华南理工大学

South China University of Technology

网络通信原理实验报告

实验名称： 移动端对 PC 机可视远程控制的实现

学　　院：电子与信息学院

专　　业：电子与通信工程

班　　级：14 级硕士 4 班

学生姓名：纪　强

学　　号：201421010283

课程教师：冯穗力

华 南 理 工 大 学

二〇一五年五月

目录

第一章 实验综述

1.1 设计目的

本实验是基于 Java 语言设计的在 Android 手机端能够远程控制电脑，分为服务器端和客户端，并能在客户端实现可视化电脑界面，能够实时控制电脑，实现键盘和鼠标的功能，在手机端即可随时随地操作办公家庭中的电脑，对程序人员来说尤其方便，经常电脑跑较占内存的程序时电脑很难同时做其他的事情，这是就可以通过手机端随时关注程序的运行情况。功能比较简单，但很实用。

1.2 设计目标

(1)只需知道电脑主机的 IP 地址和端口号，即可实现电脑与手机的连接；
(2)实现电脑屏幕在手机端的共享；
(3)手机端能实时控制电脑，控制鼠标和键盘输入；
(4)用 TCP 和 UDP 协议分别实现上述功能，以直观理解两种协议的不同。

第二章 实验技术基础

2.1 客户服务器方式

在网络边缘的端系统中运行的程序之间的通信方式通常可划分为两大类：客户服务器方式(C/S 方式)和对等方式(P2P 方式)，其中在因特网上 C/S 模式是最常用的也是最传统的方式，本实验正是基于这种方式设计的。

客户是服务的请求方，服务器是服务的提供方。在实际应用中，客户程序被用户调用后运行，在通信时主动向远地服务器发起通信请求服务，因此，客户程序必须知道服务器程序的地址。客户程序不需要提供特殊的硬件和很复杂的操作系统。服务器程序是一种专门用来提供某种服务的程序，可同时处理多个远地或本地客户的请求；系统启动后即自动调用并一直不断地运行，被动地等待并接受来自各地的客户的通信请求，服务器是不需要知道客户的地址的；服务器端一般需要强大的硬件和高级的操作系统支持。客户端和服务器建立相关的通信后，通信是可以双向的，客户和服务器都可以发送和接收数据。

2.2 TCP 协议

TCP 是一个面向连接的协议，所以在连接双方发送数据之前，都需要首先建立一条连接，数据传输后结束后要释放连接。这里的连接只不过是端系统中分配的一些缓存和状态变量，中间的分组交换机不维护任何连接状态信息。连接建立整个过程如下(即三次握手协议)：

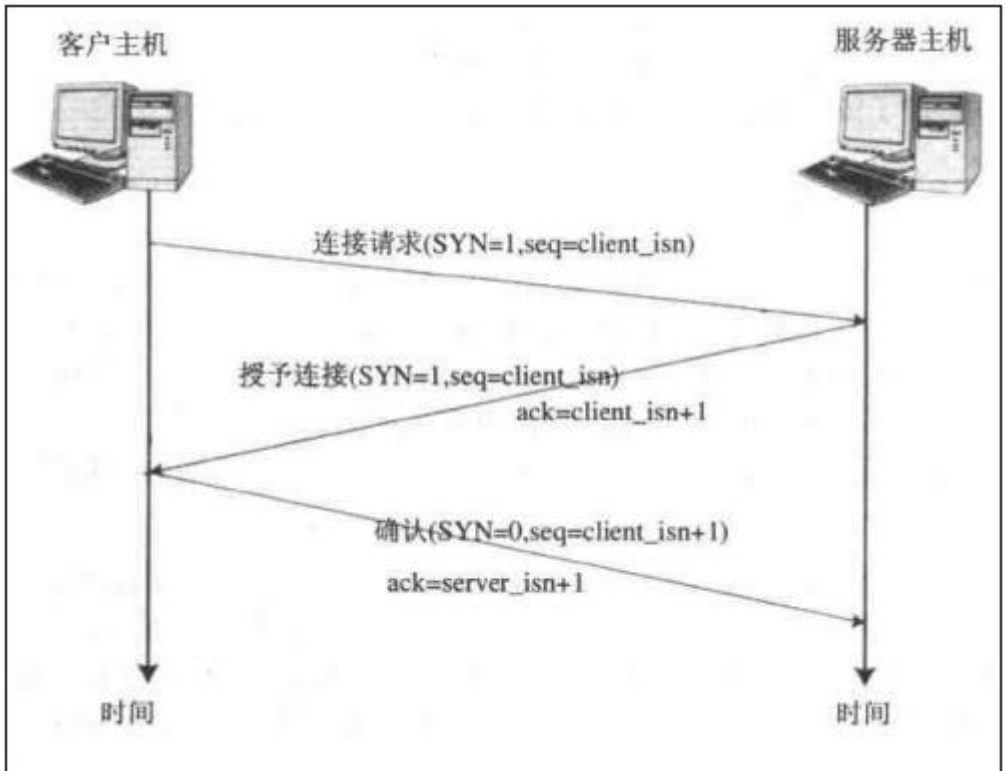
首先，客户机发送一个特殊的 TCP 报文段；

其次，服务器用另一个特殊的 TCP 报文段来响应；

最后，客户机再用第三个特殊报文段作为响应。

2.2.1 连接的建立

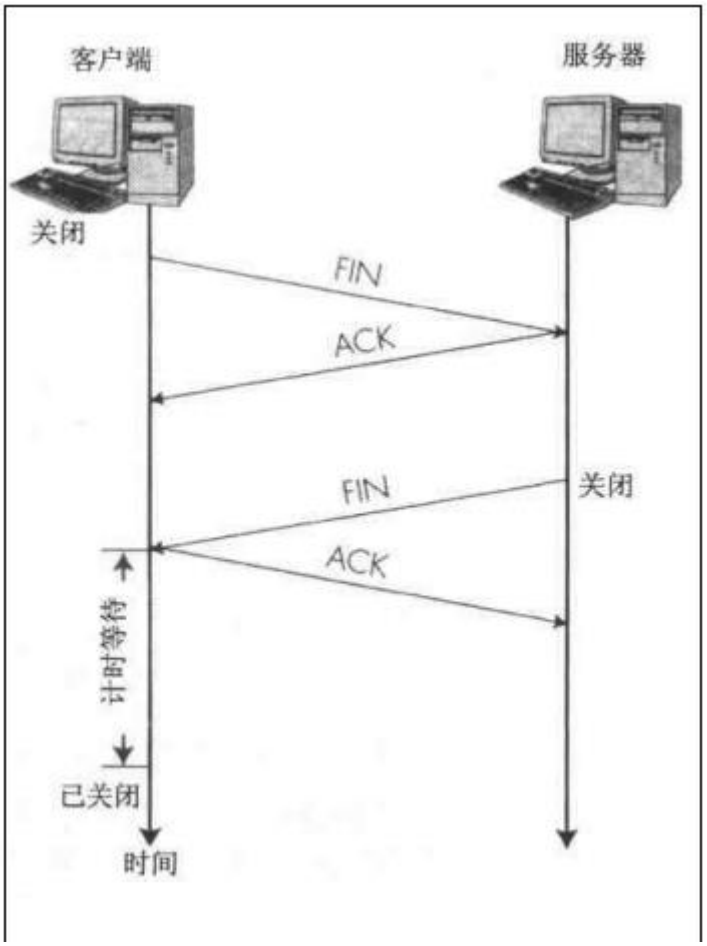
在建立连接的时候，客户端首先向服务器申请打开某一个端口(用 SYN 段等于 1 的 TCP 报文)，然后服务器端发回一个 ACK 报文通知客户端请求报文收到，客户端收到确认报文以后再次发出确认报文确认刚才服务器端发出的确认报文，至此，连接的建立完成。这就叫做三次握手。如果打算让双方都做好准备的话，一定要发送三次报文，而且只需要三次报文就可以了。



可以想见，如果再加上 TCP 的超时重传机制，那么 TCP 就完全可以保证一个数据包被送到目的地。

2.2.2 结束连接

TCP 有一个特别的概念叫做 half-close，这个概念是说，TCP 的连接是全双工（可以同时发送和接收）连接，因此在关闭连接的时候，必须关闭传和送两个方向上的连接。客户机给服务器一个 FIN 为 1 的 TCP 报文，然后服务器返回给客户端一个确认 ACK 报文，并且发送一个 FIN 报文，当客户机回复 ACK 报文后（四次握手），连接就结束了。

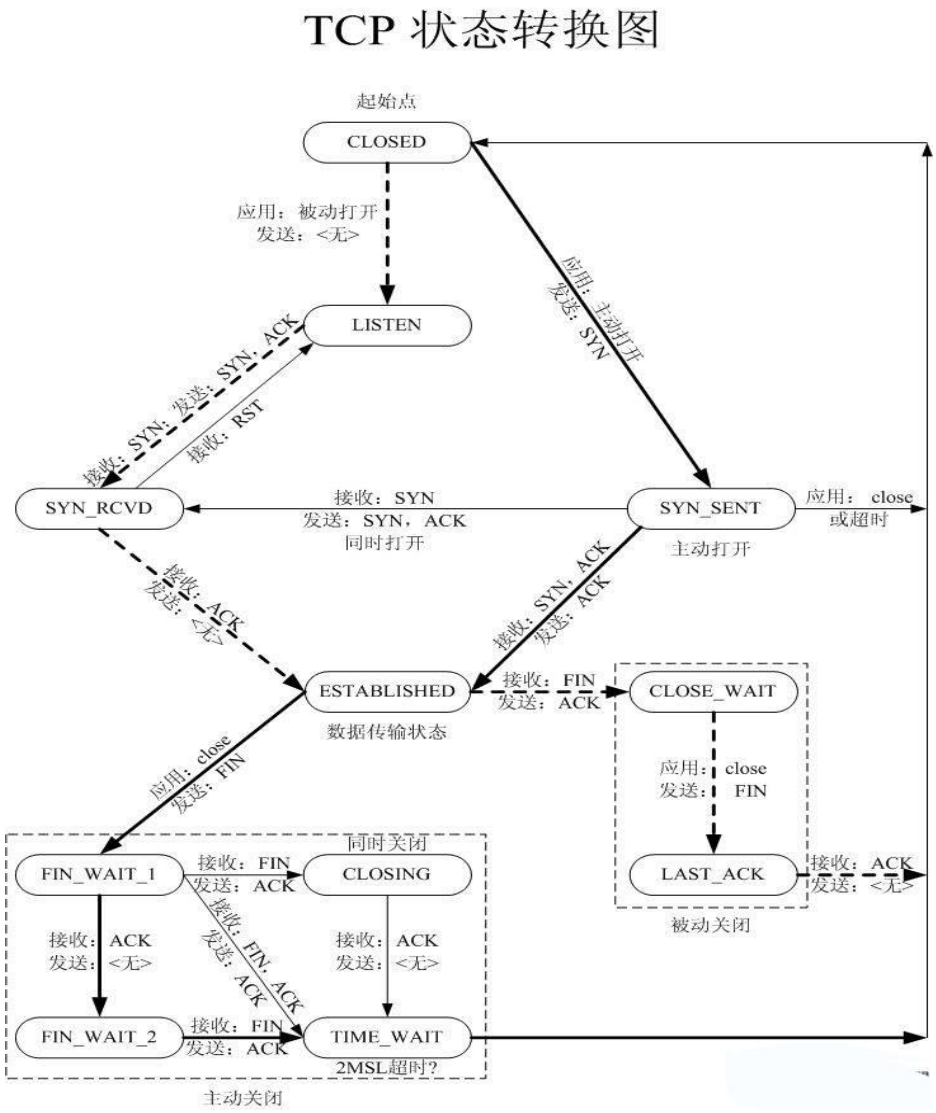


2.2.3 最大报文长度

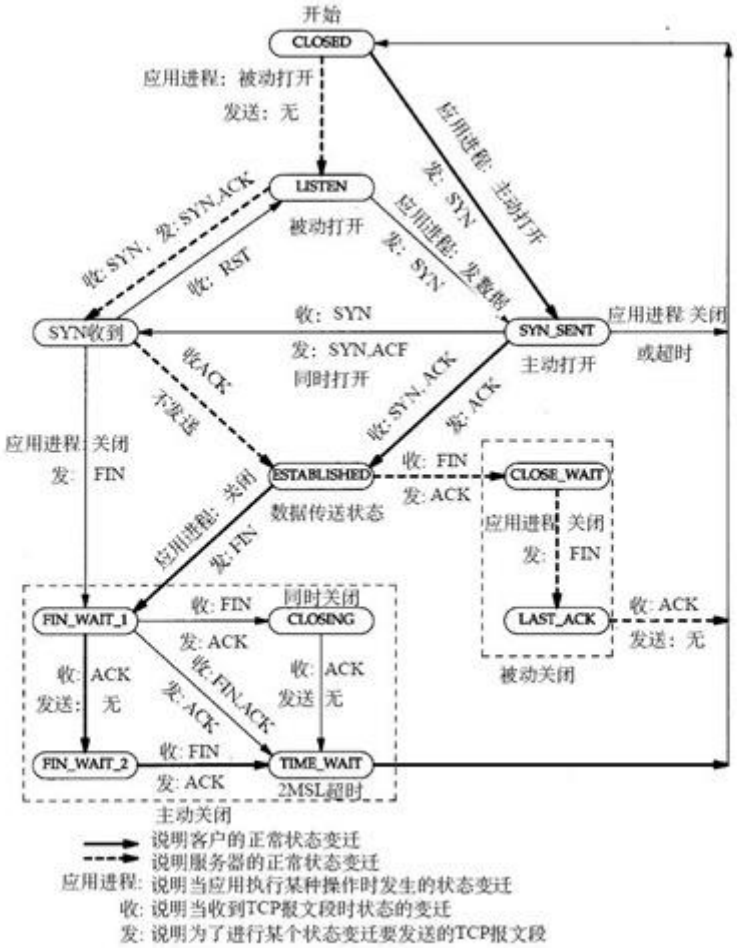
在建立连接的时候，通信的双方要互相确认对方的最大报文长度(MSS)，以便通信。一般这个 SYN 长度是 MTU 减去固定 IP 首部和 TCP 首部长度。对于一个以太网，一般可以达到 1460 字节。当然如果对于非本地的 IP，这个 MSS 可能就只有 536 字节，而且，如果中间的传输网络的 MSS 更佳的小的话，这个值还会变得更小。

2.2.4 TCP 的状态迁移图

如何的 TCP 状态图，这是一个看起来比较复杂的状态迁移图，因为它包含了两个部分---服务器的状态迁移和客户端的状态迁移，如果从某一个角度出发来看这个图，就会清晰许多，这里面的服务器和客户端都不是绝对的，发送数据的就是客户端，接受数据的就是服务器。



客户端应用程序的状态迁移图



客户端的状态可以用如下的流程来表示：
CLOSED->SYN_SENT->ESTABLISHED->FIN_WAIT_1->FIN_WAIT_2->TIME_WAIT->CLOSED

以上流程是在程序正常的情况下应该有的流程，从中的图中可以看到，在建立连接时，当客户端收到SYN报文的ACK以后，客户端就打开了数据交互地连接。而结束连接则通常是客户端主动结束的，客户端结束应用程序以后，需要经历FIN_WAIT_1，FIN_WAIT_2等状态，这些状态的迁移就是前面提到的结束连接的四次握手。

2.3 UDP 协议

与 TCP 不同的是，UDP 是面向无连接的，是面向报文的，尽最大努力交付，没有拥塞控制。

1．UDP 协议的作用

IP 协议无法区别同一个主机系统上的多个应用程序。UDP 采用端口标识同一主机上的不同应用程序。

无法采取进程 ID 来标识不同应用程序的原因：

1）系统中应用程序的进程 ID 分配和销毁是动态的，发送方无法确定该应用程序的进程 ID 是什么

2）有时可能在一个进程中实现多个功能，进程就需要对数据包进行区分，以判断是用以实现哪个功能的，使用进程 ID 无法做到这点。

3）有时需要访问主机上的某个标准服务时，无须知道实现该服务的程序是哪个，也就无须知道该进程 ID，只需要一个统一标识就可。

采用 UDP 的端口来标识，首先 UDP 端口号可以是固定的，发送方只需要发送数据到指定端口即可。对于不同功能可以分配不同的端口，来区分不同功能的数据。对于标准功能，也可以预先分配端口号，实现该功能的程序可以申请该端口号，这样，就可以将数据包发往标准端口来实现此功能。

由于 UDP 协议是如此简单，所以不要指望 UDP 连接会像 TCP 连接那样可靠，它一点都不可靠，UDP 只负责尽力的转发数据包，但是却不会把错误的数据报重新发送，它会丢弃掉所有被破坏或者损坏的数据报，并且继续后面的传送，至于被丢弃的部分，发送端不知道，也不会被接收端要求重新发送；除此之外，UDP 不具备把乱序到达的数据报重新排列的功能(因为没有 TCP 头中包含的 TCP 序列号)，这样一来，UDP 便是完全不可靠的，因为你根本就无法保证你收到的数据是完整的。但是，UDP 协议的不可靠并不代表 UDP 是毫无用处的，恰恰相反，没有了和 TCP 一样的复杂头信息，各种设备处理 UDP 数据报的时间将会大大缩短，效率比 TCP 要高得多，你可以想象，你看 13 页书比看 4 页书需要用的时间谁会更多。由于 UDP 处理的这种高效性,UDP 往往被用于那些数据报不断出现的应用，比如 IP 电话或者实时视频会议，也被用于在路由器之间传输路由表更新信息、传送网络管理和监控数据等，DNS 也是使用 UDP 协议进行域名转换。

第三章 远程控制总体设计

3.1 Java socket 编程实现

3.1.1 什么是 Socket

网络上的两个程序通过一个双向的通讯连接实现数据的交换，这个双向链路的一端称为一个 Socket。Socket 通常用来实现客户方和服务方的连接。Socket 是 TCP/IP 协议的一个十分流行的编程界面，一个 Socket 由一个 IP 地址和一个端口号唯一确定。

但是，Socket 所支持的协议种类也不光 TCP/IP 一种，因此两者之间是没有必然联系的。在 Java 环境下，Socket 编程主要是指基于 TCP/IP 协议的网络编程。

3.1.2 Socket 通讯的过程

Server 端 Listen(监听)某个端口是否有连接请求，Client 端向 Server 端发出 Connect(连接)请求，Server 端向 Client 端发回 Accept（接受）消息。一个连接就建立起来了。Server 端和 Client 端都可以通过 Send，Write 等方法与对方通信。

对于一个功能齐全的 Socket，都要包含以下基本结构，其工作过程包含以下四个基本的步骤：

(1)创建 Socket；

(2)打开连接到 Socket 的输入/出流；
(3)按照一定的协议对 Socket 进行读/写操作；
(4)关闭 Socket.(在实际应用中不推荐)

3.1.3 创建 Socket

创建 Socket
java 在包 java.net 中提供了两个类 Socket 和 ServerSocket，分别用来表示双向连接的客户端和服务端。这是两个封装得非常好的类，使用很方便。其构造方法如下：

```
Socket(InetAddress address, int port);
Socket(InetAddress address, int port, boolean stream);
Socket(String host, int prot);
Socket(String host, int prot, boolean stream);
Socket(SocketImpl impl)
Socket(String host, int port, InetAddress localAddr, int localPort)
Socket(InetAddress address, int port, InetAddress localAddr, int localPort)
ServerSocket(int port);
ServerSocket(int port, int backlog);
ServerSocket(int port, int backlog, InetAddress bindAddr)
```

其中 address、host 和 port 分别是双向连接中另一方的 IP 地址、主机名和端 口号，stream 指明 socket 是流 socket 还是数据报 socket，localPort 表示本地主机的端口号，localAddr 和 bindAddr 是本地机器的地址（ServerSocket 的主机地址），impl 是 socket 的父类，既可以用来创建 serverSocket 又可 以用来创建 Socket。count 则表示服务端所能支持的最大连接数。

```
Socket client = new Socket("127.0.01.", 80);
ServerSocket server = new ServerSocket(80);
```

注意，在选择端口时，必须小心。每一个端口提供一种特定的服务，只有给出正确的端口,才 能获得相应的服务。0~1023 的端口号为系统所保留,例如 http 服务的端口号为 80,telnet 服务的端口号为 21,ftp 服务的端口号为 23, 所以我们在选择端口号时，最好选择一个大于 1023 的数以防止发生冲突。

在创建 socket 时如果发生错误，将产生 IOException，在程序中必须对之作出处理。所以在创建 Socket 或 ServerSocket 是必须捕获或抛出例外。

3.2 基于 TCP 协议的 Socket

服务器端首先声明一个 ServerSocket 对象并且指定端口号，然后调用 Serversocket 的 accept（）方法接收客户端的数据。accept（）方法在没有数据进行接收的处于堵塞状态。(Socketsocket=serversocket.accept()),一旦接收到数据，通过 inputstream 读取接收的数据。

客户端创建一个 Socket 对象，指定服务器端的 ip 地址和端口号（Socketsocket=newSocket("172.168.10.108",8080);）,通过 inputstream 读取数据，获取服务器发出的数据

（OutputStreamoutputstream=socket.getOutputStream()），最后将要发送的数据写入到 outputstream 即可进行 TCP 协议的 socket 数据传输。

3.3 基于 UDP 协议的 Socket

服务器端首先创建一个 DatagramSocket 对象，并且指点监听的端口。接下来创建一个空的 DatagramSocket 对象用于接收数据（bytedata[]=newbyte[1024;]DatagramSocketpacket=newDatagramSocket(data, data.length)），使用 DatagramSocket 的 receive 方法接收客户端发送的数据，receive()与 serversocket 的 accepet()类似，在没有数据进行接收的处于堵塞状态。

客户端也创建个 DatagramSocket 对象，并且指点监听的端口。接下来创建一个 InetAddress 对象，这个对象类似与一个网络的发送地址（InetAddressserveraddress=InetAddress.getByName（"172.168.1.120"））. 定义要发送的一个字符串，创建一个 DatagramPacket 对象，并制定要讲这个数据报包发送到网络的那个地址以及端口号，最后使用 DatagramSocket 的对象的 send()发送数据。

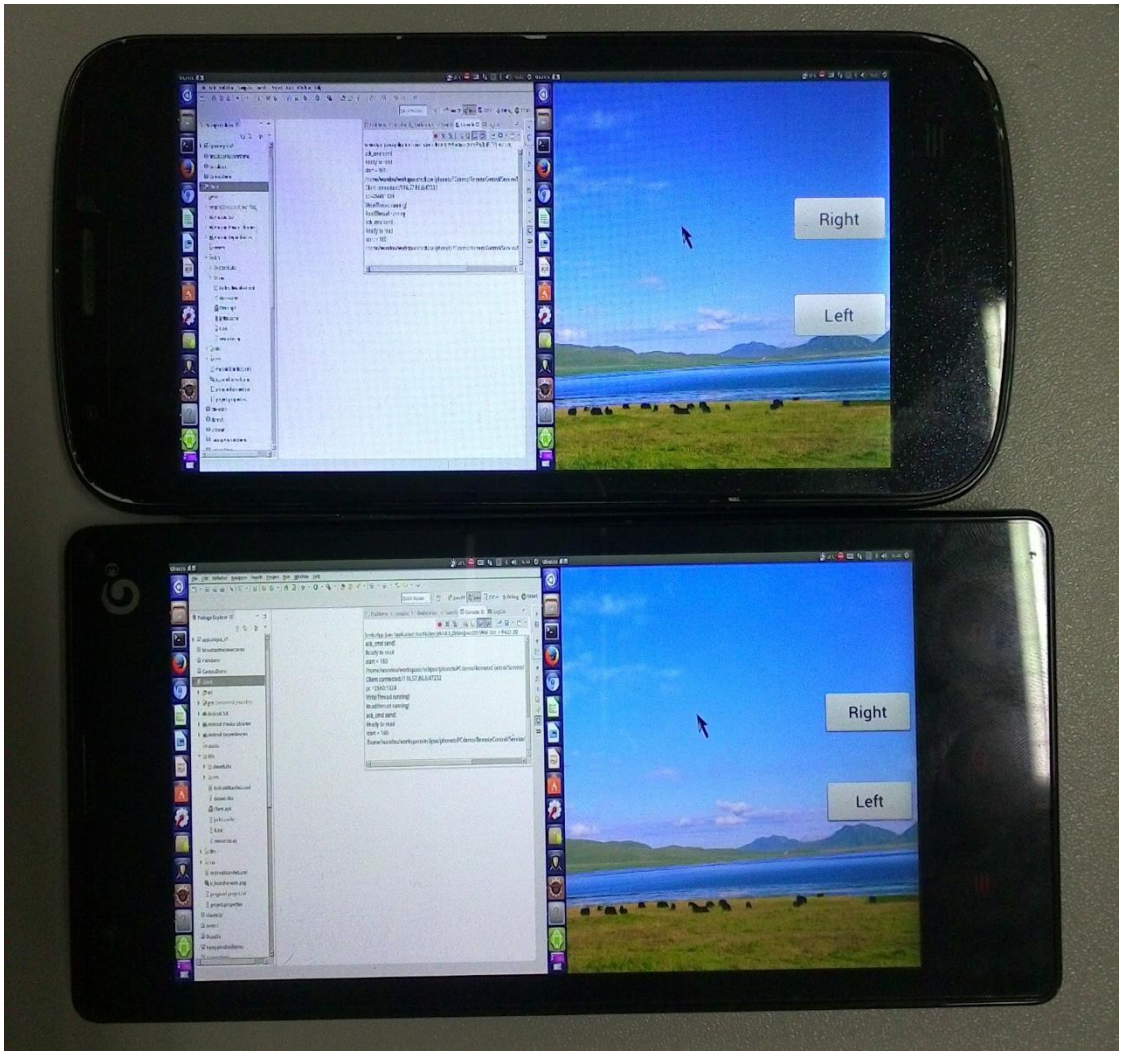
（Stringstr="hello";bytedata[]=str.getByte();DatagramPacketpacket=new DatagramPacket(data,data.length,serveraddress,4567);socket.send(packe t);）

3.4 服务器端设计

服务器端的界面效果图如图所示，打开后设定本地监听端口号，点击 Start 即搭建好了服务器端，服务器监听程序处于监听状态，直到接收到客户端的响应，服务器端是多任务的，可以同时为多个客户端提供服务。实现服务端功能的基本思路是这样的：在接收到客户端的响应建立连接后分别开启读线程和写线程，读线程负责读取客户端传递的鼠标指针位置并调用 Java 库中的 Robot 实现鼠标的左键点击，右键点击，双击和鼠标的移动；写线程抓去屏幕并将鼠标指针位置显示与屏幕图片，并调整图片大小以适应手机屏幕，调整图片大小以减轻数据负载，可以更加高效。

3.5 客户端设计

客户端的任务是将电脑屏幕实时映射到手机，登录界面需要输入 PC 主机的 IP 地址和同服务器端一致的端口号，在服务器端可以看到连接状态信息，连接成功即可在手机屏幕得到电脑的屏幕投影。客户端也开启读线程和写线程，读线程读取服务器端发送过来的图片并显示，写程序检测鼠标指针的位置并将位置信息发送出去，有 Left 和 Right 两个按键，分别代表鼠标的左右按键，检测按键事件并将按键信息发送给服务器端。至此简单的可视远程控制电脑程序就可以完成了。下图为实际测试两个手机端同时显示的双显示屏时的情况。



3.6 总体效果

上面展示的是用 TCP 协议写的数据传输 Socket，在实时性上有着稍微的延时，由于在测试的时候用 UDP 写的程序在数据报传输时图片不能很好的解析以至于在手机端显示乱码，所以放弃用 UDP 作为传输协议。这也正是 TCP 与 UDP 的区别的体现，TCP 保证数据传输的可靠性，数据流保持一定的顺序，所以在实时性上

有一定的缺陷但对图片这样的大文件传输不会出错；而 UDP 是不可靠传输的，数据包发送出去即可而不保证可靠性，所以在实时性上有较好的体验，而在图片传输则需要更复杂的解析代码，这里我就没有写了。

总体来说，无论是 UDP 版本还是 TCP 版本，都能顺利完成对电脑的操作，比如网页浏览、PPT 播放等都可以胜任，在实时性要求不高的情况下都能远程控制电脑，但由于 UDP 对图像传输的存在一定的复杂性，暂时没有实现。

第四章 总结

4.1 可以改进的方向

本次实验基本框架是搭建起来了，基本功能也实现了，但是可以优化和拓展的空间也很大，比如完善键盘输入功能，增加关机按钮，手机端的屏幕应该带有缩放功能以至于更加清晰的展示电脑屏幕，甚至可以调用传感器对鼠标进行控制等，有待后续开发。

4.2 心得体会

在截至时间之间能完成自己的设计很庆幸，虽然参考了网上的设计，但老老实实敲代码一步步调试，从最开始的 java/android 小白到稍微有点感觉，能够自己调试查找错误到最后看到自己想要的结果的兴奋，是一个很振奋人的过程。尝试用两种不同的协议实现相似功能，从中体会他们之间的差别，各自的优缺点，对 TCP、UDP 协议有了一个更加直观的认识，实践出真知，还需要继续努力。