base=rev_base+cnt;
System.out.println("窗口向后滑动" + cnt);
}
else
{
    ack[rev_num-rev_base]=true;//记录已收到文件
}
}
else if (rev_num<rev_base&&rev_num>=rev_base-N) //在此范围内一律发送 ACK
{
    SendACK(rev_num);
}
}
} catch (Exception e) {
}
}

/**
 * 发回 ACK
 *
 * @param ack 发回的 ACK 序号，为 0 到 N-1
 *
 */
public void SendACK(int ack) {
    try {
        ACK ACK = new ACK(ack);
        SendAckPacket = new DatagramPacket(ACK.ackByte, ACK.ackByte.length,
InetAddress.getLocalHost(),SendAckPort);
        ReceiverSocket.send(SendAckPacket);
        System.out.println("发回 ACK" + ack%seqnum);
    } catch (Exception e) {
    }
}
}

```

}

## 四、实验心得

纸上得来终觉浅，绝知此事要躬行。本来我自以为通过 MOOC 上的课程已经较好地掌握了 SR 滑动窗口的知识。但真正做起实验来，我才发现在 MOOC 中我只是大概了解了一下 SR 滑动窗口的操作，而对其具体的实现，只是一知半解，不得不通过书本和网络资料查找来了解其具体的操作过程。正是因此这次实验让我加深了对停-等协议、GBN 协议、SR 协议的理解，并且从完成这部分内容中收获了相当大的乐趣。

同时，代码的合理复用也很重要，本次实验中，我就是通过 GBN 协议与 SR 协议、停-等协议间的相似性（演变关系），在 GBN 的代码上加以部分改进实现了这两个协议的内容，大大提高了完成实验的效率。