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：**	

一、 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。

二、 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。

三、 实验平台

Windows 10 Pro 1803；
Cygwin GCC 编译器。

四、 程序代码

4.1 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。

4.2 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。

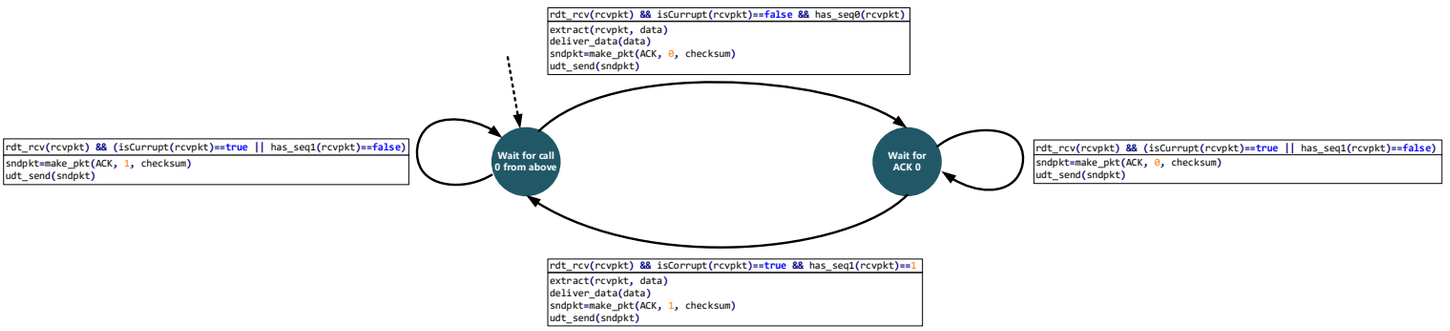

图 1 RDT 3.0 接收方

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。

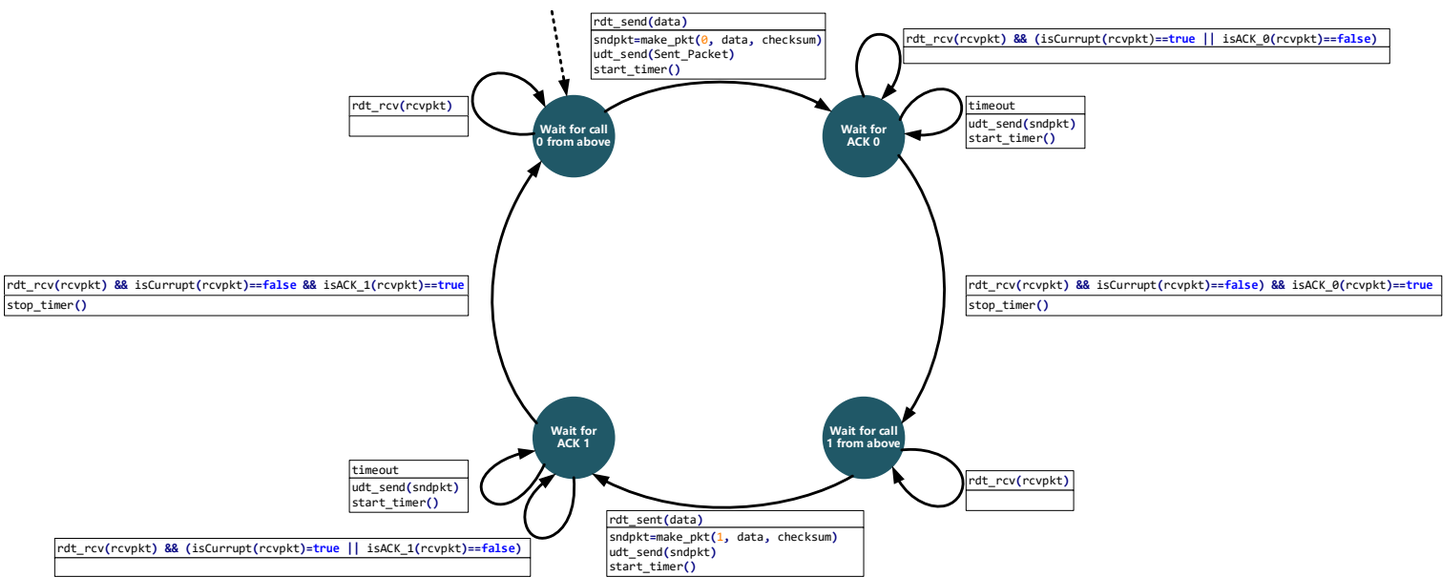

图 2 RDT 3.0 发送方

4.3 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。

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

4.4.1 Android App 端 Java 代码

由于缩进问题，在此处把 TAB 改为 2 空格。

```
1   package com.example.newton.myapplication;
2
3   import android.support.v7.app.AppCompatActivity;
4   import android.os.Bundle;
5   import android.widget.EditText;
6   import android.widget.TextView;
7   import android.widget.Button;
8
9   import java.io.IOException;
10
11  // Java 网络类库
12  import java.net.DatagramPacket;
13  import java.net.DatagramSocket;
14  import java.net.InetAddress;
15  import java.net.UnknownHostException;
16  import java.net.SocketException;
17
18  import android.view.View;
19
20  // Java 的定时器类，通过多线程的方式实现定时做某事
21  import java.util.Timer;
22  import java.util.TimerTask;
23
24  public class MainActivity extends AppCompatActivity {
25
26    EditText et;
27    TextView tv;
28    Button button;
29
30    int i = 0;
31
32    @Override
33    protected void onCreate(Bundle savedInstanceState) {
34      super.onCreate(savedInstanceState);
```

```
35      setContentView(R.layout.activity_main);
36
37      et = findViewById(R.id.et);
38      tv = findViewById(R.id.tv);
39      button = findViewById(R.id.button);
40    }
41
42    // 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。
43    public void onClick(View view) {
44
45      new Thread() {
46        @Override
47        public void run() {
48          super.run();
49
50          // 开启一个 Timer 线程来做定时工作
51          final Timer timer = new Timer();
52
53          timer.scheduleAtFixedRate(new TimerTask() {
54            @Override
55            public void run() {
56              try {
57
58                // 通过对 EditText 这个对象进行获取数据，得到一个字节数组
59                byte[] bytes = et.getText().toString().getBytes();
60
61                // 接收数据
62                InetAddress address = InetAddress.getByName("192.168.1.75");
63
64                // 1. 构造数据包
65                DatagramPacket packet = new DatagramPacket(bytes, bytes.length, address, 2222);
66
67
68                // 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。
69                DatagramSocket socket = new DatagramSocket();
70
71                // 3. 从该套接字发送 packet
72                socket.send(packet);
73
74                // 建立一个空的 packet，用来放置接收到的 ACK
75                // 这么做的目的是为了后期避免在线程之外无法引用 socket
76                final byte[] bytes1 = new byte[1024];
77                System.out.println(bytes1[0]);
78                DatagramPacket receiverPacket = new DatagramPacket(bytes1, bytes1.length);
79
80                socket.receive(receiverPacket);
81
82                runOnUiThread(new Runnable() {
83                  @Override
```

```
84                  public void run() {
85                    tv.setText(new String(bytes1, 0, bytes1.length));
86                  }
87                });
88
89                if (bytes[1] == bytes1[1]) {
90                  System.out.println("正确接收到了");
91                  cancel();
92                  timer.cancel();
93                  socket.close();
94                } else {
95                  System.out.println("出错了！");
96                }
97
98              } catch (UnknownHostException e) {
99                e.printStackTrace();
100             } catch (SocketException e) {
101               e.printStackTrace();
102             } catch (IOException e) {
103               e.printStackTrace();
104             }
105           }
106         }, 0, 8);
107         System.out.println("一切将要结束了");
108       }
109     }.start();
110   }
111 }
```

4.4.2 Android App 端运行结果

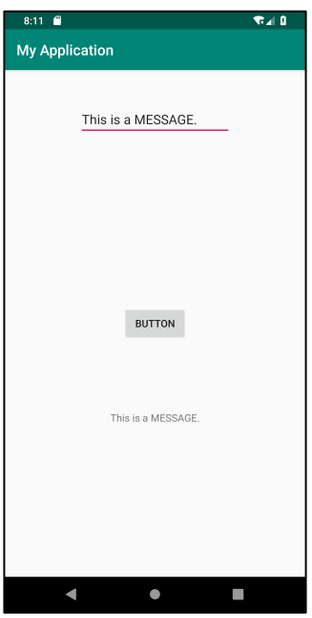

图 3 App 截图

4.4.3 App 后台反馈

```
I/AssistStructure: Flattened final assist data: 4876 bytes, containing 2 windows, 10 views
D/EGL_emulation: eglMakeCurrent: 0xde2052a0: ver 3 1 (tinfo 0xde203690)
D/EGL_emulation: eglMakeCurrent: 0xde2052a0: ver 3 1 (tinfo 0xde203690)
E/SpannableStringBuilder: SPAN_EXCLUSIVE_EXCLUSIVE spans cannot have a zero length
    SPAN_EXCLUSIVE_EXCLUSIVE spans cannot have a zero length
E/SpannableStringBuilder: SPAN_EXCLUSIVE_EXCLUSIVE spans cannot have a zero length
    SPAN_EXCLUSIVE_EXCLUSIVE spans cannot have a zero length
I/System.out: 一切将要结束了
I/System.out: 0
I/System.out: 出错了!
I/System.out: 0
I/System.out: 正确接收到了
```

图 4 App 后台成功接收到了反馈

4.4.4 Server 端 Java 代码

```java
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
```

```
33          DatagramPacket p = new DatagramPacket(buf, buf.length, packet.getAddress(), packet.get-
    Port());
34          i = 0;
35          socket.send(p);
36        } else {
37          for (i = 0; i < 7; i++) {
38            byte[] buf2 = new byte[1024]; // 发送一个错误的数组回去
39            DatagramPacket p = new DatagramPacket(buf2, buf2.length, packet.getAddress(),
40  packet.getPort());
              socket.send(p);
41            System.out.println("===Server 返回了一个错误的信息===");
42          }
43        }
44      }
45    }
46  }
```

4.4.5 Server 端运行结果

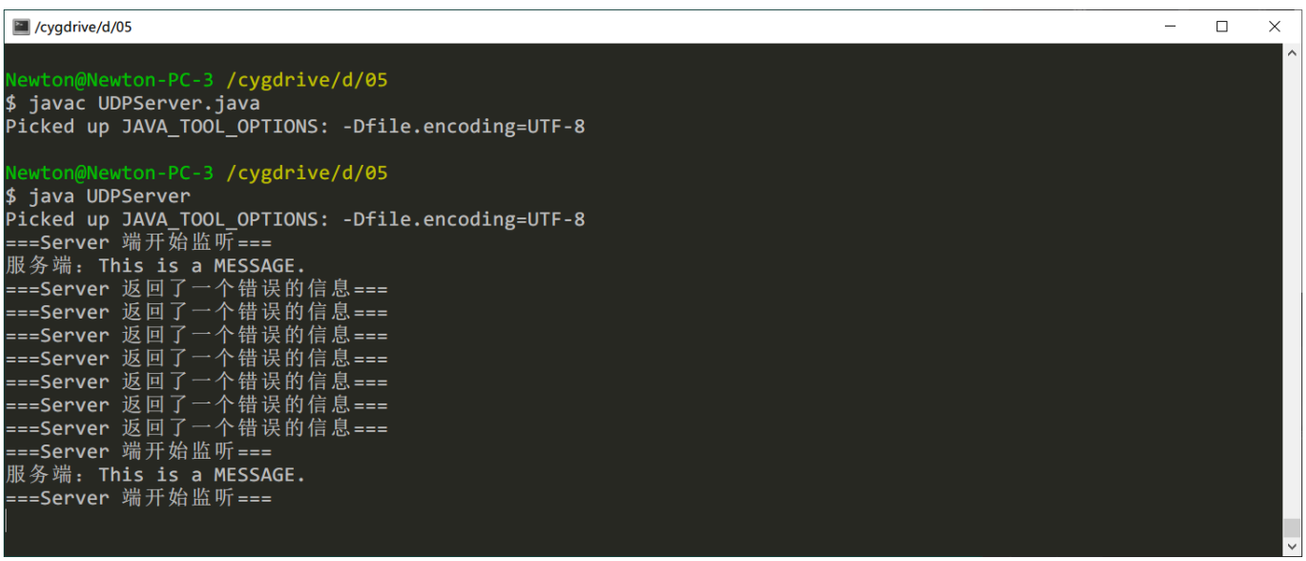

图 5 Server 端接收情况

五、 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。

六、 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