个人信息及环境、分工

个人信息

杨雅儒，2017011071。

环境

- Windows 10
- gcc version 9.2.0
- JDK 1.8.0_51
- NDK android-ndk-r10e
- Android Studio 3.6.3
- 测试手机：Samsung Galaxy S7 edge (SM-G9350), Android 6.0.1

分工

由杨雅儒同学独立完成客户端所有内容，其间并未参考他人代码。而由于服务端同学退课，故使用学校的服务器进行测试。

实验原理

原理概述

考虑在 IPv6 网络上传输 IPv4 信息，只需要在同时具有 IPv4 和 IPv6 的边界路由器上进行头部的封装转换即可。

在本次实验中，简单地分为客户端和服务端，作为隧道的两头。假定客户端仅具有 IPv6 地址但希望访问 IPv4（也就是在客户端上完成了 IPv4 到 IPv6 的封装），服务端给每个客户端分配一个 IPv4 地址。客户端将包发送给服务端后，服务端进行解封装，并利用对应的 IPv4 进行访问。

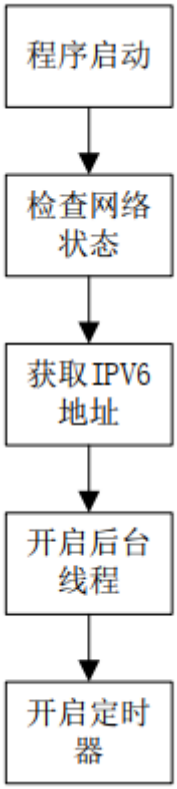
而我所做的工作主要为客户端，又分为了前后台两部分。考虑在安卓系统上，前台用于设置连接 IP、端口，显示传输流量、传输包数以及传输速率等信息，以及调用安卓的 VPN 接口等；后台用于使用 socket 连接服务器，并向其发送相关的数据包等。前后台之间通过命名管道进行连接。

另外本次实验的一大难点就是**如何在各种情况下合理地关闭和重启前后台**，而不致出现崩溃或者卡死的情况，在这一点上我下了很大的功夫，包括在断网、手动断开和重连、超时自动断开的情况下，都能合理、正确地关闭前后台所有线程。

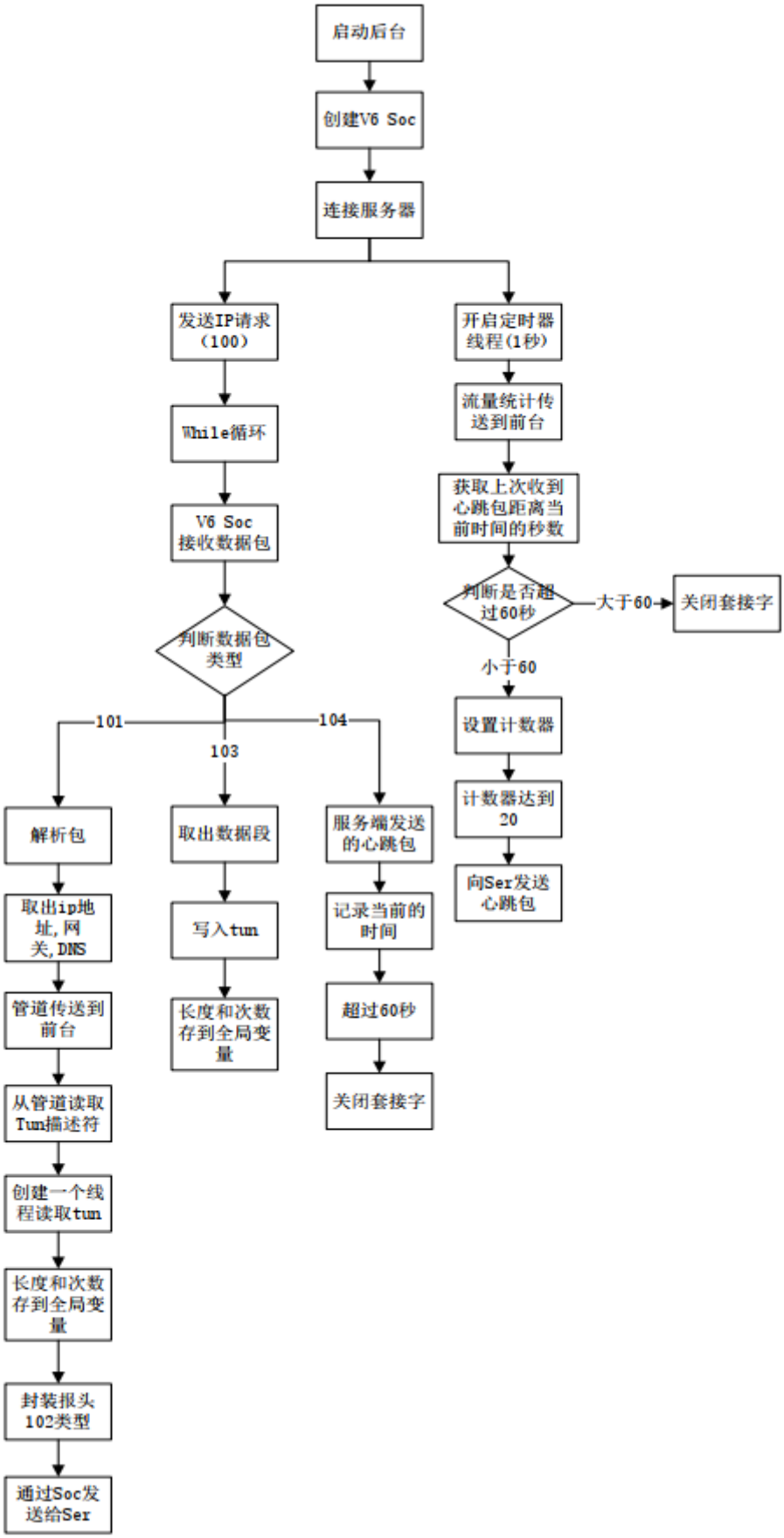
流程图

大致流程与实验指导书中的一致，**但是细节上有很多不同**，具体的细节可参见“实验内容”部分。

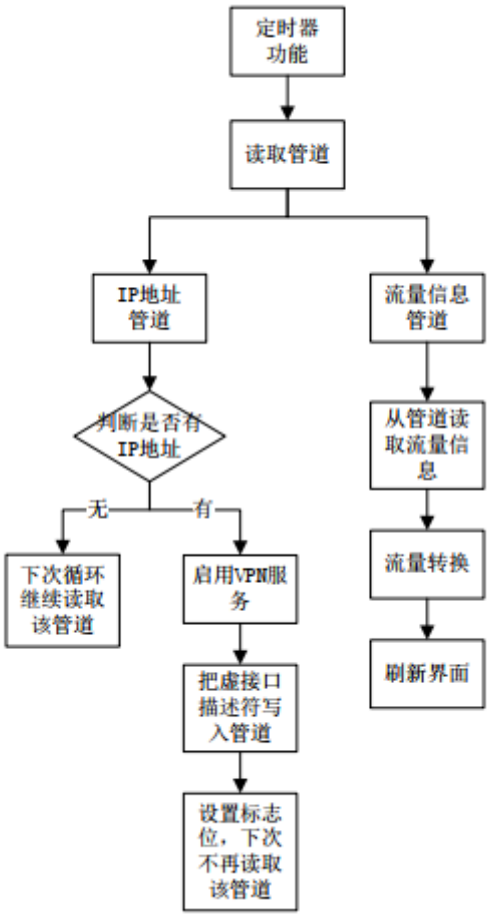
整体流程：



后台流程图：


启动后台
创建V6 Soc
连接服务器
发送IP请求（100）
While循环
V6 Soc接收数据包
判断数据包类型
101
103
104
解析包
取出ip地址,网关,DNS
管道传送到前台
从管道读取Tun描述符
创建一个线程读取tun
长度和次数存到全局变量
封装报头102类型
通过Soc发送给Ser
取出数据段
写入tun
长度和次数存到全局变量
服务端发送的心跳包
记录当前的时间
超过60秒
关闭套接字
开启定时器线程(1秒)
流量统计传送到前台
获取上次收到心跳包距离当前时间的秒数
判断是否超过60秒
大于60
关闭套接字
小于60
设置计数器
计数器达到20
向Ser发送心跳包

前台流程图：



实验内容

前后台管道通信格式

由于前后台之间的交互需要一定的规范，这里沿用客户端/服务端通信格式并对其进行了扩展，整体格式仍为：

```
struct Msg
{
    int length; //整个结构体的字节长度
    char type; //类型
    char data[4096]; //数据段
};
```

下面用 B 表示 Backend，用 F 表示 Frontend。

分为了五个 fifo：ipFifo 用于从后台向前台传输获取的 ip 等信息，tunFifo 用于从前台向后台传输 tun0 的文件描述符，statFifo 用于从后台向前台传输统计信息，debugFifo 用于从后台向前台发送调试信息，FBFifo 用于从前台向后台传输关闭的信息。

- type 101 B->F ipFifo：收到 ip 地址回应，通知前端打开 vpn，数据格式如 `13.8.0.2 0.0.0.0 202.38.120.242 8.8.8.8 202.106.0.20 13`，**注意最后多了一个数**，表示需要保护的 socketFd；
- type 105 F->B tunFifo：vpn 已开启，告知后端 tun0 的文件描述符，之后后端将开启新线程对其中内容进行处理。
- type 106 B->F statFifo: 数据格式如 `<该秒上传流量> <该秒上传包数> <该秒下载流量> <该秒下载包数>`，用字符串格式传输，流量单位为 B

- type 107 B->F debugFifo: 调试信息，数据使用字符串的形式发送调试信息。
- type 108 F->B FBFifo: 关闭后台。
- type 109 B->F debugFifo: 关闭 DebugRunnable，顺便利用 isRunning 关闭前台

前台

前台的线程和服务如下：

- 主线程：即 UI 线程，用于控制显示等；
- Backend 线程：含有 BackendHandler 和 BackendRunnable 两个类。用于向后台传入ipv6地址、端口以及各个命名管道的路径，以启用后台主线程；
- Debug 线程：含有 DebugHandler 和 DebugRunnable 两个类。用于不断读取 debugFifo 以获取调试信息（调试信息将使用 Log 进行显示），并且在接收到后台传来的关闭信息之后（即后台超时或者后台连接被服务器断开或因为网络情况不佳而断开等情况）进行前台关闭操作；
- Work 线程：含有 WorkHandler 和 WorkRunnable 两个类，用于不断获取后台传入的统计信息并通知主线程进行显示；
- MyVpnService 类：继承自 VpnService，用于启用 Vpn 服务。

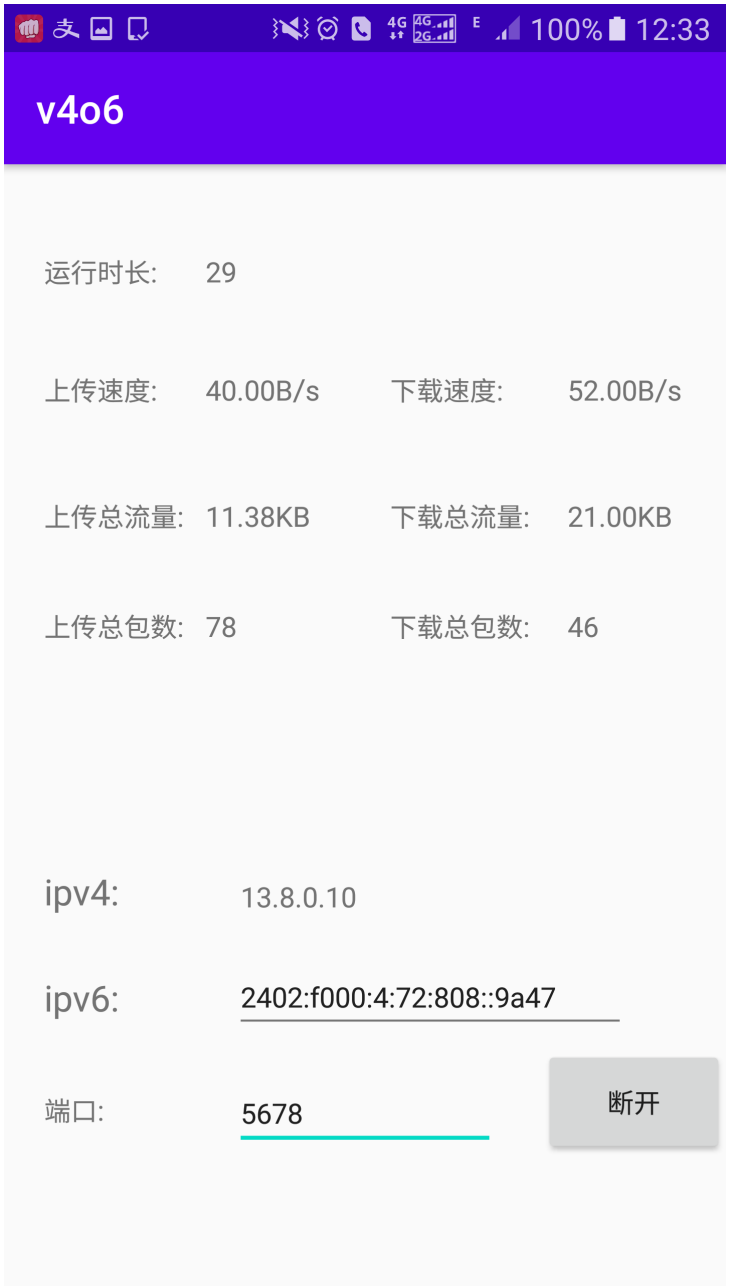
后台

- 主线程：用于发起 IP 地址请求并等待回应，连接成功之后，每当收到服务端的 103 访问响应则写入到 tun0，每当收到服务端的 104 心跳包则刷新心跳计数器。
- FBContoller 线程：用于前台对后台的控制，利用 FBFifo 实现，当前台向后台传输关闭消息时，后台开始对各线程进行清理。
- timerWorker 线程：定时器线程，用于每隔 1 秒进行心跳包发送判断（20秒向服务器发送一次心跳包）、超时判断、以及向前台发送统计信息。
- readTun 线程：用于读取 tun ，每当检查到有发往 tun0 的消息时，使用 socket 转发给服务器。

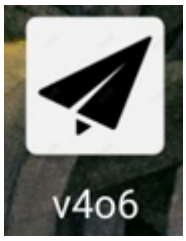
实验结果

本次实验构建了一个 app，可用于通过 ipv6 传输 ipv4 信息，并且**可以在较高鲁棒性的前提下处理各种情况的连接和断开功能。**

界面稍有些简陋，如下所示：



图标设计如下（向 shadowsocks 和 v2ray 致敬）：



代码运行命令

提供两种方式：

- 方式一：在手机上直接安装 `release/app-release.apk` 这个安装包进行使用。
- 方式二：使用相应版本的 Android Studio 打开项目，自行进行构建和运行。

遇到的问题及解决方案

VMWare 中无法连接 ipv6

需要更改NAT设置：编辑->虚拟网络编辑器->更改设置，选中NAT之后，点击 “NAT设置”，在其中勾选 “启用IPv6”。

最后重启即可。

Genymotion 中的 ipv6

Genymotion 上并没有找到合适的方法使用 ipv6。

于是最终决定在 VMWare 中使用 v6backend 这一 C++ 工程进行代码编写，而在真机上进行连通测试。

关于 stopService

需要清理 service 中的相关内容之后，调用 stopService 才会实际有效，才会实际执行 onDestroy。

解决办法是利用 bind 进行绑定，从 service 中留出接口供调用以关闭 ParcelFileDescriptor，之后再调用 stopService 即可成功。

关于存储路径

尝试在 C++ 中输出当前路径，得到 “/”；尝试在 Android 中使用 `getFilesDir().getAbsolutePath()` 输出内部存储路径，得到 “/data/user/0/com.yangyr17.v4o6/files”；尝试在 Android 中使用 `Environment.getExternalStorageDirectory().getAbsolutePath()` 输出外部存储路径，得到 “/storage/emulated/0”

关于管道

- 之前一直出问题，在 mknod 时永远都是返回 -1。

解决方案：**mknod 时如果该路径已经存在，则将会返回 -1**，故每次启动连接时应当在 Java 中将这些管道对应文件都先删去。

- 另外由于存储目录不能为外部存储，故由 `getFilesDir().getAbsolutePath()` 得到，这里在该目录下使用名为 fifo 的文件，例如 `/data/user/0/com.yangyr17.v4o6/files/fifo`。
- Java 端执行 `new FileInputStream(file)` 时卡死：这是因为 C++ 端对管道进行了 close！通讯过程中不能 close 掉!
- Java 端读 C++ 端写时，在 Java 端关闭管道之后，直到目前为止的管道消息都会消失，这是导致代码存在隐患的原因之一，解决方案是并不每次读都 close；
- C++ 端读 Java 端写时，在 Java 端关闭管道之后便不能再次打开了；
- Java 写 C++ 读时，仍然需要在 C++ 中使用 mkfifo!

关于文档中提供的服务器

需要连接 net.tsinghua.edu.cn 之后才能正常使用。

关于管道、socket等的读取和写入

一定注意传入的数量表示**最大读写字节数**，而并不能保证能够读写这么多。

关于 socket 的非阻塞模式

参考：https://blog.csdn.net/mayue_web/article/details/82873115

设置为非阻塞，目的是为了加入超时判断避免永久等待，如下：

```
int flags = fcntl(socketFd, F_GETFL, 0);
fcntl(socketFd, F_SETFL, flags|O_NONBLOCK);
```

读写时需判断是否有 `(EINTR == errno || EWOULDBLOCK == errno || EAGAIN == errno)`，这里一定注意 **errno** 是特有的变量，而不是 recv 或 send 的返回值。

在写 socket 时便可以如下：

```
// 模拟阻塞行为，并加入对 timeout 的判断
bool writeSingleToSocket(int socketFd, char *data, int len) {
    int cnt = 0;
    char ttt[100];
    while (len > cnt) {
        // 判断 timeout
        pthread_mutex_lock(&beatTimerMutex);
        if (timeout) {
            pthread_mutex_unlock(&beatTimerMutex);
            return false;
        }
        pthread_mutex_unlock(&beatTimerMutex);

        int tmp = send(socketFd, data + cnt, len - cnt, 0);
        if (tmp > 0) {
            cnt += tmp;
        } else if (errno == EINTR || errno == EWOULDBLOCK || errno == EAGAIN) {
            sprintf(ttt, "重写 errno: %d", errno);
            writeDebugMessage(ttt);
            continue;
        } else {
            return false;
        }
    }
    return true;
}
```

关于互斥锁

在后台中，各个线程需要共享变量，这个时候不能直接使用，**需要相当小心地使用 pthread_mutex_t 类型的互斥锁来避免冲突**，一定需要注意加锁之后在各个情况下都能正确解锁。