

**计算机网络**

**课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 可靠数据传输协议 | | | | | |
| 姓名 | 黄友勤 | | 院系 | 计算机学院 | | |
| 班级 | 1603108班 | | 学号 | 1160300621 | | |
| 任课教师 | 聂兰顺 | | 指导教师 | 聂兰顺 | | |
| 实验地点 | 格物207 | | 实验时间 | 2018.11.9 | | |
| 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 | | | | | | |
|  | | | | | | |

****

|  |
| --- |
| 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。 |
| 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；  4.将所设计的 GBN 协议改进为 SR 协议 |
| 实验过程： |
| **一、协议数据分组格式：**   1. **数据分组格式：**      1. **Seq**：数据的头部是一个序列号，代表的是发送的是第几个分组的序列号，序列号根据自己的分组对最大序列号取模取得。   接收方根据Seq来确定受到的分组序号并返回ACK   1. **Data**：定义一般的数据报中的数据部分，一般小于1024字节。接收方接收的数据部分。   (c)EOF0：接收数据的末尾，判断传输的数据是否是结束。  **在程序中，每一个分组都以字节数组来表示。**  **2. ACK分组格式：**  ACK分组的方法设计很简单，接收方将确认的序列号发送给发送方即可，因此ACK分组则是用字节数组，将表示数字的字符串表示即可。  如，要发送ACK10，则发送“10”.getbytes给发送方。  **二、流程图与交互过程**  **1.GBN发送方流程图：**    **2.GBN接收方流程图**    **3.GBN协议交互过程**  **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**  **发送方：设置send\_base=1,nextseqnum=1**   1. **发送方发送数据过程**   (a).如果nextseqnum<base+N,则发送数据包Data[nextseqnum]，并使seqnum++  (b).重启计时器  **2.发送方接收到ACK**  (a).得知获得的ACK的序号，如获得ACK\_ack  (b).send\_base=ack+1  (c).如果send\_base与nextseqnum相等，停止计时器。否则，重启计时器。  **3.发送方的计时器超时**  (a).重发分组Data[send\_base]到Data[nextseqnum-1]  (b).重启计时器  **接收方：设置expectseq=1（期望收到的数据包序号）**   1. **接收方接收到数据包**   (a).提取数据包的seq，判断是否等于expectseq,  ①如果相等，则发送回ACK\_seq  ②如果不等，则发送上一次受到的数据包的ACK(ACK\_last0)  **1.SR发送方**    **2.SR接收方**    **3.SR协议交互过程**  **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**  **发送方：设置send\_base=1,nextseqnum=1**  **1.发送方发送数据过程**  (a).如果nextseqnum<base+N,则发送数据包Data[nextseqnum]，并使seqnum++  (b).对Data[nextseqnum]启动一个计时器  **2.发送方接收到ACK**  (a).得知获得的ACK的序号，如获得ACK\_ack  (b).结束send\_base到ack间所有的计时器  (c).send\_base=ack+1  **3.发送方的计时器x超时**  (a).重发分组Data[x]  (b).重启计时器x  **设发送方窗口大小为N，rev\_base代表窗口的第一个分组，未接收到ACK**  **接收方：设置rev\_base=1**   1. 接收方接收到数据包   (a).提取数据包的seq  ①如果rev\_base<=seq<rev\_base+N，发送ACK\_seq，如果seq\_base等于seq，则滑动窗口  ②如果rev\_base-N<=seq<rev\_base，发送ACK\_seq  ③都不满足，则不做任何处理  **三、数据分组丢失验证模拟方法：**  ①将数据包的序号取模，如序号为x，如果x%7==0，则不发送该数据包。  ②将ACK的序号也取模，如序号为x，如果x%6==0，则不发送该ACK。  **四、GBN协议的主要类和函数的作用**  1.GBN的发送方GBNone  ①开启计时器timerBegin()    ②结束计时器timerEnd()    ③读取要传送的文件，在这里文件是一首英文歌。    ④发送数据的主要函数Send(),调用了函数SendTorRceiver和ReceiveACK    ⑥发送数据给接收方，这里要先给定序号，然后转回字节数组才能进行传输。并且要在这里对于计时器进  行重新设置。    ⑦接收ACK函数，并且对于send\_base做更新。    **2.GBN函数接收方**  ①解析收到的ACK数据包并发调用函数发回ACK\_seq    ②发送ACK使用的函数，被receive()调用    **五、SR协议的主要类和函数的作用**  1.SR的发送方SRone  ①开启计时器，q为分组序号，x为窗口序号，使用的是Execuors    ②结束计时器    ③读取要传送的文件，这里是一首英文歌，一共可分为19个分组    ④发送数据的主要函数send(),调用了函数SendToReceiver和ReceiveACK    ⑥发送数据给接收方，并对每一个数据包计时，要加上序号并转化为字节数组。    ⑦接收ACK函数    **2.SR函数接收方**  ①解析收到的ACK数据包并发调用函数发回ACK\_seq，并且判断窗口是否需要滑动。  如果需要，则进行窗口的滑动    ②发送ACK使用的函数，被receive()调用     1. **多线程函数run()，为了实现全双工数据传输（双向、同时传输）** |
| 实验结果： |
| 1. **GBN发送方：**   **窗口大小为8，发送20个数据包。**    **模拟的是数据包%5为0的丢失，则数据包0、5、10会丢失，且模拟ACK%6为0丢失。**  **可以看出，丢失以后，经过一段超时时间，由于第0个包丢失，那么要重发分组0-7(窗口大小为8)。**  **而ACK6丢失，但由于GBN是累积确认的，发送方受到ACK7，则也能知道ACK6也受到了，因此没有影响。**   1. **GBN接收方：**     **GBN接收方在接收到一个数据包后，就会提取seq并返回ACK\_seq，但这是在受到的seq和期望受到的seq相同的情况下。**  **如果收到的是不是期望的seq，则会返回上次确认的seq(如图中返回多个ACK9)。**   1. **SR发送方**     **SR发送方几乎和GBN一样，只有一点：分组丢失后，只重发丢失的分组。**  **如图中的分组0丢失，但是只重发分组0，没有将0-7一起重发。**   1. **SR接收方**       **SR接收方每次接收一个数据包就返回对应的ACK**  **我这里将接收方窗口设置为5**  **因此在收到数据包1-4后，窗口没有滑动，只有在收到ACK0的时候，接收窗口滑动。** |
| 问题讨论： |
| 实验中的GBN协议事实上还是比较好实现的，但是SR和双向传输比较困难。主要的难点在于取模后的数据包和ACK序号(它们的序号小于<N,设序号范围是0-N)。  后来我仔细研究了书本与spoc的ppt，推出了一个数学公式，可以根据send\_base和rev\_base，将ACK序号还原，看是哪一个数据包被确认。这一点在GBN里面其实用处不大，但是在SR里边很重要，决定了更新send\_base和rev\_base的正确性。  其次，对于双向数据传输，我采用的是开启多线程，协议的一方一个线程是接收，一个线程负责发送，这样就实现了全双工的数据传输。 |
| 心得体会： |
| 这一次实验事实上使我对于可靠数据传输的方法有了更多深刻的了解。比如停等协议，发送方和接收方窗口都是1。而GBN协议的发送方窗口可以是N，到了SR协议，发送方和接收方的窗口大小可以自己设置(只要注意它们的和要小于序列号范围)。  另外，对于双向数据传输，我一直想要找到一个全双工的方法来解决。后来，我使用的是多线程的技术，完美解决了此问题，让我对于java多线程的使用和应用都有了新的理解。 |
| 源代码： |
| **①GBNone.java**  package cnExperiment2;  import java.io.BufferedReader;  import java.io.File;  import java.io.FileReader;  import java.io.IOException;  import java.net.DatagramPacket;  import java.net.DatagramSocket;  import java.net.InetAddress;  import java.util.TimerTask;  import java.util.concurrent.Executors;  import java.util.concurrent.ScheduledExecutorService;  import java.util.concurrent.TimeUnit;  public class GBNone implements Runnable{  private static int seqnum=15;    private static int N = 8;  private int choice;    public GBNone(int choice) {  super();  this.choice = choice;  }  // 发送数据部分  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 boolean flag = false;  private static int timeout = 2;  private static String filestring = 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  \*/  private void timerBegin() {  TimerTask task=new TimerTask() {  @Override  public void run() {  if (send\_base>= Math.ceil(B.length / 1478)-1) {  return;  }  try {  for (int i = send\_base; i < nextseqnum; i++) {  byte[] tempb = getByteArray(i);  String temp = new String(tempb);  String s = new String(i%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:" + i%seqnum+" 重发的是第"+i+"个包");  timerBegin();  }  } catch (Exception e) {  }  }  };    if (!flag) {  flag = true;  } else {  executor.shutdown();  //timer.cancel();  //timer.purge();  }  executor=Executors.newSingleThreadScheduledExecutor();  executor.scheduleWithFixedDelay(task, timeout, timeout, TimeUnit.SECONDS);    }  /\*\*  \* 结束计时  \*/  private void timerEnd() {  if (flag) {  //timer.purge();  //timer.cancel();  executor.shutdown();  flag = false;  }  }  /\*\*  \* 读取文件，存入filestring和字节数组B中  \*  \* @param fileName  \*/  public static void readFileByLines(String fileName) {  File file = new File(fileName);  BufferedReader reader = null;  try {  reader = new BufferedReader(new FileReader(file));  String tempString = null;  while ((tempString = reader.readLine()) != null) {  filestring = filestring + tempString + "\r\n";  }  reader.close();  B = filestring.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  \*/  private byte[] getByteArray(int nextseqnum) {  byte[] temp = new byte[1478];  for (int i = 0; i < 1478; i++) {  if (nextseqnum \* 1478 + i >= B.length) {  break;  }  temp[i] = B[nextseqnum \* 1478 + i];  }  return temp;  }  /\*\*  \* 发送数据  \*/  public void send() {  readFileByLines("test.txt");  team=(int) Math.ceil(B.length/1478);  try {  SenderSocket = new DatagramSocket();  ReceiveAckSocket = new DatagramSocket(ReceiveAckPort);  while (true) {  SendToReciver();  ReceiveACK();  if (send\_base >= team) {  break;  }  }  System.out.println("Send Over");  } catch (Exception e) {  }  }  /\*\*  \* 发送数据给接收方  \*/  private void SendToReciver() {  try {  while (nextseqnum < send\_base + N) {  if (send\_base>= team||nextseqnum>=team) {  break;  }  byte[] tempb = getByteArray(nextseqnum);  String temp = new String(tempb);  String 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) {  timerBegin();  }  nextseqnum++;    // System.out.println(nextseqnum);  }  } catch (Exception e) {  e.printStackTrace();  }  }  /\*\*  \* 接收ACK  \* @throws InterruptedException  \*/  private 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++) {  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("ACK" + ack);  //send\_base = Math.max(ack + 1, send\_base);  int m;  //System.out.println(send\_base);  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");  }  TimerReset();  //System.out.println("base="+send\_base);  //System.out.println("next\_seq="+nextseqnum);  } catch (IOException e) {  }  }  /\*\*  \* 重置计时器  \*/  private void TimerReset() {  if (send\_base == nextseqnum) {  timerEnd();  } else {  timerBegin();  }  }  // 接收数据部分  private static int SendAckPort = 10243;  private static int ReceiveDataPort = 10242;  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);  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;  }  }  // 判断是否是顺序到达的  if (expectedSeqNum == Integer.valueOf(num)) {  int ack = expectedSeqNum;  sendACKback(ack);  expectedSeqNum = (expectedSeqNum + 1)%seqnum;  last=ack;  }  else {  if (last>=0) {  sendACKback(last);  }  }  }  } catch (Exception e) {  }  }  /\*\*  \* 发回ACK  \*  \* @param ack  \* 发回的ACK序号，为0到N-1  \*/  private void sendACKback(int ack) {  try {  ackFrame ACK = new ackFrame(ack);  SendAckPacket = new DatagramPacket(ACK.ackByte, ACK.ackByte.length, InetAddress.getLocalHost(),  SendAckPort);  ReciverSocket.send(SendAckPacket);  System.out.println("Send ACK" + ack + " Back");  } catch (Exception e) {  }  }  @Override  public void run() {  if (choice==0) {  send();  }  else if (choice==1){  receive();  }  }    public static void main(String[] args) {  new Thread(new GBNone(0)).start();  new Thread(new GBNone(1)).start();  }  }  **②GBNtwo.java**  **package cnExperiment2;**  **import java.io.BufferedReader;**  **import java.io.File;**  **import java.io.FileReader;**  **import java.io.IOException;**  **import java.net.DatagramPacket;**  **import java.net.DatagramSocket;**  **import java.net.InetAddress;**  **import java.util.TimerTask;**  **import java.util.concurrent.Executors;**  **import java.util.concurrent.ScheduledExecutorService;**  **import java.util.concurrent.TimeUnit;**  **public class GBNtwo implements Runnable{**  **private static int N = 8;**  **private int choice;**  **public GBNtwo(int choice) {**  **super();**  **this.choice = choice;**  **}**  **// 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);**  **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;**  **}**  **}**  **// 判断是否是顺序到达的**  **if (expectedSeqNum == Integer.valueOf(num)) {**  **int ack = expectedSeqNum;**  **sendACKback(ack);**  **expectedSeqNum = (expectedSeqNum + 1)%seqnum;**  **last=ack;**  **}**  **else {**  **if (last>=0) {**  **sendACKback(last);**  **}**  **}**  **}**  **} catch (Exception e) {**  **}**  **}**  **/\*\***  **\* 发回ACK**  **\***  **\* @param ack**  **\* 发回的ACK序号，为0到N-1**  **\*/**  **private void sendACKback(int ack) {**  **try {**  **ackFrame ACK = new ackFrame(ack);**  **SendAckPacket = new DatagramPacket(ACK.ackByte, ACK.ackByte.length, InetAddress.getLocalHost(),**  **SendAckPort);**  **ReciverSocket.send(SendAckPacket);**  **System.out.println("Send ACK" + ack + " Back");**  **} catch (Exception e) {**  **}**  **}**  **// 发送数据部分**  **private static int seqnum=15;**  **private static int SendDataPort = 10242;**  **private static int ReceiveAckPort = 10243;**  **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;**  **private static int timeout = 2;**  **private static String filestring = 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**  **\*/**  **private void timerBegin() {**  **TimerTask task=new TimerTask() {**  **@Override**  **public void run() {**  **if (send\_base>= Math.ceil(B.length / 1478)-1) {**  **return;**  **}**  **try {**  **for (int i = send\_base; i < nextseqnum; i++) {**  **byte[] tempb = getByteArray(i);**  **String temp = new String(tempb);**  **String s = new String(i%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:" + i%seqnum+" 重发的是第"+i+"个包");**  **timerBegin();**  **}**  **} catch (Exception e) {**  **}**  **}**  **};**    **if (!flag) {**  **flag = true;**  **} else {**  **executor.shutdown();**  **//timer.cancel();**  **//timer.purge();**  **}**  **executor=Executors.newSingleThreadScheduledExecutor();**  **executor.scheduleWithFixedDelay(task, timeout, timeout, TimeUnit.SECONDS);**    **}**  **/\*\***  **\* 结束计时**  **\*/**  **private void timerEnd() {**  **if (flag) {**  **//timer.purge();**  **//timer.cancel();**  **executor.shutdown();**  **flag = false;**  **}**  **}**  **/\*\***  **\* 读取文件，存入filestring和字节数组B中**  **\***  **\* @param fileName**  **\*/**  **public static void readFileByLines(String fileName) {**  **File file = new File(fileName);**  **BufferedReader reader = null;**  **try {**  **reader = new BufferedReader(new FileReader(file));**  **String tempString = null;**  **while ((tempString = reader.readLine()) != null) {**  **filestring = filestring + tempString + "\r\n";**  **}**  **reader.close();**  **B = filestring.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**  **\*/**  **private byte[] getByteArray(int nextseqnum) {**  **byte[] temp = new byte[1478];**  **for (int i = 0; i < 1478; i++) {**  **if (nextseqnum \* 1478 + i >= B.length) {**  **break;**  **}**  **temp[i] = B[nextseqnum \* 1478 + i];**  **}**  **return temp;**  **}**  **/\*\***  **\* 发送数据**  **\*/**  **public void send() {**  **readFileByLines("test.txt");**  **team=(int) Math.ceil(B.length/1478);**  **try {**  **SenderSocket = new DatagramSocket();**  **ReceiveAckSocket = new DatagramSocket(ReceiveAckPort);**  **while (true) {**  **SendToReciver();**  **ReceiveACK();**  **if (send\_base >= team) {**  **break;**  **}**  **}**  **System.out.println("Send Over");**  **} catch (Exception e) {**  **}**  **}**  **/\*\***  **\* 发送数据给接收方**  **\*/**  **private void SendToReciver() {**  **try {**  **while (nextseqnum < send\_base + N) {**  **if (send\_base>= team||nextseqnum>=team) {**  **break;**  **}**  **byte[] tempb = getByteArray(nextseqnum);**  **String temp = new String(tempb);**  **String 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) {**  **timerBegin();**  **}**  **nextseqnum++;**    **// System.out.println(nextseqnum);**  **}**  **} catch (Exception e) {**  **e.printStackTrace();**  **}**  **}**  **/\*\***  **\* 接收ACK**  **\* @throws InterruptedException**  **\*/**  **private 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++) {**  **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("ACK" + ack);**  **//send\_base = Math.max(ack + 1, send\_base);**  **int m;**  **//System.out.println(send\_base);**  **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");**  **}**  **TimerReset();**  **//System.out.println("base="+send\_base);**  **//System.out.println("next\_seq="+nextseqnum);**  **} catch (IOException e) {**  **}**  **}**  **/\*\***  **\* 重置计时器**  **\*/**  **private void TimerReset() {**  **if (send\_base == nextseqnum) {**  **timerEnd();**  **} else {**  **timerBegin();**  **}**  **}**    **@Override**  **public void run() {**  **if (choice==0) {**  **receive();**  **}**  **else if (choice==1){**  **send();**  **}**  **}**  **public static void main(String[] args) {**  **new Thread(new GBNtwo(0)).start();**  **new Thread(new GBNtwo(1)).start();**  **}**  **}**  **③SRone.java**  **package cnExperiment2;**  **import java.io.BufferedReader;**  **import java.io.File;**  **import java.io.FileReader;**  **import java.io.IOException;**  **import java.net.DatagramPacket;**  **import java.net.DatagramSocket;**  **import java.net.InetAddress;**  **import java.util.TimerTask;**  **import java.util.concurrent.Executors;**  **import java.util.concurrent.ScheduledExecutorService;**  **import java.util.concurrent.TimeUnit;**  **public class SRone implements Runnable{**  **private static int seqnum = 16;**  **private int choice;**  **public SRone(int choice) {**  **super();**  **this.choice = choice;**  **}**  **// 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 String filestring = new String();**  **private static byte[] B;**  **private static int team;**  **private static boolean[] ackarray = new boolean[N];**  **private static boolean[] flags = new boolean[N];**  **private static ScheduledExecutorService[] executors = new ScheduledExecutorService[N];**  **/\*\***  **\* 开始计时或者重新计时，超时时间为2s**  **\*/**  **private void timerBegin(int q, int x) {**  **TimerTask task = new TimerTask() {**  **@Override**  **public void run() {**  **if (send\_base >= Math.ceil(B.length / 1478) - 1) {**  **return;**  **}**  **try {**  **byte[] tempb = getByteArray(x);**  **String temp = new String(tempb);**  **String 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**  **\*/**  **private void timerEnd(int q, int x) {**  **/\***  **\* if (flag) { //timer.purge(); //timer.cancel(); executors[q].shutdown(); flag**  **\* = false; }**  **\*/**  **if (flags[q]) {**  **flags[q] = false;**  **executors[q].shutdown();**  **}**  **}**  **/\*\***  **\* 读取文件，存入filestring和字节数组B中**  **\***  **\* @param fileName**  **\*/**  **public static void readFileByLines(String fileName) {**  **File file = new File(fileName);**  **BufferedReader reader = null;**  **try {**  **reader = new BufferedReader(new FileReader(file));**  **String tempString = null;**  **while ((tempString = reader.readLine()) != null) {**  **filestring = filestring + tempString + "\r\n";**  **}**  **reader.close();**  **B = filestring.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**  **\*/**  **private byte[] getByteArray(int nextseqnum) {**  **byte[] temp = new byte[1478];**  **for (int i = 0; i < 1478; i++) {**  **if (nextseqnum \* 1478 + i >= B.length) {**  **break;**  **}**  **temp[i] = B[nextseqnum \* 1478 + i];**  **}**  **return temp;**  **}**  **/\*\***  **\* 发送数据**  **\*/**  **public void send() {**  **readFileByLines("test.txt");**  **team = (int) Math.ceil(B.length / 1478);**  **try {**  **SenderSocket = new DatagramSocket();**  **ReceiveAckSocket = new DatagramSocket(ReceiveAckPort);**  **while (true) {**  **SendToReciver();**  **ReceiveACK();**  **if (send\_base >= team) {**  **break;**  **}**  **}**  **System.out.println("Send Over");**  **} catch (Exception e) {**  **}**  **}**  **/\*\***  **\* 发送数据给接收方**  **\*/**  **private void SendToReciver() {**  **try {**  **while (nextseqnum < send\_base + N) {**  **if (send\_base >= team || nextseqnum >= team) {**  **break;**  **}**  **byte[] tempb = getByteArray(nextseqnum);**  **String temp = new String(tempb);**  **String 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);**  **timerBegin(nextseqnum - send\_base, nextseqnum);**  **/\***  **\* if (send\_base == nextseqnum) { timerBegin(); }**  **\*/**  **nextseqnum++;**  **// System.out.println(nextseqnum);**  **}**  **} catch (Exception e) {**  **e.printStackTrace();**  **}**  **}**  **/\*\***  **\* 接收ACK**  **\***  **\* @throws InterruptedException**  **\*/**  **private 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++) {**  **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("ACK" + ack);**  **int a = ack, b = ack + seqnum;**  **while (!(send\_base >= a && send\_base <= b)) {**  **a += seqnum;**  **b += seqnum;**  **}**  **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) {**  **// sendACKback(rev\_num);**  **ackarray[ack - send\_base] = true;**  **timerEnd(ack - send\_base, ack);**  **if (ack == send\_base) {**  **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;**  **// timerEnd(i, ack);**  **// executors[i];**  **// flags[i]=false;**  **}**  **send\_base = send\_base + cnt;**  **}**  **}**  **// } else {**  **// System.out.println("模拟ACK" + ack + "丢失");**  **// }**  **} catch (IOException e) {**  **}**        **}**  **// 接收数据部分**  **private static int SendAckPort = 10243;**  **private static int ReceiveDataPort = 10242;**  **private static DatagramSocket ReciverSocket;**  **private static DatagramPacket SendAckPacket;**  **private static int RecWinSize=5;**  **private static int rev\_base=0;**  **private static boolean ack[]=new boolean[RecWinSize];**    **/\*\***  **\* 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);**  **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+RecWinSize) {**  **sendACKback(rev\_num);**  **if (rev\_num==rev\_base) {**  **ack[0]=true;**  **int cnt=0;**  **for (int i=0;i<RecWinSize;i++) {**  **if (ack[i]) {**  **cnt++;**  **}**  **else {**  **break;**  **}**  **}**  **for (int i=0;i<RecWinSize-cnt;i++) {**  **ack[i]=ack[i+cnt];**  **}**  **for (int i=RecWinSize-cnt;i<RecWinSize;i++) {**  **ack[i]=false;**  **}**  **rev\_base=rev\_base+cnt;**  **}**  **else {**  **ack[rev\_num-rev\_base]=true;**  **}**  **}**  **else if (rev\_num<rev\_base&&rev\_num>=rev\_base-RecWinSize) {**  **sendACKback(rev\_num);**  **}**  **}**  **} catch (Exception e) {**  **}**  **}**  **/\*\***  **\* 发回ACK**  **\***  **\* @param ack**  **\* 发回的ACK序号，为0到N-1**  **\*/**  **private void sendACKback(int ack) {**  **try {**  **ackFrame ACK = new ackFrame(ack);**  **SendAckPacket = new DatagramPacket(ACK.ackByte, ACK.ackByte.length, InetAddress.getLocalHost(),**  **SendAckPort);**  **ReciverSocket.send(SendAckPacket);**  **System.out.println("Send ACK" + ack + " Back");**  **} catch (Exception e) {**  **}**  **}**    **@Override**  **public void run() {**  **if (choice==0) {**  **send();**  **}**  **else if (choice==1){**  **receive();**  **}**  **}**    **public static void main(String[] args) {**  **new Thread(new SRone(0)).start();**  **new Thread(new SRone(1)).start();**  **}**  **}**  **④SRtwo.java**  **package cnExperiment2;**  **import java.io.BufferedReader;**  **import java.io.File;**  **import java.io.FileReader;**  **import java.io.IOException;**  **import java.net.DatagramPacket;**  **import java.net.DatagramSocket;**  **import java.net.InetAddress;**  **import java.util.TimerTask;**  **import java.util.concurrent.Executors;**  **import java.util.concurrent.ScheduledExecutorService;**  **import java.util.concurrent.TimeUnit;**  **public class SRtwo implements Runnable{**    **int choice;**  **public SRtwo(int choice) {**  **super();**  **this.choice = choice;**  **}**    **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 RecWinSize=5;**  **private static int rev\_base=0;**  **private static boolean ack[]=new boolean[RecWinSize];**    **/\*\***  **\* 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);**  **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+RecWinSize) {**  **sendACKback(rev\_num);**  **if (rev\_num==rev\_base) {**  **ack[0]=true;**  **int cnt=0;**  **for (int i=0;i<RecWinSize;i++) {**  **if (ack[i]) {**  **cnt++;**  **}**  **else {**  **break;**  **}**  **}**  **for (int i=0;i<RecWinSize-cnt;i++) {**  **ack[i]=ack[i+cnt];**  **}**  **for (int i=RecWinSize-cnt;i<RecWinSize;i++) {**  **ack[i]=false;**  **}**  **rev\_base=rev\_base+cnt;**  **}**  **else {**  **ack[rev\_num-rev\_base]=true;**  **}**  **}**  **else if (rev\_num<rev\_base&&rev\_num>=rev\_base-RecWinSize) {**  **sendACKback(rev\_num);**  **}**  **}**  **} catch (Exception e) {**  **}**  **}**  **/\*\***  **\* 发回ACK**  **\***  **\* @param ack**  **\* 发回的ACK序号，为0到N-1**  **\*/**  **private void sendACKback(int ack) {**  **try {**  **ackFrame ACK = new ackFrame(ack);**  **SendAckPacket = new DatagramPacket(ACK.ackByte, ACK.ackByte.length, InetAddress.getLocalHost(),**  **SendAckPort);**  **ReciverSocket.send(SendAckPacket);**  **System.out.println("Send ACK" + ack + " Back");**  **} catch (Exception e) {**  **}**  **}**      **// 发送数据部分**  **private static int N = 8;**  **private static int SendDataPort = 10242;**  **private static int ReceiveAckPort = 10243;**  **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 String filestring = new String();**  **private static byte[] B;**  **private static int team;**  **private static boolean[] ackarray = new boolean[N];**  **private static boolean[] flags = new boolean[N];**  **private static ScheduledExecutorService[] executors = new ScheduledExecutorService[N];**  **/\*\***  **\* 开始计时或者重新计时，超时时间为2s**  **\*/**  **private void timerBegin(int q, int x) {**  **TimerTask task = new TimerTask() {**  **@Override**  **public void run() {**  **if (send\_base >= Math.ceil(B.length / 1478) - 1) {**  **return;**  **}**  **try {**  **byte[] tempb = getByteArray(x);**  **String temp = new String(tempb);**  **String 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**  **\*/**  **private void timerEnd(int q, int x) {**  **/\***  **\* if (flag) { //timer.purge(); //timer.cancel(); executors[q].shutdown(); flag**  **\* = false; }**  **\*/**  **if (flags[q]) {**  **flags[q] = false;**  **executors[q].shutdown();**  **}**  **}**  **/\*\***  **\* 读取文件，存入filestring和字节数组B中**  **\***  **\* @param fileName**  **\*/**  **public static void readFileByLines(String fileName) {**  **File file = new File(fileName);**  **BufferedReader reader = null;**  **try {**  **reader = new BufferedReader(new FileReader(file));**  **String tempString = null;**  **while ((tempString = reader.readLine()) != null) {**  **filestring = filestring + tempString + "\r\n";**  **}**  **reader.close();**  **B = filestring.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**  **\*/**  **private byte[] getByteArray(int nextseqnum) {**  **byte[] temp = new byte[1478];**  **for (int i = 0; i < 1478; i++) {**  **if (nextseqnum \* 1478 + i >= B.length) {**  **break;**  **}**  **temp[i] = B[nextseqnum \* 1478 + i];**  **}**  **return temp;**  **}**  **/\*\***  **\* 发送数据**  **\*/**  **public void send() {**  **readFileByLines("test.txt");**  **team = (int) Math.ceil(B.length / 1478);**  **try {**  **SenderSocket = new DatagramSocket();**  **ReceiveAckSocket = new DatagramSocket(ReceiveAckPort);**  **while (true) {**  **SendToReciver();**  **ReceiveACK();**  **if (send\_base >= team) {**  **break;**  **}**  **}**  **System.out.println("Send Over");**  **} catch (Exception e) {**  **}**  **}**  **/\*\***  **\* 发送数据给接收方**  **\*/**  **private void SendToReciver() {**  **try {**  **while (nextseqnum < send\_base + N) {**  **if (send\_base >= team || nextseqnum >= team) {**  **break;**  **}**  **byte[] tempb = getByteArray(nextseqnum);**  **String temp = new String(tempb);**  **String 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);**  **timerBegin(nextseqnum - send\_base, nextseqnum);**  **/\***  **\* if (send\_base == nextseqnum) { timerBegin(); }**  **\*/**  **nextseqnum++;**  **// System.out.println(nextseqnum);**  **}**  **} catch (Exception e) {**  **e.printStackTrace();**  **}**  **}**  **/\*\***  **\* 接收ACK**  **\***  **\* @throws InterruptedException**  **\*/**  **private 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++) {**  **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("ACK" + ack);**  **int a = ack, b = ack + seqnum;**  **while (!(send\_base >= a && send\_base <= b)) {**  **a += seqnum;**  **b += seqnum;**  **}**  **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) {**  **// sendACKback(rev\_num);**  **ackarray[ack - send\_base] = true;**  **timerEnd(ack - send\_base, ack);**  **if (ack == send\_base) {**  **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;**  **// timerEnd(i, ack);**  **// executors[i];**  **// flags[i]=false;**  **}**  **send\_base = send\_base + cnt;**  **}**  **}**  **// } else {**  **// System.out.println("模拟ACK" + ack + "丢失");**  **// }**  **} catch (IOException e) {**  **}**  **}**    **@Override**  **public void run() {**  **if (choice==0) {**  **receive();**  **}**  **else if (choice==1){**  **send();**  **}**  **}**    **public static void main(String[] args) {**  **new Thread(new SRtwo(0)).start();**  **new Thread(new SRtwo(1)).start();**  **}**  **}**  **⑤ackFrame.java**  **package** cnExperiment2;  **public** **class** ackFrame {  **protected** **int** ACK;  **protected** String ack;  **protected** **byte**[] ackByte;    **public** ackFrame(**int** ACK) {  **this**.ACK=ACK;  ack=String.*valueOf*(ACK);  ackByte=ack.getBytes();  }      } |