云南大学数学与统计学院  
《计算机网络实验》上机实践报告

|  |  |  |
| --- | --- | --- |
| 课程名称：计算机网络实验 | 年级：2015级 | 上机实践成绩： |
| 指导教师：陆正福 | 姓名：刘鹏 | 专业：信息与计算科学 |
| 上机实践名称：基于UDP/IP协议与Socket接口的可靠通信编程实验 | 学号：20151910042 | 上机实践日期：2018-11-27 |
| 上机实践编号：No.05 | 组号： |  |

# 实验目的

1. 熟悉基于UDP/IP协议与Socket接口的网络通信编程实验
2. 熟悉教材第三章的基本概念
3. 理解并掌握可靠数据传输的基本机制。

# 实验内容

1. 查在前期实验（计算机网络实验4）的基础上，编程实现主讲教材第3章（Chapter 3 Transport Layer）RDT协议的各个版本。
2. 剖析消息处理与消息交换在RDT协议的可靠性增强中的作用。

# 实验平台

Windows 10 Pro 1803；

Cygwin GCC编译器。

# 程序代码

## 实验分析

UDP本身已经是一种可以使用的传输层协议了，但是其本身是一个面向数据包的、非连接的不可靠协议。本实验的目的应该是利用这个现成的传输协议，加上一部分冗余、重传，使得可以在UDP的基础上达成可靠交易。这个实验需要借鉴很多RDT协议的细节。

RDT 3.0描述如下，这是一个停等协议，简而言之，对于发送方而言：一次发送一个分组，然后一直等待，等待超时就重发，收到错误的ACK或者收到了破损的分组就重发；对于接收方而言，该协议与RDT 2.2没有丝毫区别。

## Java实现分析

如果涉及到Java代码实现，就必须考虑定时器的存在。由于这里是实验性质的，所以不考虑EstimatedRTT的处理，仅仅是单纯地选择一个停等时间。Java有定时器对象，可以如下考虑，建立一个定时器对象，具体是建立一个TimerTask对象，该对象在Java中被认为是可以被安排执行一次，或者在Timer对象的管理下重复执行。TimerTask对象是不可复用的，一旦一个TimerTask对象在Timer对象的操作下被执行或者取消了，那么之后所有对此对象的尝试性执行都会被认定为非法[1]。在Timer线程里面，嗅探一下socket接口，查看接收到回复没有，如果收到了不合法的回复，如packet破损、序号不对等，则继续查看。一旦发现超时了（TimerTask被Timer控制，delay之后自动停止），就不再做“超时临界处”的回复检查，直接重传并启动定时器。

Java实现比较适合用并发模式，Thread构造器只需要一个Runnable对象。调用Thread对象的start()方法为该线程执行必需的初始化操作，然后调用Runnable的run()方法，以便在这个新线程中启动该任务。



图 1 RDT 3.0 接收方

与接收方不同，发送方的有限状态机（Finite State Machine, FSM）比较复杂，因为RDT 3.0把主动权和判断任务都交给了发送方。



图 2 RDT 3.0发送方

## 实验设计反思

解决了多线程的问题，剩下的就是RDT 3.0的原理的实现。由于是仿真，所以需要模拟现实中的扰动：接收方发回了错误的ACK或者接收方没有收到packet、收到了损坏的packet。这里我本想采用高斯噪声对数据进行干扰，然而后来我发现这个操作相当麻烦，尤其是对于不太熟悉的数据类型。所以干脆就做一点简单的干扰。在实际的操作中，没有接收到正确ACK之后不可能一直重传，所以这里是等待8个时间片，也就是说，8次重传仍然收不到正确分组就不传了。在实验中发现，当设置出错次数为20的时候（即在第21次才反馈正确的ACK），发送方是一直都接收不到正确回文的。但是出错次数为7的时候，总是可以在最后一次接受正确。

比较遗憾的是，由于Java没有给Packet的校验和提供接口，所以没有能够实现校验。但是这个机制就是在本实验代码的基础上加一个判断，与我主动发送错误的反馈没有区别，所以在这个版本的报告里不涉及checksum，只涉及重传反馈。

基于网络的编程涉及到了很多并发的知识，一旦涉及到计时器就必然有这种线程操作。而RDT 3.0仅仅是一个原理性的协议，与真实的、具体的协议相比，还缺乏了很多细节。本次实验集中精力完成了客户端的线程计时，服务端（接收端）的反馈重传。

## 使用Java和Android实现基于UDP/IP协议与Socket接口的可靠通信

### Android App端Java代码

由于缩进问题，在此处把TAB改为2空格。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111 | package com**.**example**.**newton**.**myapplication**;**  **import** android**.**support**.**v7**.**app**.**AppCompatActivity**;**  **import** android**.**os**.**Bundle**;**  **import** android**.**widget**.**EditText**;**  **import** android**.**widget**.**TextView**;**  **import** android**.**widget**.**Button**;**  **import** java**.**io**.**IOException**;**  // Java网络类库  **import** java**.**net**.**DatagramPacket**;**  **import** java**.**net**.**DatagramSocket**;**  **import** java**.**net**.**InetAddress**;**  **import** java**.**net**.**UnknownHostException**;**  **import** java**.**net**.**SocketException**;**  **import** android**.**view**.**View**;**  // Java的定时器类，通过多线程的方式实现定时做某事  **import** java**.**util**.**Timer**;**  **import** java**.**util**.**TimerTask**;**  public class MainActivity **extends** AppCompatActivity **{**  EditText et**;**  TextView tv**;**  Button button**;**  int i **=** 0**;**  @Override  protected void onCreate**(**Bundle savedInstanceState**)** **{**  **super.**onCreate**(**savedInstanceState**);**  setContentView**(**R**.**layout**.**activity\_main**);**  et **=** findViewById**(**R**.**id**.**et**);**  tv **=** findViewById**(**R**.**id**.**tv**);**  button **=** findViewById**(**R**.**id**.**button**);**  **}**  // 这一部分代码继承自实验报告4，同样采取一个点击操作，发送并会回文一个字符串。  public void onClick**(**View view**)** **{**  **new** Thread**()** **{**  @Override  public void run**()** **{**  **super.**run**();**  // 开启一个Timer线程来做定时工作  final Timer timer **=** **new** Timer**();**  timer**.**scheduleAtFixedRate**(new** TimerTask**()** **{**  @Override  public void run**()** **{**  **try** **{**  // 通过对 EditText 这个对象进行获取数据，得到一个字节数组  byte**[]** bytes **=** et**.**getText**().**toString**().**getBytes**();**  // 接收数据  InetAddress address **=** InetAddress**.**getByName**(**"192.168.1.75"**);**  // 1. 构造数据包  DatagramPacket packet **=** **new** DatagramPacket**(**bytes**,** bytes**.**length**,** address**,** 2222**);**  // 2. 创建数据报套接字并将其绑定到本地主机上的指定端口。  DatagramSocket socket **=** **new** DatagramSocket**();**  // 3. 从该套接字发送 packet  socket**.**send**(**packet**);**  // 建立一个空的 packet，用来放置接收到的ACK  // 这么做的目的是为了后期避免在线程之外无法引用 socket  final byte**[]** bytes1 **=** **new** byte**[**1024**];**  System**.**out**.**println**(**bytes1**[**0**]);**  DatagramPacket receiverPacket **=** **new** DatagramPacket**(**bytes1**,** bytes1**.**length**);**  socket**.**receive**(**receiverPacket**);**  runOnUiThread**(new** Runnable**()** **{**  @Override  public void run**()** **{**  tv**.**setText**(new** String**(**bytes1**,** 0**,** bytes1**.**length**));**  **}**  **});**  **if** **(**bytes**[**1**]** **==** bytes1**[**1**])** **{**  System**.**out**.**println**(**"正确接收到了"**);**  cancel**();**  timer**.**cancel**();**  socket**.**close**();**  **}** **else** **{**  System**.**out**.**println**(**"出错了！"**);**  **}**  **}** **catch** **(**UnknownHostException e**)** **{**  e**.**printStackTrace**();**  **}** **catch** **(**SocketException e**)** **{**  e**.**printStackTrace**();**  **}** **catch** **(**IOException e**)** **{**  e**.**printStackTrace**();**  **}**  **}**  **},** 0**,** 8**);**  System**.**out**.**println**(**"一切将要结束了"**);**  **}**  **}.**start**();**  **}**  **}** |

### Android App端运行结果

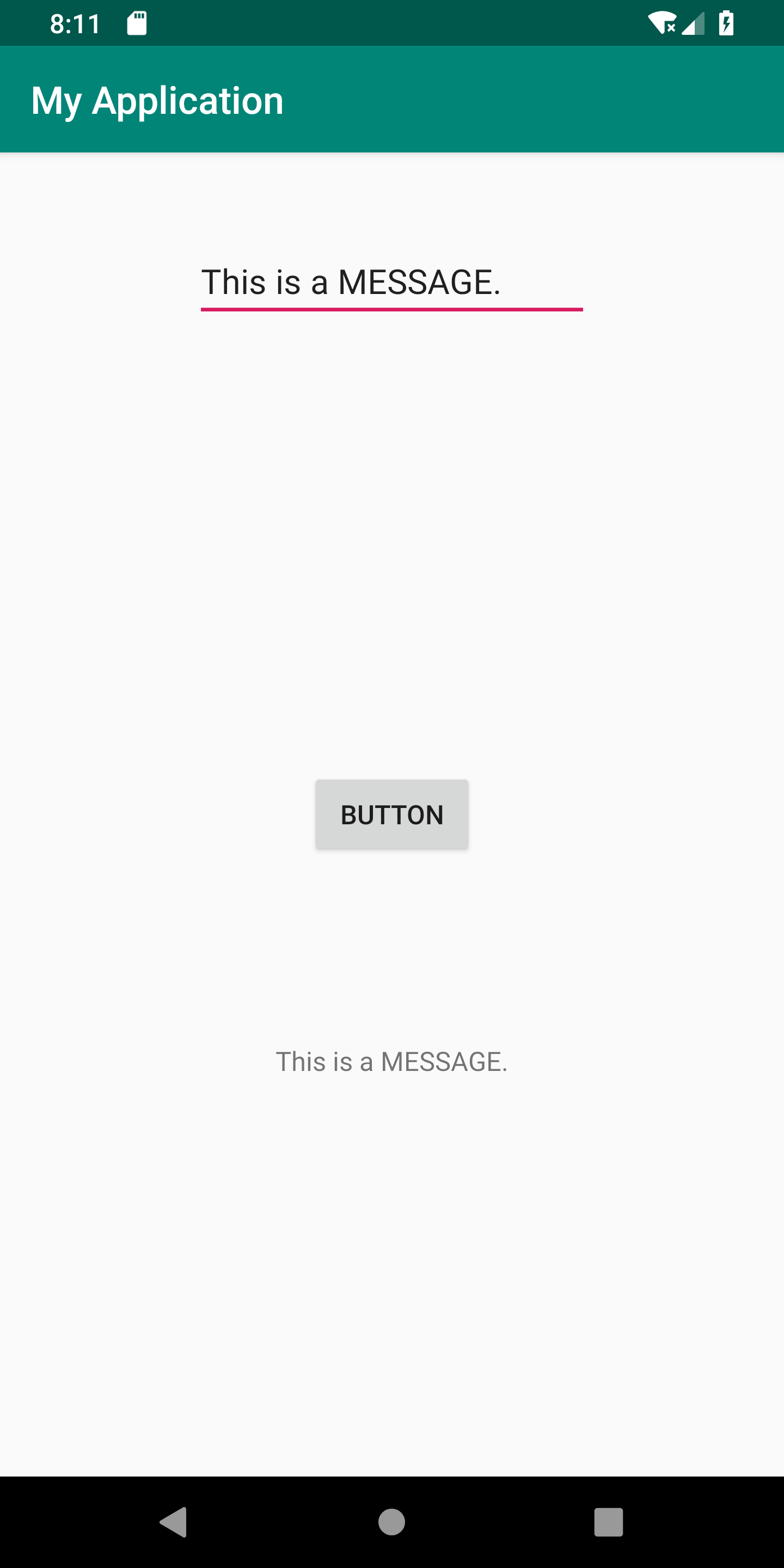


图 3 App截图

### App后台反馈

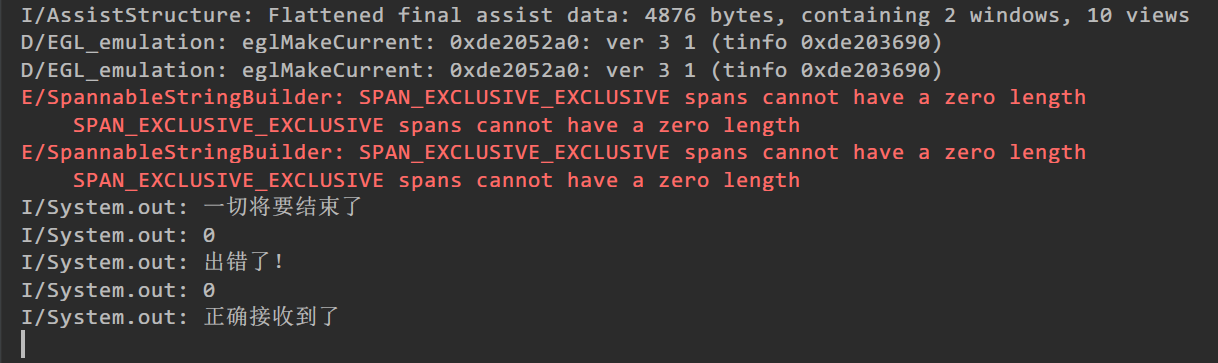


图 4 App后台成功接收到了反馈

### Server端Java代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | **import** java**.**io**.**IOException**;**  **import** java**.**net**.**DatagramPacket**;**  **import** java**.**net**.**DatagramSocket**;**  **import** java**.**net**.**InetSocketAddress**;**  public class UDPServer **{**  public static void main**(**String**[]** args**)** **throws** IOException **{**  // 生成一个1MB的存储空间  byte**[]** buf **=** **new** byte**[**1024**];**  // 1. 接收数据  // (1) 创建接受数据的数据包  DatagramPacket packet **=** **new** DatagramPacket**(**buf**,** buf**.**length**);**  // (2) 创建UDP的Socket  DatagramSocket socket **=** **new** DatagramSocket**(**2222**);**  // 利用 i 这个变量进行若干次干扰！  int i **=** 0**;**  // (3) 接收数据  **while** **(true)** **{**  System**.**out**.**println**(**"===Server 端开始监听==="**);**  socket**.**receive**(**packet**);**  // (4) 处理数据  System**.**out**.**println**(**"服务端：" **+** **new** String**(**buf**,** 0**,** buf**.**length**));**  **if** **(**i **==** 7**)** **{**  // 2. 返回数据  DatagramPacket p **=** **new** DatagramPacket**(**buf**,** buf**.**length**,** packet**.**getAddress**(),** packet**.**getPort**());**  i **=** 0**;**  socket**.**send**(**p**);**  **}** **else** **{**  **for** **(**i **=** 0**;** i **<** 7**;** i**++)** **{**  byte**[]** buf2 **=** **new** byte**[**1024**];** // 发送一个错误的数组回去  DatagramPacket p **=** **new** DatagramPacket**(**buf2**,** buf2**.**length**,** packet**.**getAddress**(),** packet**.**getPort**());**  socket**.**send**(**p**);**  System**.**out**.**println**(**"===Server 返回了一个错误的信息==="**);**  **}**  **}**  **}**  **}**  **}** |

### Server端运行结果

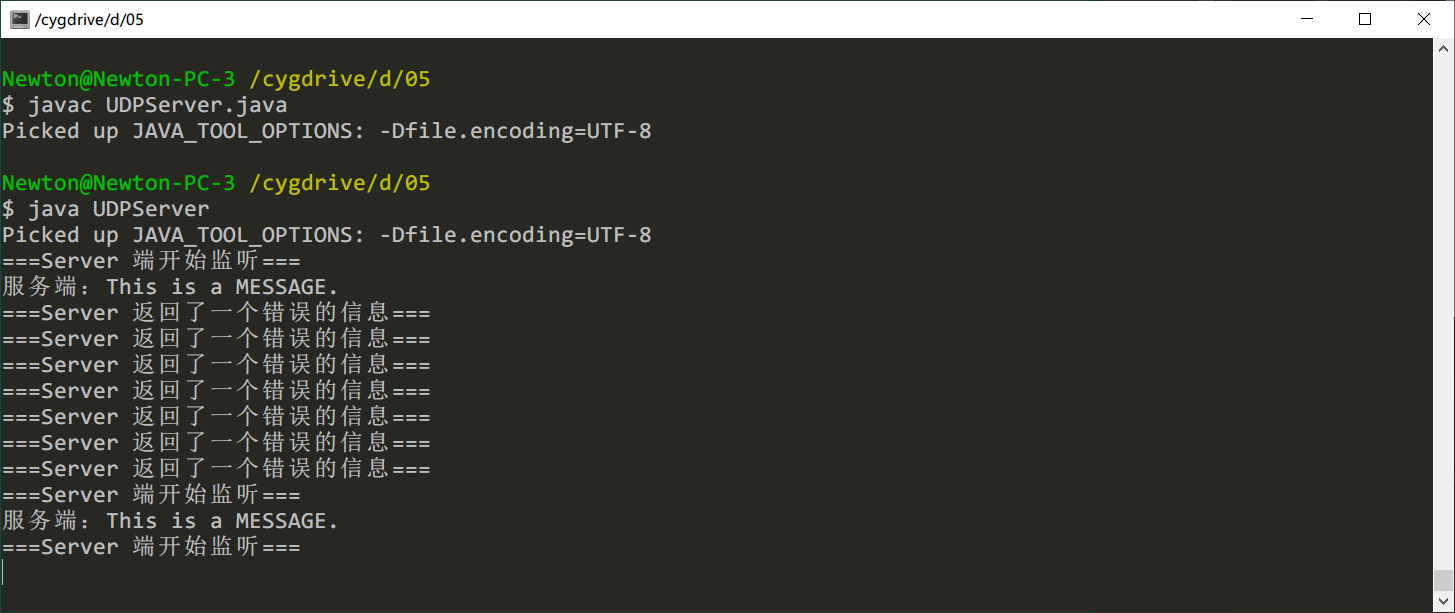


图 5 Server端接收情况

# 实验体会

通过本次实验，学习了如何在不使用强纠错码的情况下，通过交互式重传机制来进行可靠通信。由于编程水平有限，仅仅是实验性地完成了基于UDP的可靠通信传输。实验中面临的最大问题是对于Java语言的不熟悉。其实辩证地思考一下，换做其他任何一门语言都不会对网络编程部分很熟悉，所以Java也一样，而且用Java还有学习新语言的好处。

实验中最困难的部分在于使用Java的定时器对象，Timer的某个方法里面需要TimerTask对象做参数，如果前期做好了TimerTask对象的构建工作，那么后期将很难在收到ACK之后取消定时器。后来通过一番思考，我想到了一个绝妙的办法：把TimerTask的构建过程放在参数位置完成，这样就可以一举解决无法调用Timer的cancel方法的问题。

# 参考文献

[1] java Timer: <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Timer.html>

[2] java TimerTask: <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/TimerTask.html>