**计算机网络编程**

**实验报告**

**班级：07111707**

**组长：1120171189 崔程远**

**成员：1120172149 吴沁璇**

**1120172153 张澈**

**1120172163 王晓媛**

**1120172733 张鉴昊**

**1120172765 曾煜瑾**

**1120173326 曾紫飞**

**北京理工大学**

**计算机学院**

**2020年6月**

**第三章 实验4 基于连续ARQ协议的可靠通信**

**1. 实验目的**

模拟数据链路层双工传输数据的后退N帧协议，加深对gbn协议的理解和掌握。

1. **实验内容**

采用UDP Socket编程接口作为模拟物理层接口实现帧的发送和接收，协议采用双工方式进行数据通信。假设Host1和Host2分别向对方发送大文件，Host1先发送一帧到Host2，通过数据链路层的帧每次完成数据块的可靠传输，采用GBN协议，差错编码采用CRC-CCITT标准。以教材协议5为基础，在帧末尾增加CRC校验字段。

配置文件关键要点：

数据传输目的UDP端口

UDPPort=8888

增添发送过滤程序，模拟传输出错或丢数据帧，下面两项指明每发送多少帧出现一次出错或丢帧，此例表示每10帧中一帧出错，每10帧中一帧丢失

FilterError=10

FilterLost=10

Host1程序运行屏幕输出关键要点：

显示ack\_expected, next\_frame\_to\_send和frame\_expected变量的值，以及正在发送帧的编号和确认序号

显示经过过滤器后是正确发送、模拟传输出错还是模拟帧丢失（实际没有发送）

显示接收到对方帧，该帧的发送序号和确认序号，以及当前frame\_expected变量的值

或者显示超时，重传帧的发送序号，以及ack\_expected, next\_frame\_to\_send变量的值

回到开始重复一直到文件发送完成

Host2程序运行屏幕输出关键要点：

显示frame\_expected变量的值

接收帧是否出错（CRC余数是否为零），正确则显示接收帧的发送帧序号

显示ack\_expected, next\_frame\_to\_send和frame\_expected变量的值，以及正在发送帧的编号和确认序号

显示经过过滤器后是正确发送、模拟传输出错还是模拟帧丢失（实际没有发送）

或者显示超时，重传帧的发送序号，以及ack\_expected, next\_frame\_to\_send变量的值

回到开始重复一直到文件接收完成

1. **实验原理**

在后退N帧协议中，发送方无须在收到上一个帧的ACK后才能开始发送下一帧，而是可以连续发送帧。当接收方检测出失序的信息帧后，要求发送方重发最后一个正确接收的信息帧之后的所有未被确认的帧；或者当发送方发送了N个帧后，若发现该N个帧的前一个帧在计时器超时后仍未返回其确认信息，则该帧被判为出错或丢失，此时发送方就不得不重传该出错帧及随后的N个帧。换句话说，接收方只允许按顺序接收帧。

源站向目的站发送数据帧，源站每发送完一帧就要为该帧设置超时计时器。由于连续发送了许多帧，所以确认帧必须要指明是对哪一帧进行确认。为了减少开销，GBN协议还规定个正确的数据帧后，才对最后一个数据帧发确认信息，或者可在自己有数据要发送时才将对以前正确收到的帧加以捎带确认。这就是说，对某一数据帧的确认就表明该数据帧和此前所有的数据帧均已正确无误地收到。此外，当接收端收到了不按序的无差错帧，但应重复发送已发送的最后一个确认帧。

1. **实验环境**

|  |  |  |
| --- | --- | --- |
| 语言 | 集成开发环境 | 编译器 |
| C++ | Visual Studio 2017 | gcc version 4.8.1 |
| Java | Eclipse 2019 | java version "1.8.0\_65" |
| Python | Pycharm 2017 | Python 3.7.0 |

1. **实验步骤**

本实验C代码由一位同学负责，Java和Python由我负责，我打算双工传输文件，Java成功实现了，然后Python的多线程出了很多问题，写了代码后程序不能正常运行，碍于已经在本实验花了太多太多时间，还有其他的课程考试要复习，所以python无法再花更多时间去调试了，这个实验真的很难。

我对此实验的做法是：分两个线程，一个专门负责发，另一个专门负责收，收的线程每收到一帧就进行捎带确认，并填充缓冲区，然后进行一系列的判断，更改变量。如果超时没收到则表示已发送的帧出错，需要发的线程重发。发的线程根据变量选择重发或者正常发送。然而这个实验更复杂的是对文件头、文件尾的判断，很多种情况：一端发送完数据，另一端还在发，接收帧是否包含了数据、是否有确认等。对于本实验而言，两端的代码基本一致，下面以UDPHost1.java代码为例进行分析。

**·变量定义**

**private** **static** **int** *host1\_Port* = 9999;

**private** **static** **int** *host2\_Port* = 8888;

**private** **static** DatagramSocket *ds*;

**private** **static** InetAddress *address*;

**private** **static** InputStream *is*;

**private** **static** OutputStream *os*;

**private** **static** **int** *infoLen* = 20;

**private** **int** frameLen = 28;

**private** **int** kindPos = 0;

**private** **byte** hasData = 1;

**private** **byte** notHasData = 0;

**private** **int** seqPos = 1;

**private** **int** ackPos = 2;

**private** **byte** notHasAck = 127;

**private** **int** infoFlagPos = 3;

**private** **byte** lastInfoFrame = 0;

**private** **byte** notLastInfoFrame = 1;

**private** **int** infoStartPos = 4;

**private** **int** crcStartPos = 24;

**private** **int** crcLastPos = 25;

**private** **static** **int** *MAX\_SEQ* = 7;

**private** **static** **byte** *buffer*[][] = **new** **byte**[*MAX\_SEQ* + 1][*infoLen* + 2];

**private** **static** **int** *nbuffered* = 0;

**private** **static** **int** *number*[] = **new** **int**[*MAX\_SEQ* + 1];

**private** **static** **int** *nextFrameToSend* = 0;

**private** **static** **int** *ackExpected* = 0;

**private** **static** **int** *frameExpected* = 0;

**private** **static** **int** *rank* = 0;

**private** **int** filterRank = 0;

**private** **int** filterError = 10;

**private** **int** filterLost = 10;

**private** **int** firstError = 3;

**private** **int** firstLost = 8;

**final** **private** **int** right = 0;

**final** **private** **int** error = 1;

**final** **private** **int** lost = 2;

**private** **static** **boolean** *needToResend* = **false**;

**private** **static** **boolean** *waitForResend* = **false**;

**private** **static** **boolean** *receiveInfoFinished* = **false**;

**private** **static** **boolean** *lastSendAck* = **false**;

**private** **static** **boolean** *receiveFirstInfo* = **false**;

**private** **static** **int** *readFileFlag* = 0;

**private** **static** **int** *overtimeFlag* = 0;

**·辅助函数**

**public** **int** Inc(**int** k) {

**if** (k < *MAX\_SEQ*) {

**return** k + 1;

} **else** {

**return** 0;

}

}

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

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

**return** **true**;

} **else** {

**return** **false**;

}

}

**public** String getBinaryString(**byte** b[]) {

String str = "";

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

str += Integer.*toBinaryString*((b[i] & 0xFF) + 0x100).substring(1);

}

**return** str;

}

**public** String getRemainderString(String dividendStr, String divisorStr) {

**int** dividendLen = dividendStr.length();

**int** divisorLen = divisorStr.length();

**for** (**int** i = 0; i < divisorLen - 1; i++) {

dividendStr += "0";

}

**char** str1[] = dividendStr.toCharArray();

**char** str2[] = divisorStr.toCharArray();

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

**if** (str1[i] == '1') {

str1[i] = '0';

**for** (**int** j = 1; j < divisorLen; j++) {

**if** (str1[i + j] == str2[j]) {

str1[i + j] = '0';

} **else** {

str1[i + j] = '1';

}

}

}

}

String remainderStr = Arrays.*toString*(str1).replaceAll(", ", "").substring(dividendLen + 1,

dividendLen + divisorLen);

**return** remainderStr;

}

**public** String getCRCString(**byte** info[], **int** index) {

**byte** realInfo[] = **new** **byte**[index];

System.*arraycopy*(info, 0, realInfo, 0, index);

String binaryStr = getBinaryString(realInfo);

String gxStr = "10001000000100001";

**return** getRemainderString(binaryStr, gxStr);

}

**public** **boolean** isInfoRight(**byte** frame[]) {

**int** index = infoStartPos;

**for** (; index < infoStartPos + *infoLen*; index++) {

**if** (frame[index] == 0) {

**break**;

}

}

**byte** info[] = **new** **byte**[index - infoStartPos + 2];

System.*arraycopy*(frame, infoStartPos, info, 0, index - infoStartPos);

System.*arraycopy*(frame, crcStartPos, info, index - infoStartPos, 2);

String crcStr = getCRCString(info, info.length);

**if** (Integer.*parseInt*(crcStr, 2) == 0) {

**return** **true**;

} **else** {

**return** **false**;

}

}

**public** **boolean** fetchNewInfo() **throws** Exception {

**byte** info[] = **new** **byte**[*infoLen*];

**if** (*readFileFlag* != -1) {

*readFileFlag* = *is*.read(info);

**if** (*readFileFlag* == -1) {

**return** **false**;

}

**for** (**int** i = *readFileFlag*; i < *infoLen*; i++) {

info[i] = 0;

}

String crcStr = getCRCString(info, *readFileFlag*);

System.*arraycopy*(info, 0, *buffer*[*ackExpected*], 0, *infoLen*);

*buffer*[*ackExpected*][*infoLen*] = (**byte**) Integer.*parseInt*(crcStr.substring(0, 8), 2);

*buffer*[*ackExpected*][*infoLen* + 1] = (**byte**) Integer.*parseInt*(crcStr.substring(8, 16), 2);

*number*[*ackExpected*] = *rank*;

*rank*++;

**return** **true**;

}

**return** **false**;

}

**·接收线程**

//接收线程

@Override

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

// **TODO** Auto-generated method stub

**byte** info[] = **new** **byte**[*infoLen*];

//加载窗口大小的数据到发送缓存

**for** (*rank* = 0; *rank* < *MAX\_SEQ* + 1; *rank*++) {

**try** {

*readFileFlag* = *is*.read(info);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**if** (*readFileFlag* == -1) {

**break**;

}

**for** (**int** i = *readFileFlag*; i < *infoLen*; i++) {

info[i] = 0;

}

String crcStr = getCRCString(info, *readFileFlag*);

System.*arraycopy*(info, 0, *buffer*[*rank*], 0, *infoLen*);

*buffer*[*rank*][*infoLen*] = (**byte**) Integer.*parseInt*(crcStr.substring(0, 8), 2);

*buffer*[*rank*][*infoLen* + 1] = (**byte**) Integer.*parseInt*(crcStr.substring(8, 16), 2);

*number*[*rank*] = *rank*;

}

*nbuffered* = *rank*;

**byte** receFrame[] = **new** **byte**[frameLen];

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

**try** {

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

*ds*.receive(dp);

**if** (receFrame[kindPos] == notHasData) {//没有数据，只有确认，对方数据已经接收完毕

SimpleDateFormat df = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

System.***out***.println("Current time: " + df.format(**new** Date()));

System.***out***.println("Receive a frame, only has ack, the ack is: " + receFrame[ackPos]);

System.***out***.println();

//捎带确认加载发送数据到缓存

**while** (Between(*ackExpected*, receFrame[ackPos], *nextFrameToSend*)) {

*nbuffered* = *nbuffered* - 1;

**try** {

**if** (fetchNewInfo() == **true**) {

*nbuffered* = *nbuffered* + 1;

}

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

*ackExpected* = Inc(*ackExpected*);

**if** (*nbuffered* == 0) {

System.***out***.println("SendFile was sended completed.");

System.***out***.println();

}

}

**if** (*nbuffered* == 0 && *receiveInfoFinished* == **true**) {//发送完毕，接收完毕，退出接收线程

**break**;

}

} **else** **if** (receFrame[kindPos] == hasData) {//有数据，可能有确认，也可能没确认

**if** (receFrame[ackPos] != notHasAck) {//有确认，捎带确认加载发送数据到缓存

**while** (Between(*ackExpected*, receFrame[ackPos], *nextFrameToSend*)) {

*nbuffered* = *nbuffered* - 1;

**try** {

**if** (fetchNewInfo() == **true**) {

*nbuffered* = *nbuffered* + 1;

}

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

*ackExpected* = Inc(*ackExpected*);

**if** (*nbuffered* == 0) {

System.***out***.println("SendFile was sended completed.");

System.***out***.println();

}

}

}

//接收帧序号正确且数据正确

**if** (receFrame[seqPos] == *frameExpected* && isInfoRight(receFrame) == **true**) {

*os*.write(receFrame, infoStartPos, *infoLen*);

SimpleDateFormat df = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

System.***out***.println("Current time: " + df.format(**new** Date()));

**if** (receFrame[ackPos] == notHasAck) {//没有确认

System.***out***.println(

"Receive a frame, both seq and info are right, doesn't has ack, seq | frame\_expected are: "

+ receFrame[seqPos] + " | " + *frameExpected*);

} **else** {//有确认

System.***out***.println(

"Receive a frame, both seq and info are right, seq | ack | frame\_expected are: "

+ receFrame[seqPos] + " | " + receFrame[ackPos] + " | " + *frameExpected*);

}

System.***out***.println();

*frameExpected* = Inc(*frameExpected*);

*waitForResend* = **false**;

*receiveFirstInfo* = **true**;

**if** (receFrame[infoFlagPos] == lastInfoFrame) {//收到了最后一个数据帧

System.***out***.println(111111111);

System.***out***.println("ReceiveFile was received completed.");

System.***out***.println();

*receiveInfoFinished* = **true**;//表明接收完毕

**if** (*nbuffered* == 0) {//如果发送缓存为空，不需要再接收了，退出接收线程

**break**;

}

}

} **else** {//接收帧序号和数据不全正确

**if** (*waitForResend* == **true**) {

**continue**;

}

SimpleDateFormat df1 = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

System.***out***.println("Current time: " + df1.format(**new** Date()));

**if** (receFrame[seqPos] != *frameExpected* && isInfoRight(receFrame) != **true**) {

System.***out***.println("Receive a frame, both seq and info are wrong, wait for resend!");

} **else** **if** (receFrame[seqPos] != *frameExpected*) {

System.***out***.println("Receive a frame, seq is wrong, wait for resend!");

} **else** **if** (isInfoRight(receFrame) != **true**) {

System.***out***.println("Receive a frame, info is wrong, wait for resend!");

}

System.***out***.println();

*waitForResend* = **true**;

}

}

} **catch** (Exception e) {//接收超时，意味帧丢失

SimpleDateFormat df = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

System.***out***.println("Current time: " + df.format(**new** Date()));

System.***out***.println("Receiving is overtime, need to resend!");

System.***out***.println();

*overtimeFlag*++;

*needToResend* = **true**;

}

}

}

**·构造发送帧并发送的函数**

//构造发送帧并发送

**public** **void** fillAndSend(**int** method) **throws** Exception {

**byte** sendFrame[] = **new** **byte**[frameLen];

sendFrame[kindPos] = hasData;

sendFrame[seqPos] = (**byte**) *nextFrameToSend*;

**if** (*lastSendAck* == **false** && *receiveFirstInfo* == **true**) {

sendFrame[ackPos] = (**byte**) ((*frameExpected* + *MAX\_SEQ*) % (*MAX\_SEQ* + 1));

**if** (*receiveInfoFinished* == **true**) {

*lastSendAck* = **true**;

}

} **else** {

sendFrame[ackPos] = notHasAck;

}

**if** (*readFileFlag* == -1 && *nbuffered* == 1) {

sendFrame[infoFlagPos] = lastInfoFrame;

} **else** {

sendFrame[infoFlagPos] = notLastInfoFrame;

}

System.*arraycopy*(*buffer*[*nextFrameToSend*], 0, sendFrame, infoStartPos, *infoLen* + 2);

//System.out.println("nbuffered: " + nbuffered);

**if** (method == error) {//模拟传输出错

sendFrame[crcLastPos] = (**byte**) ((sendFrame[crcLastPos] + 1) % 128);

SimpleDateFormat df = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

System.***out***.println("Current time: " + df.format(**new** Date()));

System.***out***.println("ack\_expected | next\_frame\_to\_send | frame\_expected are: " + *ackExpected* + " | "

+ *nextFrameToSend* + " | " + *frameExpected*);

**if** (sendFrame[ackPos] != notHasAck) {

System.***out***.println("Stimulate sending wrongly, the number and ack of the frame are: "

+ *number*[*nextFrameToSend*] + " | " + sendFrame[ackPos]);

} **else** {

System.***out***.println("Stimulate sending wrongly, doesn't has ack, the number of the frame is: "

+ *number*[*nextFrameToSend*]);

}

System.***out***.println();

DatagramPacket dp = **new** DatagramPacket(sendFrame, sendFrame.length, *address*, *host2\_Port*);

*ds*.send(dp);

} **else** **if** (method == lost) {//模拟传输丢失

SimpleDateFormat df = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

System.***out***.println("Current time: " + df.format(**new** Date()));

System.***out***.println("ack\_expected | next\_frame\_to\_send | frame\_expected are: " + *ackExpected* + " | "

+ *nextFrameToSend* + " | " + *frameExpected*);

**if** (sendFrame[ackPos] != notHasAck) {

System.***out***.println("Stimulate sending lost, the number and ack of the frame are: "

+ *number*[*nextFrameToSend*] + " | " + sendFrame[ackPos]);

} **else** {

System.***out***.println("Stimulate sending lost, doesn't has ack, the number of the frame is: "

+ *number*[*nextFrameToSend*]);

}

System.***out***.println();

} **else** **if** (method == right) {//模拟正确传输

SimpleDateFormat df = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

System.***out***.println("Current time: " + df.format(**new** Date()));

System.***out***.println("ack\_expected | next\_frame\_to\_send | frame\_expected are: " + *ackExpected* + " | "

+ *nextFrameToSend* + " | " + *frameExpected*);

**if** (sendFrame[ackPos] != notHasAck) {

System.***out***.println("Stimulate sending rightly, the number and ack of the frame are: "

+ *number*[*nextFrameToSend*] + " | " + sendFrame[ackPos]);

} **else** {

System.***out***.println("Stimulate sending rightly, doesn't has ack, the number of the frame is: "

+ *number*[*nextFrameToSend*]);

}

System.***out***.println();

DatagramPacket dp = **new** DatagramPacket(sendFrame, sendFrame.length, *address*, *host2\_Port*);

*ds*.send(dp);

}

}

**·发送线程**

//发送线程，调用fillAndSend函数发送

**public** **void** Send() **throws** Exception {

**while** (*nbuffered* != 0) {//缓存区不为空

**if** (*waitForResend* == **true**) {

**continue**;

}

**if**(*waitForResend* == **true** && *overtimeFlag* % 3 != 0) {

**continue**;

}

**if** (*needToResend* == **true**) {//重发出错帧后缓存区中的所有帧

*needToResend* = **false**;

*nextFrameToSend* = *ackExpected*;

**for** (**int** i = 1; i <= *nbuffered*; i++) {

fillAndSend(right);//均模拟正确发送

filterRank++;

*nextFrameToSend* = Inc(*nextFrameToSend*);

TimeUnit.***MILLISECONDS***.sleep(500);

**if** (*needToResend* == **true**) {//重发过程中接收超时，又需要重发

i = 1;

*needToResend* = **false**;

*nextFrameToSend* = *ackExpected*;

}

}

**continue**;

}

//needToResend为false的情况

**if** ((filterRank - firstError) % filterError == 0) {

fillAndSend(error);

} **else** **if** ((filterRank - firstLost) % filterLost == 0) {

fillAndSend(lost);

} **else** {

fillAndSend(right);

}

filterRank++;

*nextFrameToSend* = Inc(*nextFrameToSend*);

TimeUnit.***MILLISECONDS***.sleep(500);

}

//缓冲区为空，表示数据已经发送完毕，根据lastSendAck的取值判断是否一直发送确认帧

**byte** sendFrame[] = **new** **byte**[frameLen];

**while** (*lastSendAck* == **false**) {

sendFrame[kindPos] = notHasData;

sendFrame[ackPos] = (**byte**) ((*frameExpected* + *MAX\_SEQ*) % (*MAX\_SEQ* + 1));

**if** (*receiveInfoFinished* == **true**) {//收到最后一个数据帧

*lastSendAck* = **true**;

}

SimpleDateFormat df = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

System.***out***.println("Current time: " + df.format(**new** Date()));

System.***out***.println("Send a frame, only has ack, ack is: " + sendFrame[ackPos]);

System.***out***.println();

DatagramPacket dp = **new** DatagramPacket(sendFrame, sendFrame.length, *address*, *host2\_Port*);

*ds*.send(dp);

TimeUnit.***MILLISECONDS***.sleep(1000);

**if** (*receiveInfoFinished* == **true**) {

*lastSendAck* = **true**;

}

}

}

**·主函数**

**public** **static** **void** main(String[] args) **throws** Exception {

**try** {

*is* = **new** FileInputStream(**new** File("D:\\desktop\\SendText1.txt"));

*os* = **new** FileOutputStream(**new** File("D:\\desktop\\ReceiveText2.txt"));

*ds* = **new** DatagramSocket(*host1\_Port*);

*ds*.setSoTimeout(2000);

*address* = InetAddress.*getByName*(**null**);

} **catch** (Exception e) {

e.printStackTrace();

}

UDPHost1 send = **new** UDPHost1();

UDPHost1 receive = **new** UDPHost1();

**new** Thread(receive).start();

TimeUnit.***MILLISECONDS***.sleep(1000);

send.Send();

**try** {

*is*.close();

*os*.close();

*ds*.close();

} **catch** (Exception e) {

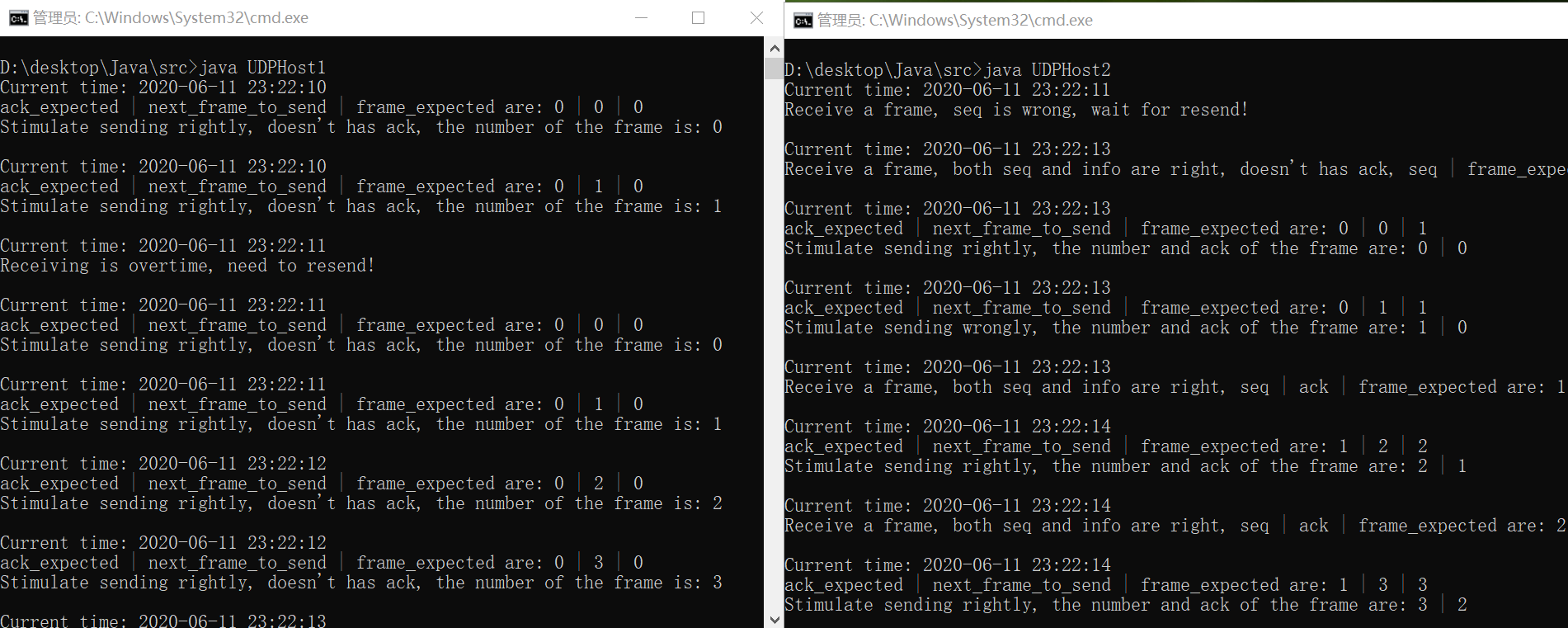
e.printStackTrace();

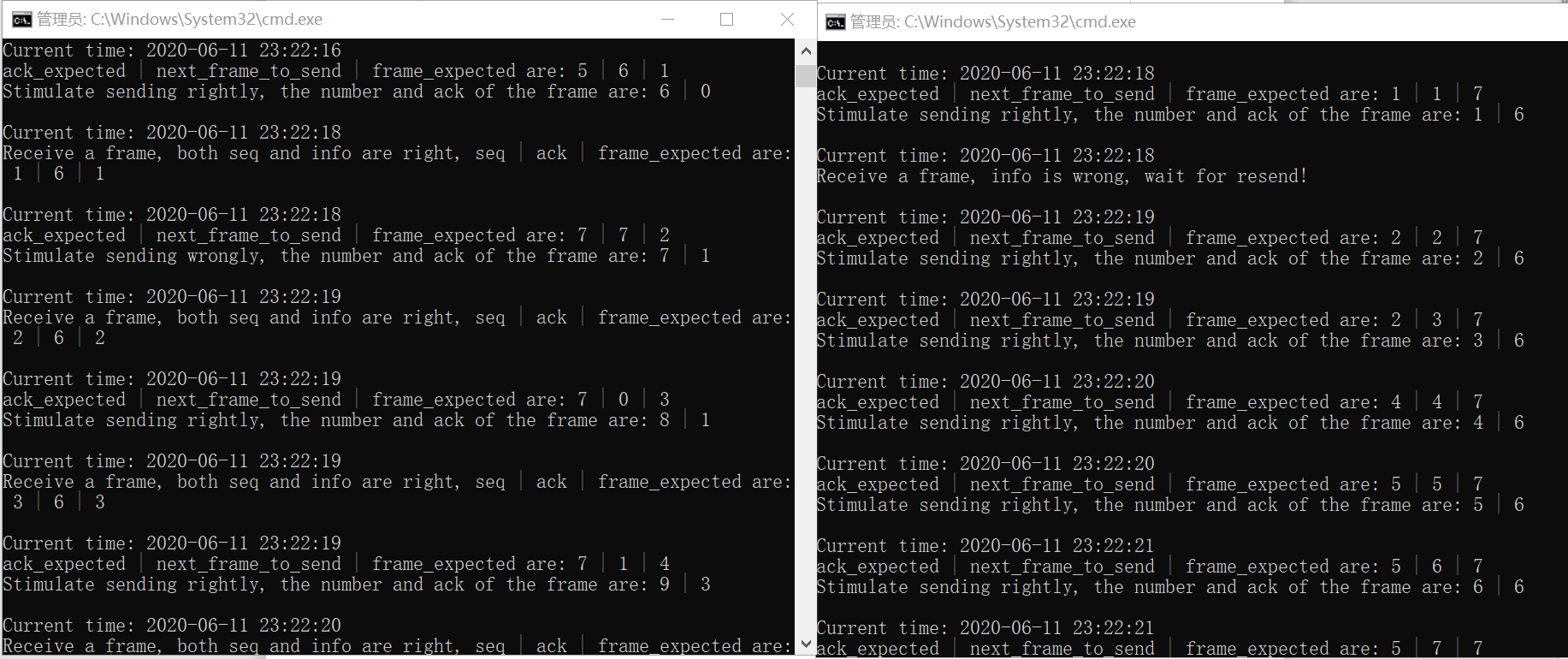
}

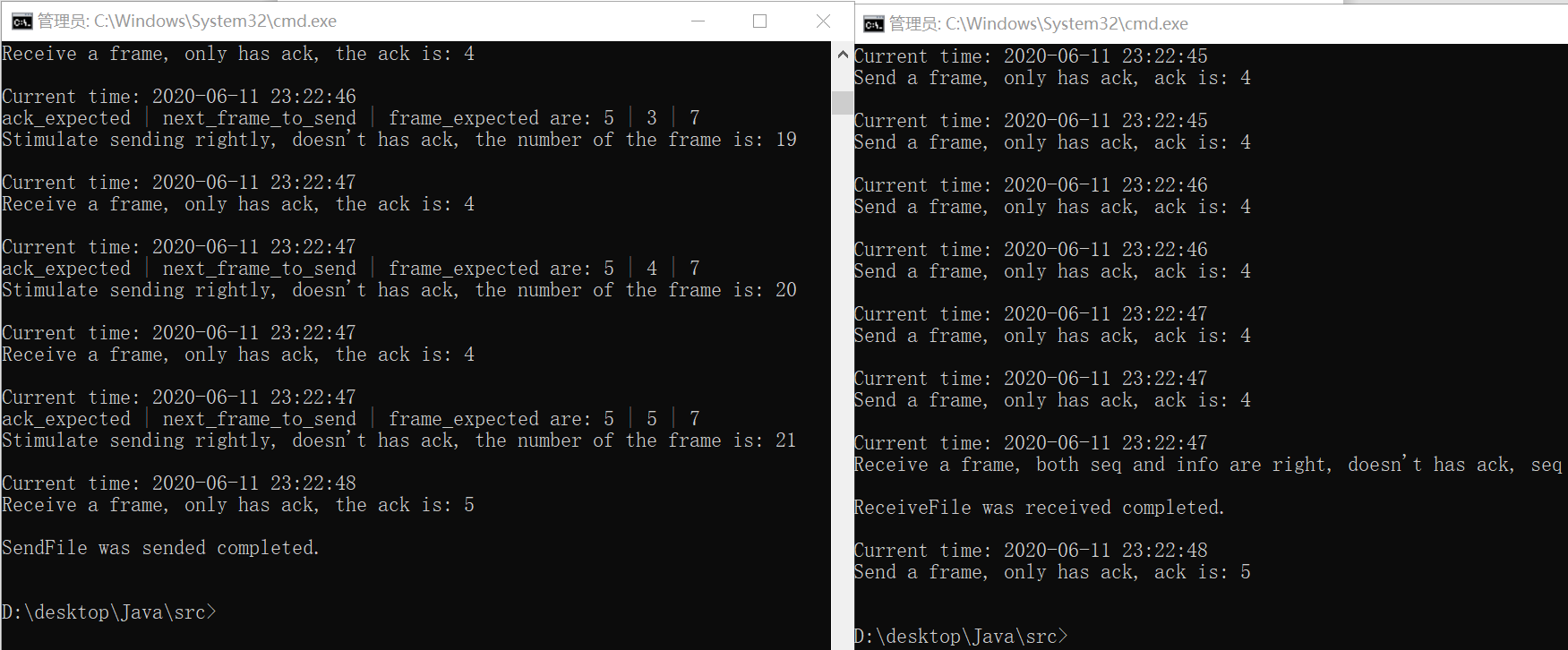
}

1. **实验结果**

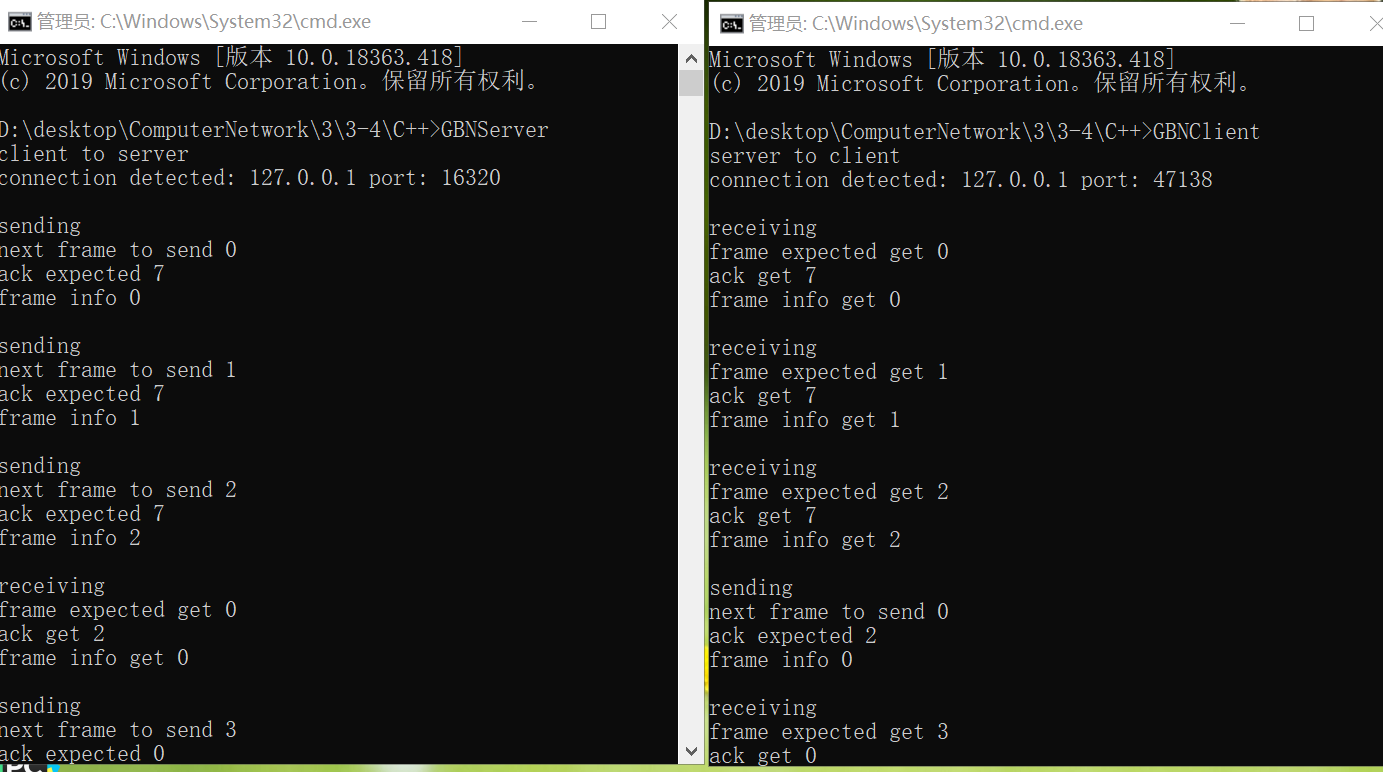
**·Java**







**·C++**



1. **实验总结**

这个实验真的很难，对于Java，一端的代码就写了将近500行，从思考到写完java花了五天，python也写了两天，然而bug太多了，运行有很大的问题。这是我写的这么多计网实验中最扎心的一个，觉得自己会做，然而太麻烦太难调试了，无论结果与否，也只能这样了。