**重庆大学Project报告**

Project题目： 共享白板系统

学 院： 计算机学院

专业班级： 计算机科学技术、计算机科学技术、

信息安全、信息安全

年 级： 2018

姓 名：陈江渝、许康乐、邹艳丽、周泽峻

学 号：20180007、20184969、20181759、20184228

完成时间： 2020 年 12 月 29 日

成 绩：

指导教师：

重庆大学教务处制

Project教师评定成绩表

A. 课程设计的成绩构成

|  |  |  |  |
| --- | --- | --- | --- |
| 课程设计项目 | 该项目标准分值 | 评定分值（100分） | |
| 项目完成质量（含检查） | 项目报告 |
| **项目1** | 10 | 80% | 20% |

B. 课程设计项目的成绩评定标准

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 项目 | 得分情况 | | | | |
| 优秀  (90～100) | 良好  (80～89) | 中等  (70～79 | 及格  60～69 | 不及格  0～59 |
| 参考标准 | 参考标准 | 参考标准 | 参考标准 | 参考标准 |
| 项目完成质量 | 1.分工明确，独立完成课程设计项目，有源码  2.设计合理，功能完善，满足课程设计全部要求，且有拓展  3.界面友好，Bug极少，针对异常情况有处理  4.汇报检查时，讲解清晰，演示流畅，能正确回答问题 | 1.分工明确，独立完成课程设计项目，有源码  2.设计合理，功能完善，满足课程设计全部要求  3.界面良好，Bug较少，针对异常情况有处理  4.汇报检查时，讲解清晰，演示流畅，能正确回答问题 | 1.独立完成课程设计项目，有源码  2.功能较完善，基本满足课程设计要求  3.界面一般，Bug较多，无异常情况有处理  4.汇报检查时，讲解较清晰，完成演示，基本能正确回答问题 | 1.独立完成课程设计项目，有源码  2.功能基本完成，基本满足课程设计要求  3.界面一般，Bug很多，无异常情况有处理  4.汇报检查时，能基本讲清楚，主要功能能演示，基本能正确回答问题 | 1.抄袭或被抄袭以0分计，没有参与课程设计检查以0分计；  2.未能独立完成课程设计项目；  3.不能通过检查，不能进行课程设计汇报，主要功能无法演示，不能正确回答问题。 |
| 项目报告 | 1.结构严谨，逻辑清晰  2.文档规范，文字、图表表达清楚，用词专业，完全符合规范化要求  3.计算机打印  4.有完整的项目设计方案，采用的理论技术合理正确，结果及分析正确，功能有拓展  5.报告内容与课程设计检查结果相符 | 1.结构合理，符合逻辑；  2. 文档规范，文字、图表表达较清楚，用词专业，符合规范化要求  3.计算机打印  4.有较完整的项目设计方案，采用的理论与技术正确，结果及分析正确  5.报告内容与课程设计检查结果相符 | 1.结构合理，符合逻辑  2. 文档较规范，文字、图表表达较清楚，基本达到规范化要求  3.计算机打印  4.项目设计方案基本完整，采用的理论与技术基本正确，结果及分析基本正确。  5.报告内容与课程设计检查结果相符 | 1.结构基本合理,逻辑基本清楚  2. 文档较规范，文字、图表表达较清楚，勉强达到规范要求  3.计算机打印  4.项目设计方案有欠缺，采用的理论与技术基本可行，没有测试结果与结果分析  5.报告内容与课程设计检查结果相符 | 1.抄袭与被抄袭以0分计；  2.结构混乱，内容空泛，文字表达不清，错别字较多，达不到规范化要求  3.人工书写或计算机打印  4.报告内容与课程设计检查结果不相符 |

说明：评定分值要折算成项目标准分值。课程设计项目得分 = 课程设计项目的评定分值\*10%

指导教师评定成绩：

指导教师签名： 年 月 日

课题任务分工说明

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 角色 | 承担的任务 | 备注 |
| 1 |  |  | 组长 | （1）  （2）  （3）汇总各成员代码，并进行相应调整和部分改动 |  |
| 2 |  |  | 组员 | （1）  （2）  （3）  …… |  |
| 3 |  |  | 组员 | （1）  （2）  （3）  …… |  |
| 4 |  |  | 组员 | （1）  （2）  （3）  …… |  |

说明：应该把每个人承担的任务写清楚、仔细

目录

[1 需求分析 2](#_Toc390336267)

[2 系统设计 2](#_Toc390336268)

[2.1 系统分析 2](#_Toc390336269)

[2.2 系统结构 2](#_Toc390336270)

[3 关键代码（技术）描述 2](#_Toc390336271)

[3.1 Buffer 2](#_Toc390336272)

[3.1.2 Buffer类的重要属性： 2](#_Toc390336273)

[3.1.2 Buffer类的重要方法： 2](#_Toc390336274)

[3.2 DataLinkLayer 2](#_Toc390336275)

[3.2.1 DataLinkLayer 类的重要属性： 2](#_Toc390336276)

[3.2.1 DataLinkLayer 类的重要方法： 2](#_Toc390336277)

[3.3 PhysicalLayer 2](#_Toc390336278)

[3.3.1 PhysicalLayer 类重要属性： 2](#_Toc390336279)

[3.3.2 PhysicalLayer 类重要方法： 2](#_Toc390336280)

[3.4 Timer 2](#_Toc390336281)

[3.4.1 Timer 类重要属性： 2](#_Toc390336282)

[3.4.2 Timer 类重要方法： 2](#_Toc390336283)

[4 系统测试运行 2](#_Toc390336284)

[4.1 无丢失，无损坏情况下的Go-back-n滑动窗口协议测试 2](#_Toc390336285)

[4.1.1 双方正常发送数据：正常 2](#_Toc390336286)

[4.1.2 一方不发送数据：正常 2](#_Toc390336287)

[4.2 有丢失，损坏情况下的Go-back-n滑动窗口协议 2](#_Toc390336288)

[4.2.1 窗口大小都1：正常 2](#_Toc390336289)

[4.2.2 双发都发数据：正常 2](#_Toc390336290)

[4.2.3 只有一方发数据：正常 2](#_Toc390336291)

5 [总结 2](#_Toc390336295)

# 1 需求分析

**共享白板系统**

* 实现基于服务器转发的多点间的文字、作图等信息的实时同步共享。
* 作图包括笔刷、划线、矩形、圆形等，可以调节粗细、颜色、清屏。
* 实时同步在线成员列表。
* 所有参与者均可发起共享或取消已有的共享。
* 其他参与者要能及时展示出发起或取消的共享信息。
* 所有参与者实时同步白板内容。
* 新上线的用户可以显示历史白板记录。
* 基于SOCKET编程实现
* 传输层可采用TCP或UDP协议
* 能够具有良好的图形界面。

# 2 系统设计

## 2.1 系统分析

系统实现的是一个基于服务器转发的多点间的文字、作图等信息的实时同步共享的共享白板系统。服务端和客户端基于SOCKET编程实现，传输层采用TCP协议。

服务端通过多线程处理多个客户端的连接，主要负责解析客户端的请求，并进行相应的处理。请求/响应主要有四种类型：

1. **分配用户ID**
2. **在线用户信息**
3. **发送画图数据**
4. **用户断开连接**

一个基于UDP的，Go-back-n的滑动窗口模拟。涉及的协议层有物理层，数据链路层和网络层，其中数据链路层是核心协议层。

在Go-back-n的滑动窗口中，数据链路层的主要功能是处理帧到达 (frame arrived) 、超时 (time out) 和网络层准备就绪 (Network Layer ready) 这三件事。现对这三个事件分别分析如下

1. **帧到达 frame arrived**

新帧的到达涉及到物理层和数据链路层。

物理层总是监听者信号通道，当有新的一帧到达时，接收，校对验错码，通知数据链路层。

对于数据链路层，当接收到物理层帧到达的通知后，有两种可能的行为：一种是物理层接收到的是损坏的帧，那么数据链路层不做任何反应；另外一种情况是接收到的是完好的、正确的帧，那么数据链路层①首先从数据链路层接收帧并拆分，②然后验证新帧是否是希望收到的帧，如果是，则将数据发送给网路程并将期望的帧递增，否则丢弃新帧的数据，③最后检查帧的确认好是否在滑动窗口内，如果是，则移动滑动窗口。

1. **超时 time out**

超时仅仅影响到数据链路层。

每次发送新帧时，计时器开始计时。当在规定的时间内，没有收到确认时，计时器通知数据链路层帧超时。

数据链路层发现超时事件后，重发所有发送窗口缓存的帧，相应的计时器重新开始计时。

1. **网络层准备就绪 Network Layer ready**

网路层准备就绪将会影响到网络层和数据链路层。

当网络层有数据想要发送时，向数据链路层发送网络层准备就绪的标志。

当数据链路层发现网络层准备就绪时，首先从网络层接收报，然后封装消息成帧，最后将帧发送给数据链路层。

## 2.2 系统设计

### 2.2.1 帧设计

帧的设计如下所示:

|  |  |  |  |
| --- | --- | --- | --- |
| 8bit | 8bit | 8bit | 0-480bit |
| 帧序号 | 确认帧 | 检错码 | 数据 |

1. 帧序号：长度为8bit，序号从0-255循环
2. 确认帧：长度为8bit，发送窗口最大为254
3. 检错码：长度为8bit，为了简化模拟程序，将检错码提前至此
4. 数据：0-480bit，

### 2.3.2 超时处理

每次发送帧之后，启动计时器，如果在规定时间内未收到确认帧，重发发送窗口内的所有帧。

### 2.3.3 确认机制

采用捎带确认机制，但是未考虑一方长时间没有帧发送的情况。

### 2.2.2 类结构

通过2.1节的分析，结合实际，我们可以发现整个系统应该如下核心部分：

1. DataLinkLayer 类

* 应该是一个线程的子类，时刻检查着是否有新事件发生
* 能够向网络层传输包和从网络层接收包
* 能够从物理层接收帧和向物理层接受帧
* 能够处理三类事件
* 当自己的发送窗口满是，禁止再从网络层接收数据

图 1

1. PhysicalLayer 类

* 应该是一个线程的子类，时刻监听者信道
* 当有新帧到达时，接收并缓存，想数据链路层报告新帧到达
* 可以向其他主机传送信息

图 2

1. Host 类

* 实现NetworkLayer的功能
* 提供一个良好的人机界面

图 3

1. Timer 类

* 应该是一个线程的子类，时刻统计着时间
* 当时间超出定额时，能向数据链路层报告超时发生

图4

1. Buffer 类

* 能够模拟真实的内存空间
* 缓存物理层收到的帧
* DataLinkLayer、PhysicalLayer、TimerLayer和Timer都应该可以访问及修改Buffer中的指定内容

图 5

C:\Users\fey\Desktop\类图.tif类与类之间的相互关系如下所示:

图 6

# 3 关键代码（技术）描述

## 3.1 Buffer

### 3.1.2 Buffer类的重要属性：

**private** **float** fFrameLost ; // 帧丢失的概率

**private** **boolean** bNewState = **false** ; // 锁， 当物理层传入的数据，//数据链路层未取走时 物理层不能再次传入数据

**private** **boolean** bFrameArrived = **false** ;

**private** **boolean** bTimeOut = **false** ;

**private** **boolean** bCksumErr = **false** ;

**private** **boolean** bNetworkLayerReady = **false** ;

**private** **byte**[] frame ; // 缓存物理层传入的帧

### 3.1.2 Buffer类的重要方法：

// 模拟传播时间，正常或超长的

**public** **void** tranmissionTime () {

**if** ( Math.*random* () < fFrameLost ) { //TimeOut

**try** {

Thread.*sleep* ( 10000 ) ;

} **catch** ( InterruptedException e ) {

e.printStackTrace () ;

}

} **else** {

**try** {

Thread.*sleep* ( 100 ) ;

} **catch** ( InterruptedException e ) {

e.printStackTrace () ;

}

}

}

## 3.2 DataLinkLayer

### 3.2.1 DataLinkLayer 类的重要属性：

**private** **int** iMAXSEQ ; // 窗口大小

**private** Buffer buf ; // 内存

**private** PhysicalLayer pl ; // 物理层

**private** Timer[] clock ; // 时钟

**private** DatagramSocket ds ;

**private** String myIP ;

**private** **int** myPort ;

**private** String deIP ;

**private** **int** dePort ;

**private** **int** iWindowSize ; // 窗口大小

**private** **float** fFrameSend ; // 有帧要发送的概率率 （每毫秒）

**private** **float** fFrameLost ; // 帧丢失的概率

**private** **float** fFrameError ; // 帧出错的概率

**private** **int** iNextFrameToSend ;

**private** **int** iFrameExpected ;

**private** **int** iAckExpected ;

**private** **byte**[] sFrame, rFrame ;

**private** **byte**[] rPacket ;

**private** **int** iBuf ;

**private** **byte**[][] sPacket = **null** ;

**private** **int** iBufPacket = 0 ;

**private** **boolean** networkLayerFlag ; // 标志位，为true时，允许网络层发送数据

### 3.2.1 DataLinkLayer 类的重要方法：

1 构造函数

**public** DataLinkLayer ( String sName, String myIP, **int** myPort, String deIP, **int** dePort,

**int** iWindowSize, **float** fFrameSend, **float** fFrameLost, **float** fFrameError,

JLabel jl0, JLabel jl1, JLabel jl2, JTextArea jta0, JTextArea jta1, JTextArea jta2,

**boolean** bFirst ) {

**this**.sName = sName ;

**this**.myIP = myIP ;

**this**.myPort = myPort ;

**this**.deIP = deIP ;

**this**.dePort = dePort ;

**this**.iWindowSize = iWindowSize ;

**this**.fFrameSend = fFrameSend ;

**this**.fFrameLost = fFrameLost ;

**this**.fFrameError = fFrameError ;

**this**.jl[0] = jl0 ;

**this**.jl[1] = jl1 ;

**this**.jl[2] = jl2 ;

**this**.jta[0] = jta0 ;

**this**.jta[1] = jta1 ;

**this**.jta[2] = jta2 ;

**this**.bFirst = bFirst ;

**this**.iMAXSEQ = iWindowSize ;

buf = **new** Buffer ( "Buffer ", fFrameLost ) ;

**try** {

ds = **new** DatagramSocket ( myPort ) ;

} **catch** ( SocketException e ) {

e.printStackTrace () ;

}

pl = **new** PhysicalLayer ( "physical layer ", buf, ds ) ;

sPacket = **new** **byte**[iMAXSEQ + 1][] ;

clock = **new** Timer[iMAXSEQ + 1] ;

**for** ( **int** i = 0; i <= iMAXSEQ; i++ ) {

String sTemp = "clock" + i ;

clock[i] = **new** Timer ( sTemp, buf ) ;

clock[i].start () ;

}

**if** ( bFirst ) {

fileName = "A.txt" ;

} **else** {

fileName = "B.txt" ;

}

file = **new** File ( fileName ) ;

**try** {

in = **new** FileInputStream ( file ) ;

} **catch** ( FileNotFoundException e ) {

e.printStackTrace () ;

}

// 等待 一段时间，

**try** {

Thread.*sleep* ( 10000 ) ;

} **catch** ( InterruptedException e ) {

e.printStackTrace () ;

}

}

2 run函数

**public** **void** run () {

initState () ;

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

dealEvent () ;

}

}

3 initState 函数

**public** **void** initState () {

pl.start () ;

iAckExpected = 0 ;

iNextFrameToSend = 0 ;

iFrameExpected = 0 ;

iBuf = 0 ;

**this**.enableNetworkLayer () ;

}

4 dealEvent 函数

**public** **void** dealEvent () {

**if** ( **this**.buf.isNewState ()) {

**if** ( **this**.buf.isTimeOut ()) {

dealWithTimeOut () ;

} **else** **if** ( **this**.buf.isCksumErr ()) {

dealWithCksumErr () ;

}**else** **if** ( **this**.buf.isFrameArrived ()) {

dealWithFrameArrived () ;

}

**this**.buf.setNewState ( **false** ) ;

}

// 模仿网络层随机准备的过程

**if** ( Math.*random* () < fFrameSend && **this**.networkLayerFlag ) {

**this**.buf.setNetworkLayerReady ( **true** ) ;

} **else** {

**this**.buf.setNetworkLayerReady ( **false** ) ;

}

**if** ( **this**.buf.isNetworkLayerReady ()) {

dealWithNetworkLayerReady () ;

}

**if** ( iBuf < iMAXSEQ ) {

**this**.enableNetworkLayer () ;

} **else** {

**this**.disableNetworkLayer () ;

}

5 具体的事件处理函数

**public** **void** dealWithTimeOut () {

// 输出

toPrint ( " timeOut ", 1 ) ;

iNextFrameToSend = iAckExpected ;

**this**.jl[0].setText ( "" + iNextFrameToSend ) ;

**this**.jl[2].setText ( "From " + iAckExpected + " to " + iNextFrameToSend ) ;

**for** ( **int** i = 0; i < iBuf; i++ ) {

sendFrame ( iNextFrameToSend, iFrameExpected, sPacket ) ;

iNextFrameToSend = inc ( iNextFrameToSend ) ;

// 输出

**this**.jl[0].setText ( "" + iNextFrameToSend ) ;

**this**.jl[2].setText ( "From " + iNextFrameToSend + " to " + iAckExpected ) ;

}

**this**.buf.setTimeOut ( **false** ) ;

}

**public** **void** dealWithFrameArrived () {

rFrame = fromPhysicalLayer () ;

**int** rSeq = Byte.*valueOf* ( rFrame[0] ).intValue () ;

**int** rAck = Byte.*valueOf* ( rFrame[1] ).intValue () ;

rPacket = **new** **byte**[rFrame.length - 4] ;

**for** ( **int** i = 3; i < rFrame.length - 1; i++ )

rPacket[i - 3] = rFrame[i] ;

**int** num = Byte.*valueOf* ( rFrame[ rFrame.length - 1] ).intValue () ;

// 输出

toPrint ( "receive fame" + num + " with seq : " + rSeq + " and ack : " + rAck, 1 ) ;

**if** ( rSeq == iFrameExpected ) {

toNetworkLayer ( rPacket ) ;

iFrameExpected = inc ( iFrameExpected ) ;

// 输出

**this**.jl[1].setText ( "" + iFrameExpected ) ;

}

**while** ( **this**.between( iAckExpected, rAck, iNextFrameToSend )) {

iBuf-- ;

clock[iAckExpected].pause () ;

iAckExpected = inc ( iAckExpected ) ;

// 输出

**this**.jl[2].setText ( "From " + iNextFrameToSend + " to " + iAckExpected ) ;

}

}

**public** **void** dealWithNetworkLayerReady () {

sPacket[iNextFrameToSend] = fromNetworkLayer () ;

iBuf++ ;

sendFrame ( iNextFrameToSend, iFrameExpected, sPacket ) ;

iNextFrameToSend = inc ( iNextFrameToSend ) ;

// 输出

**this**.jl[0].setText ( "" + iNextFrameToSend ) ;

**this**.jl[2].setText ( "From " + iNextFrameToSend + " to " + iAckExpected ) ;

**this**.buf.setNetworkLayerReady ( **false** ) ;

}

**public** **void** dealWithCksumErr () {

toPrint ( " cksumErr ", 1 ) ;

**this**.buf.setCksumErr ( **false** ) ;

}

6辅助函数

**public** **int** inc ( **int** a ) {

**if** ( a < iMAXSEQ ) {

**return** a + 1 ;

} **else** {

**return** 0 ;

}

}

**public** **boolean** between ( **int** a, **int** b, **int** c ) {

// 在窗口内？

**if** ((( a <= b ) && ( b < c )) || (( c < a ) && ( a <= b)) || (( b < c) && ( c < a ))) {

**return** **true** ;

} **else** {

**return** **false** ;

}

}

**public** **void** sendFrame ( **int** seq, **int** frameExpected, **byte**[][] packet ) {

**byte**[] sFrame = combineFrame ( seq, ( frameExpected + iMAXSEQ ) % ( iMAXSEQ + 1 ), iMAXSEQ - iBufPacket, packet[seq] ) ;

toPrint ( "send frame" + ( count - 1 ) + " with seq : " + seq + " and ack : " + ( frameExpected + iMAXSEQ ) % ( iMAXSEQ + 1 ), 2 ) ;

toPhysicalLayer ( sFrame ) ;

clock[seq].begin () ;

}

**public** **byte**[] combineFrame ( **int** seq, **int** ack, **int** iExcessWindow, **byte**[] packet ) {

**byte**[] frame = **new** **byte**[packet.length + 4] ;

frame[0] = Integer.*valueOf* ( seq ).byteValue () ;

frame[1] = Integer.*valueOf* ( ack ).byteValue () ;

frame[2] = occurError () ;

**for** ( **int** i = 0; i < packet.length; i++ )

frame[i + 3] = packet[i] ;

frame[frame.length - 1] = Integer.*valueOf* ( count++ ).byteValue () ;

**return** frame ;

}

**public** **void** disableNetworkLayer () {

**this**.networkLayerFlag = **false** ;

}

**public** **void** enableNetworkLayer () {

**this**.networkLayerFlag = **true** ;

}

**public** **byte** occurError () {

**byte** ck ;

**if** ( Math.*random* () < fFrameError ) { //FrameError

ck = Integer.*valueOf* ( 0 ).byteValue () ;

} **else** {

ck = Integer.*valueOf* ( 1 ).byteValue () ;

}

**return** ck ;

}

7 与网络层的接口

**public** **void** toNetworkLayer ( **byte**[] packet ) {

jta[0].append ( **new** String ( packet, 0, packet.length )) ;

}

**public** **byte**[] fromNetworkLayer () {

**byte**[] packet ;

**byte**[] btTemp = **new** **byte**[100] ;

**int** len = 0 ;

**try** {

len = in.read ( btTemp ) ;

} **catch** ( IOException e ) {

e.printStackTrace () ;

}

packet = **new** **byte**[len] ;

**for** ( **int** i = 0; i < len ; i++ )

packet[i] = btTemp[i] ;

**return** packet ;

}

8 与物理层的接口

**public** **void** toPhysicalLayer ( **byte**[] frame ) {

String[] ipStr = deIP.split ( "\\." ) ;

**byte**[] ipBuf = **new** **byte**[4] ;

**for** ( **int** i = 0; i < 4; i++ )

ipBuf[i] = (**byte**) ( Integer.*parseInt* ( ipStr[i]) & 0xff ) ;

DatagramPacket dp = **null** ;

**try** {

dp = **new** DatagramPacket ( frame, frame.length,

InetAddress.*getByAddress* ( ipBuf ), dePort ) ;

} **catch** ( UnknownHostException e ) {

e.printStackTrace () ;

}

**try** {

ds.send ( dp ) ;

} **catch** ( IOException e ) {

e.printStackTrace () ;

}

}

**public** **byte**[] fromPhysicalLayer () {

**byte**[] frame = **null** ;

frame = **this**.buf.getFrame () ;

**return** frame ;

}

## 3.3 PhysicalLayer

### 3.3.1 PhysicalLayer 类重要属性：

**private** Buffer buf ;

**private** DatagramSocket ds ;

### 3.3.2 PhysicalLayer 类重要方法：

1 run 函数

**public** **void** run () {

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

receiveFrame () ;

}

}

2 关键函数

**public** **void** receiveFrame () {

**if** ( ! **this**.buf.isNewState ()) {

fun () ;

}

}

**public** **void** fun () {

DatagramPacket dp = **null** ;

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

dp = **new** DatagramPacket ( btTemp, btTemp.length ) ;

**try** {

ds.receive ( dp ) ;

} **catch** ( IOException e ) {

e.printStackTrace () ;

}

**byte**[] frame = **new** **byte**[ dp.getLength ()] ;

**for** ( **int** i = 0; i < dp.getLength (); i++ )

frame[i] = dp.getData () [i] ;

**this**.buf.setFrame ( frame ) ;

**this**.buf.setFrameArrived ( **true** ) ;

**this**.buf.setNewState ( **true** ) ;

}

## 3.4 Timer

### 3.4.1 Timer 类重要属性：

**private** Buffer buf ;

**private** **long** lStartTime ; // 记录开始时间

**private** **boolean** bFlag = **false** ; // 标志位， 当为true是，计时器正常运转

// 实现时期的pause功能，减少所需计时器的数量

**private** **static** **long** *TIMEOUT* = 10000l ;

### 3.4.2 Timer 类重要方法：

1 run函数

**public** **void** run () {

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

**while** ( bFlag ) {

**long** lCurrentTime = **new** Date ().getTime () ;

**if** ( lCurrentTime - lStartTime > *TIMEOUT* ) {

**this**.buf.setNewState ( **true** ) ;

**this**.buf.setTimeOut ( **true** ) ;

**this**.bFlag = **false** ;

}

}

**try** {

**this**.join ( *TIMEOUT* / 10 ) ;

} **catch** ( InterruptedException e ) {

e.printStackTrace () ;

}

}

}

2 begin函数和pause函数

**public** **void** begin () {

**this**.bFlag = **true** ;

lStartTime = **new** Date ().getTime () ;

}

**public** **void** pause () {

**this**.bFlag = **false** ;

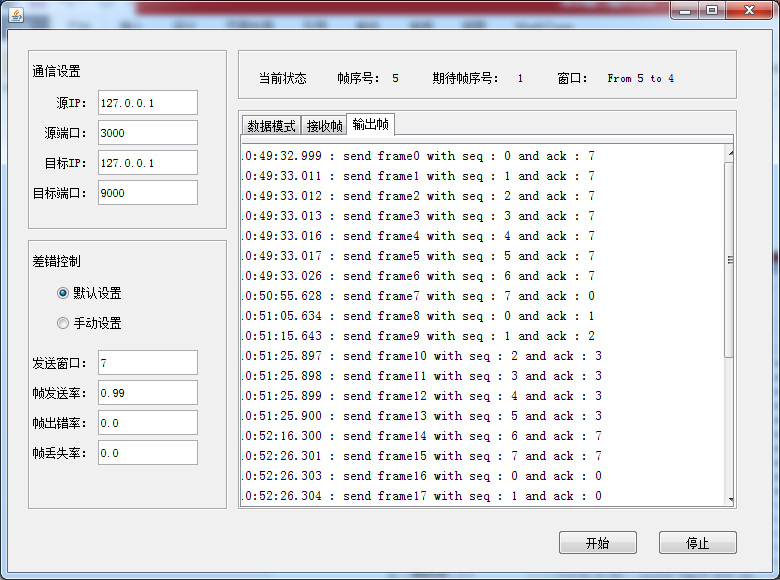
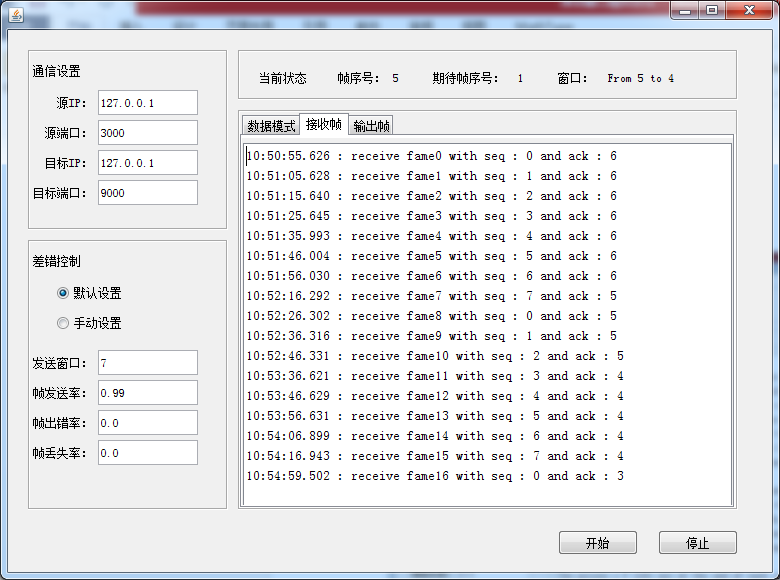
}

# 4 系统测试运行

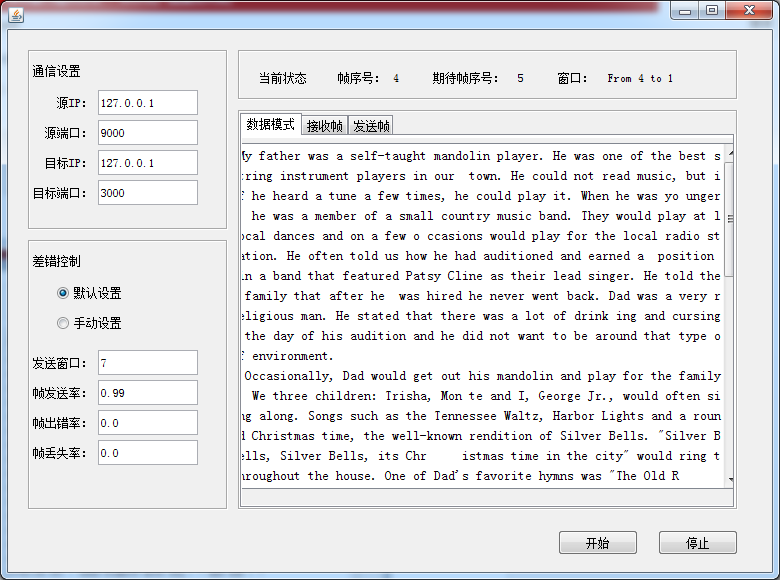
## 4.1 无丢失，无损坏情况下的Go-back-n滑动窗口协议测试

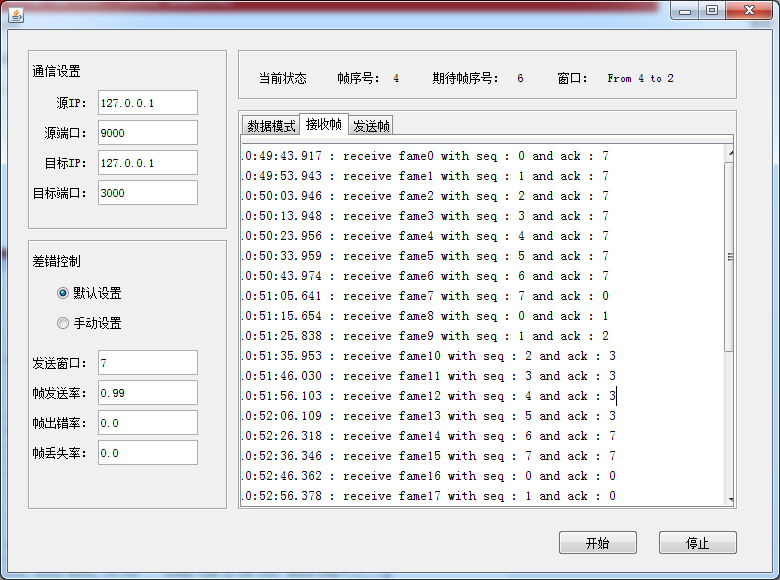
### 4.1.1 双方正常发送数据：正常

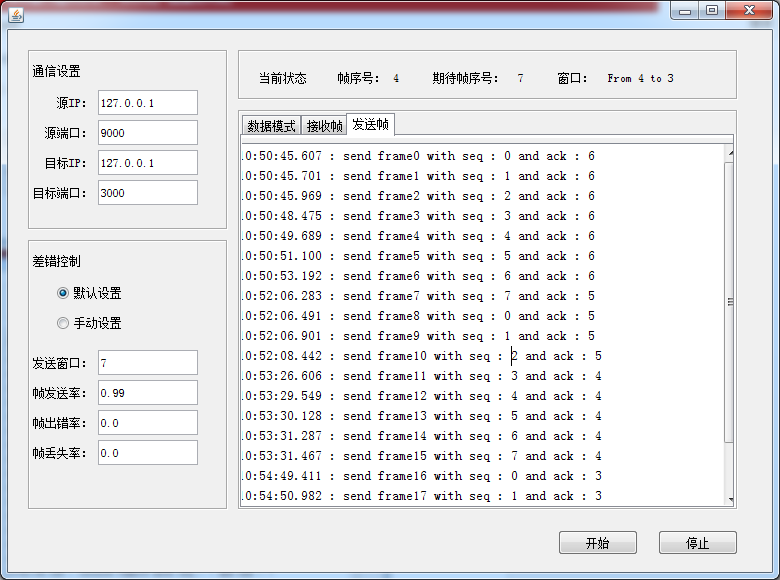
Host A



Host B

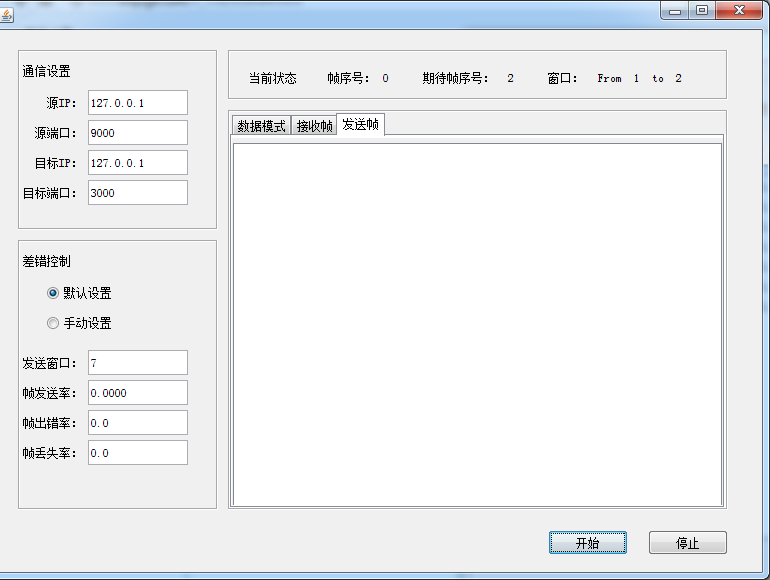
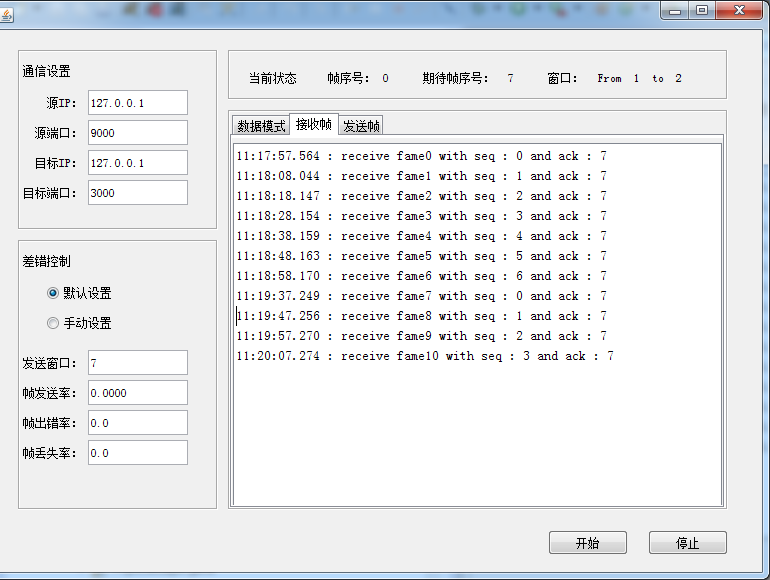
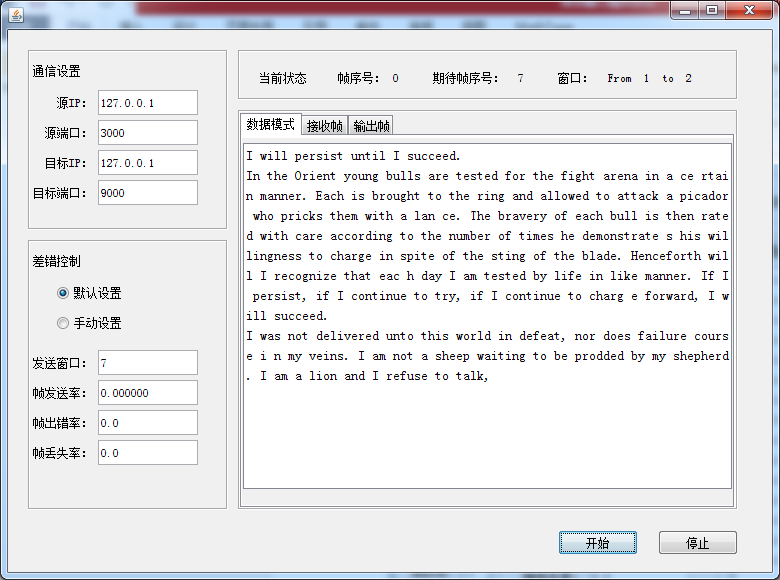




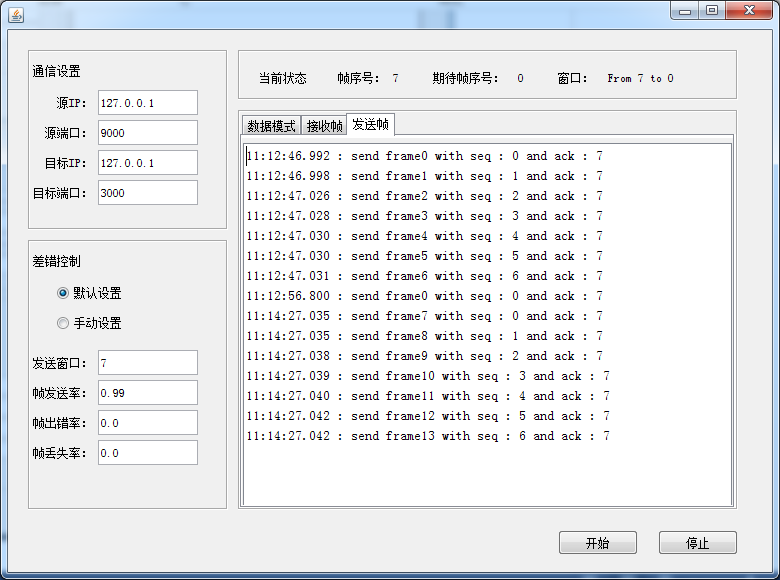
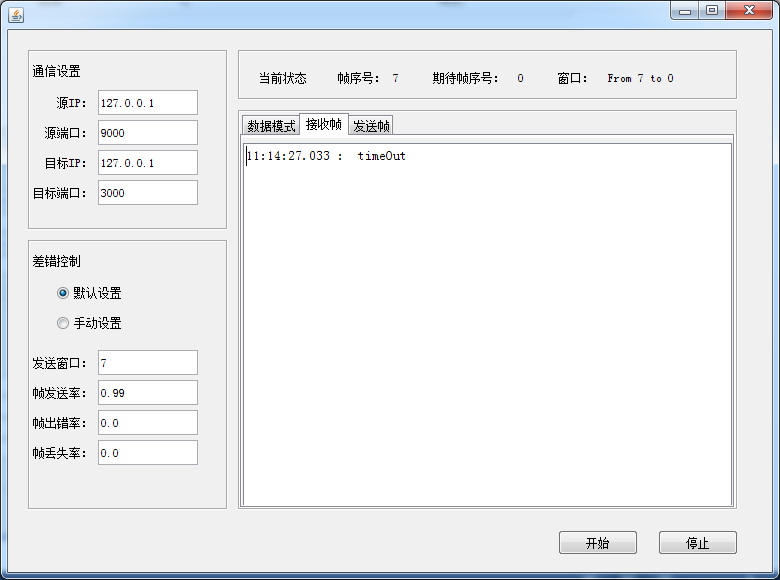
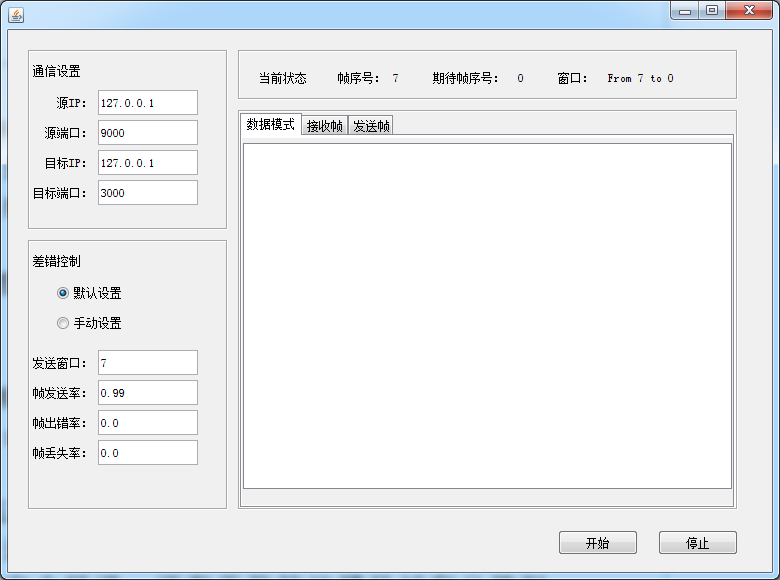


### 4.1.2 一方不发送数据：正常

Host A （不发送数据） ：



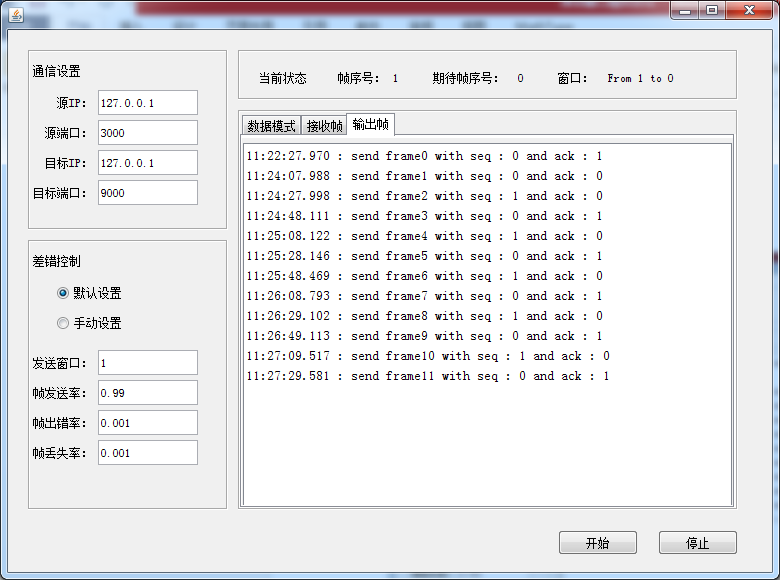
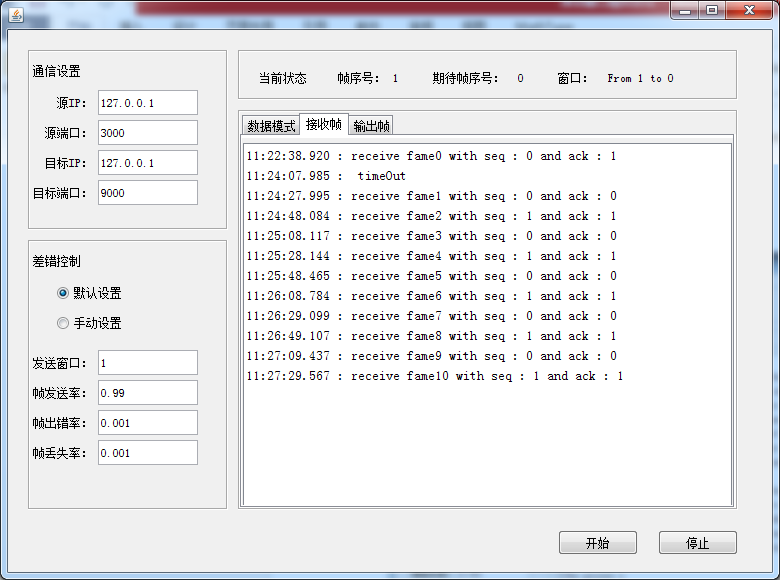
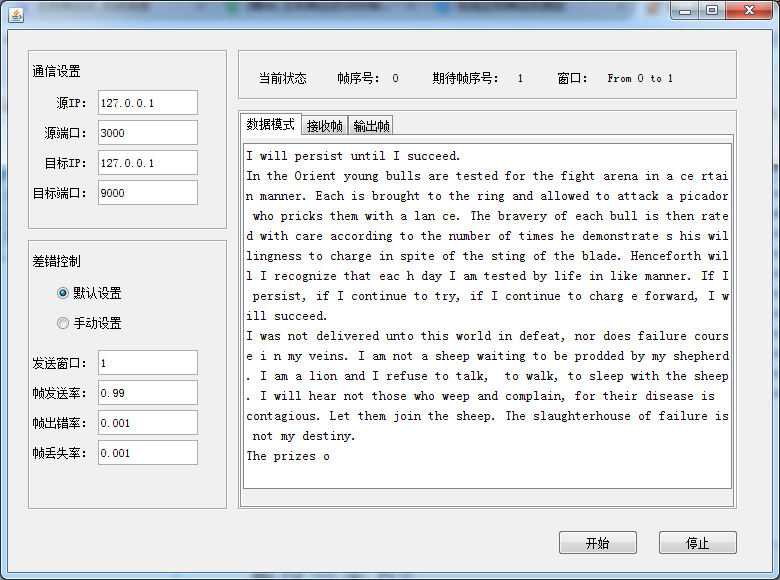
Host B （发送数据） ：



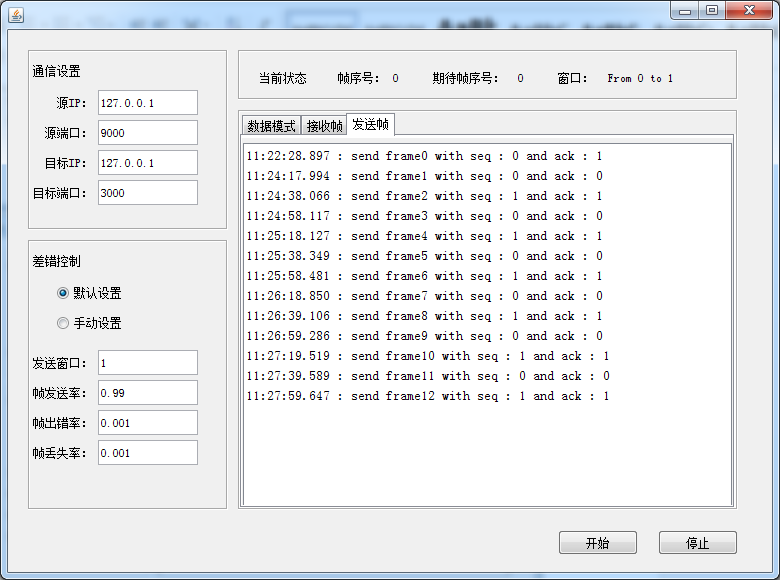
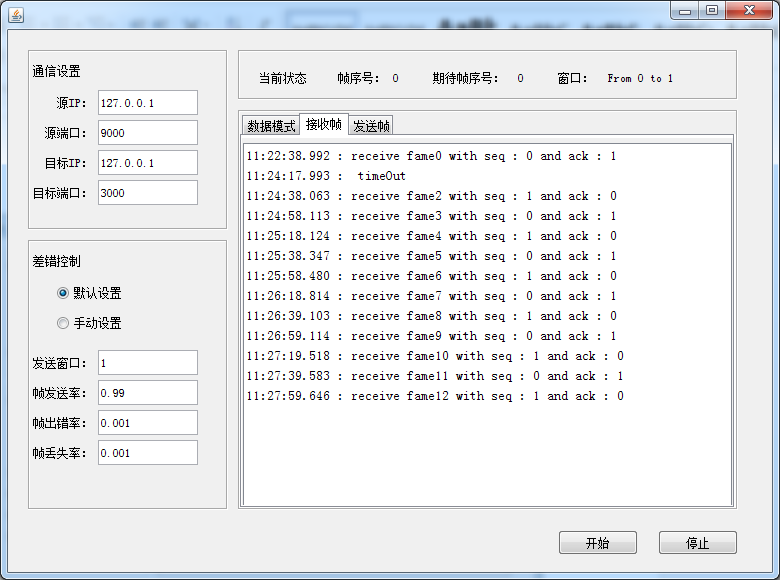
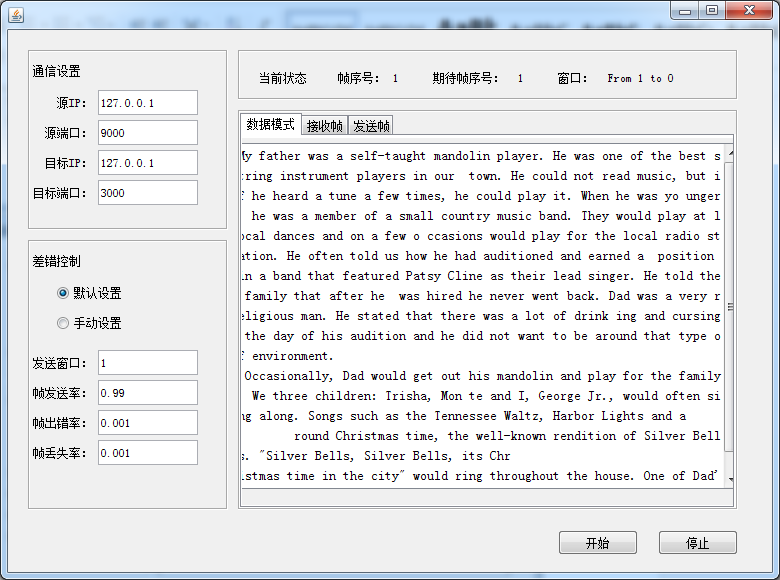
## 4.2 有丢失，损坏情况下的Go-back-n滑动窗口协议

### 4.2.1 窗口大小都1：正常

Host A ：

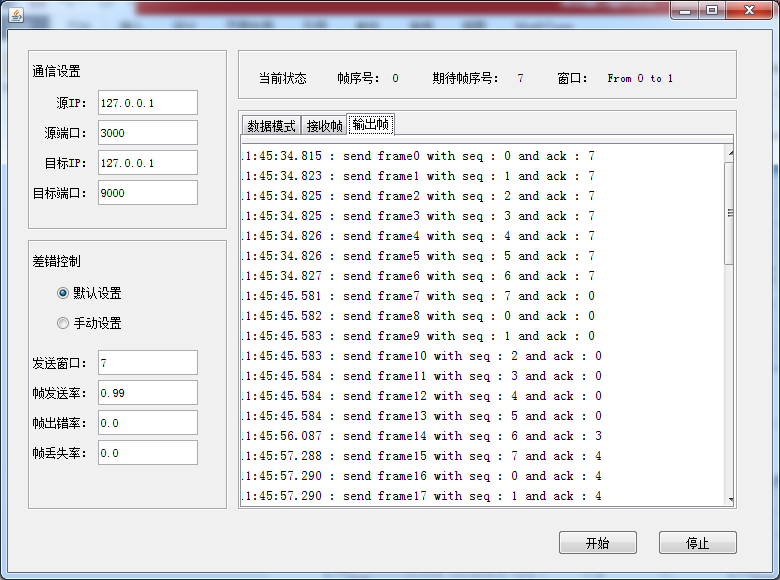
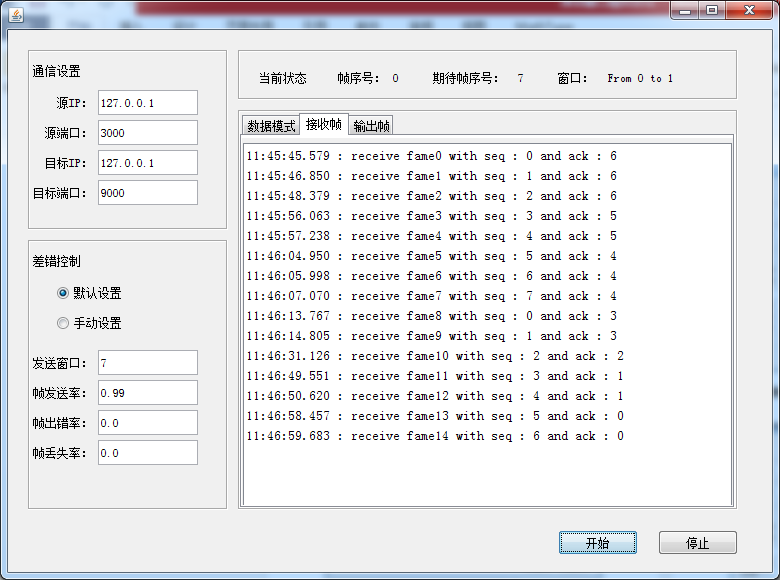


Host B ：

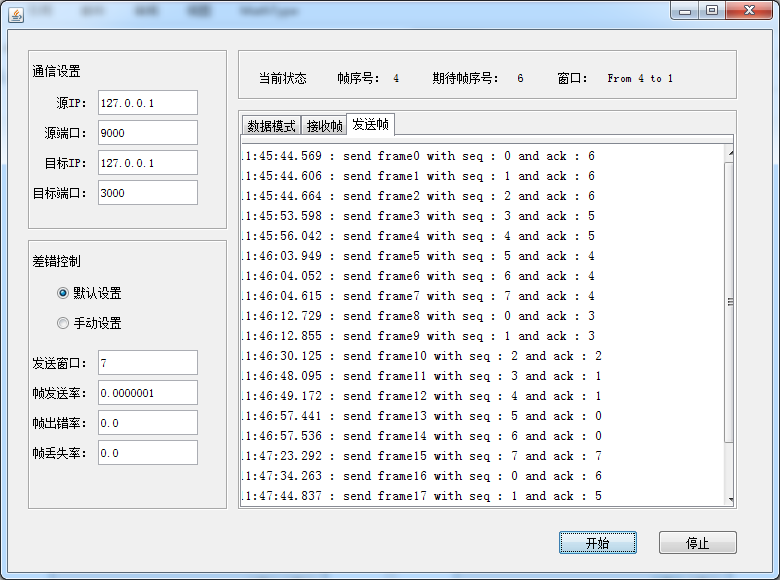
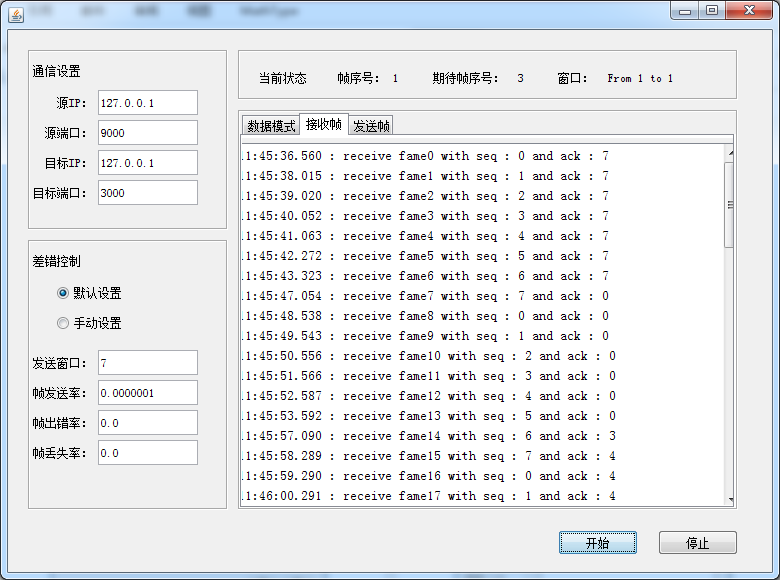
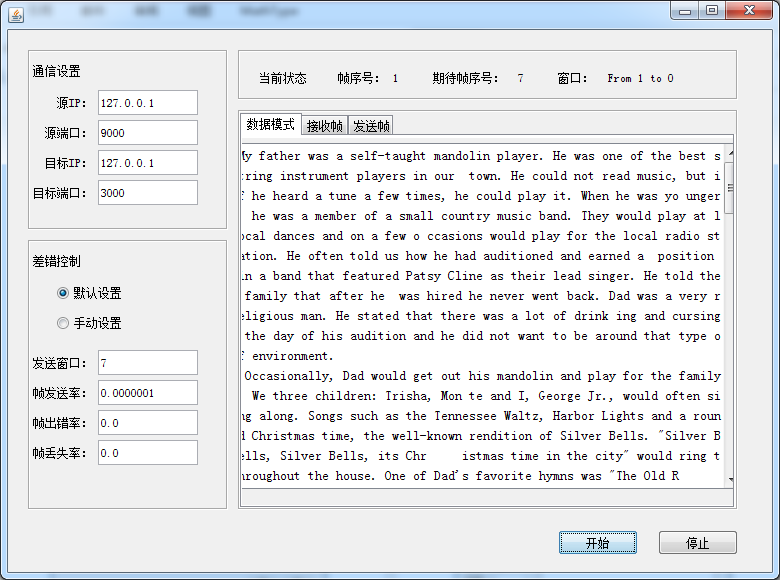


### 4.2.2 双发都发数据：正常

Host A ：

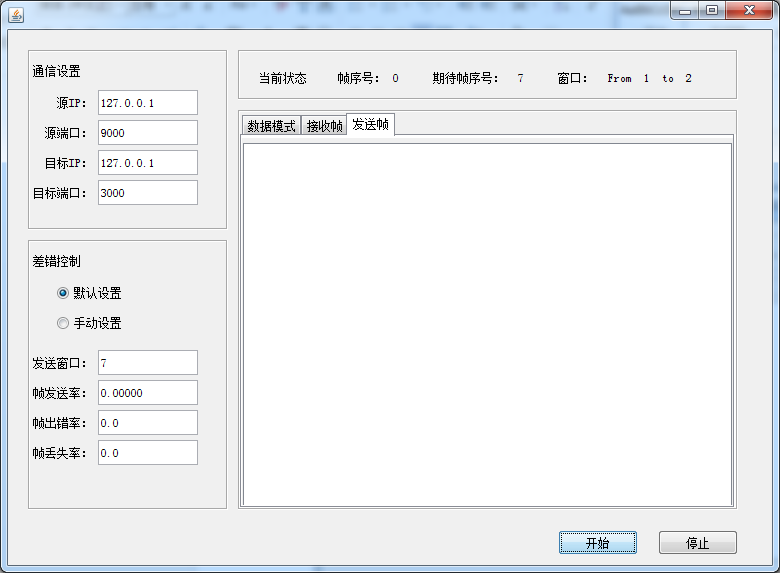
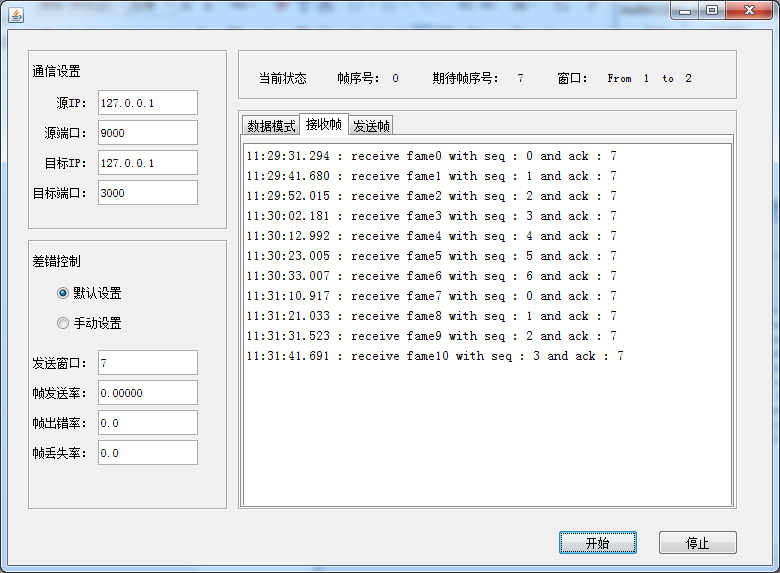
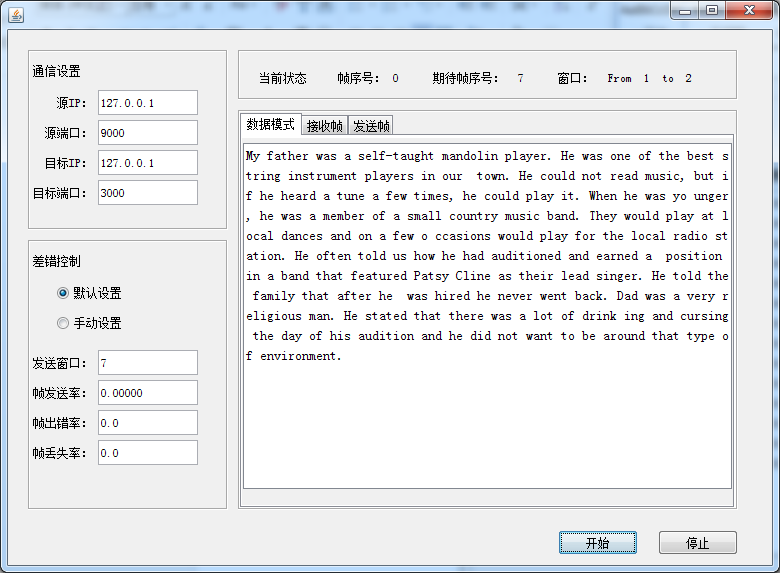


Host B ：

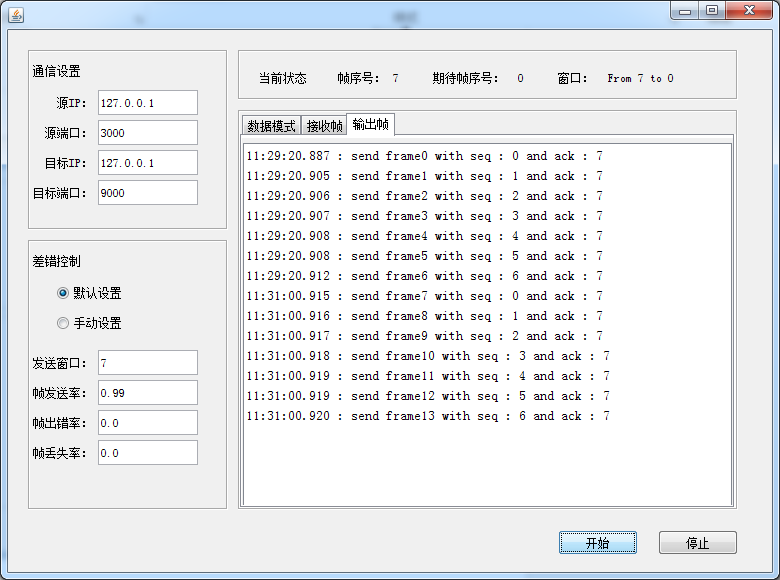
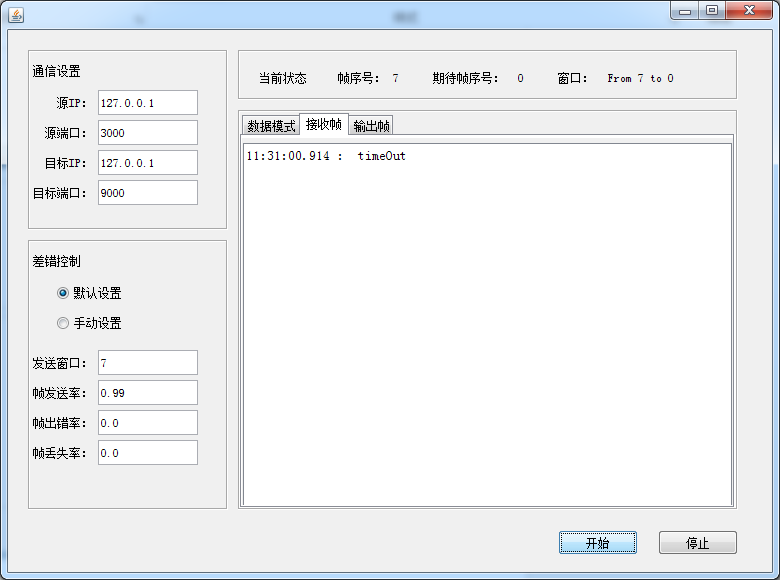
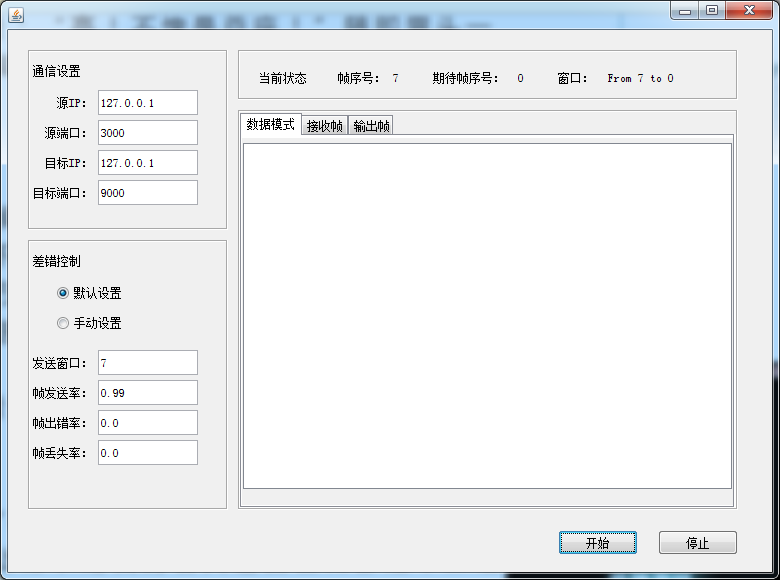


### 4.2.3 只有一方发数据：正常

Host A （不发数据） ：



Host B （发数据） ：



# 5 总结

* 在Project中详细的了解了Java线程的工作机制和原理，学习并掌握了了多线程编程。
* 熟悉了类的设计，但是类的功能的划分中仍然存在一定的缺陷