Ipv4 over Ipv6 隧道协议实验客户端报告

计 41 张盛豪 2014011450

计 41 李永斌 2014011442

计 41 陈伟楠 2014011452

2017-05-23

目录

1 实验目的

- 掌握 Android 下应用程序开发环境的搭建和使用
- 掌握 IPv4 over IPv6 隧道的工作原理

2 客户端实验要求

在安卓设备上实现一个 4over6 隧道系统的客户端程序

- 实现安卓界面程序，显示隧道报文收发状态（java 语言）；
- 启用安卓 VPN 服务（java 语言）；
- 实现底层通信程序，对 4over6 隧道系统控制消息和数据消息的处理（C 语言）。

3 实验内容

3.1 前台是 java 语言的显示界面

- 进行网络检测并获取上联物理接口 IPV6 地址；
- 启动后台线程；
- 开启定时器刷新界面；
- 界面显示网络状态；
- 开启安卓 VPN 服务。

3.2 后台是 C 语言客户端与 4over6 隧道服务器之间的数据交互

- 连接服务器；
- 获取下联虚接口 IPV4 地址并通过管道传到前台；
- 获取前台传送到后台的虚接口描述符；
- 读写虚接口；
- 对数据进行解封装；
- 通过 IPV6 套接字与 4over6 隧道服务器进行数据交互；
- 实现保活机制，定时给服务器发送 keeplive 消息。

4 实验原理

4.1 面向 Android 终端的隧道原理

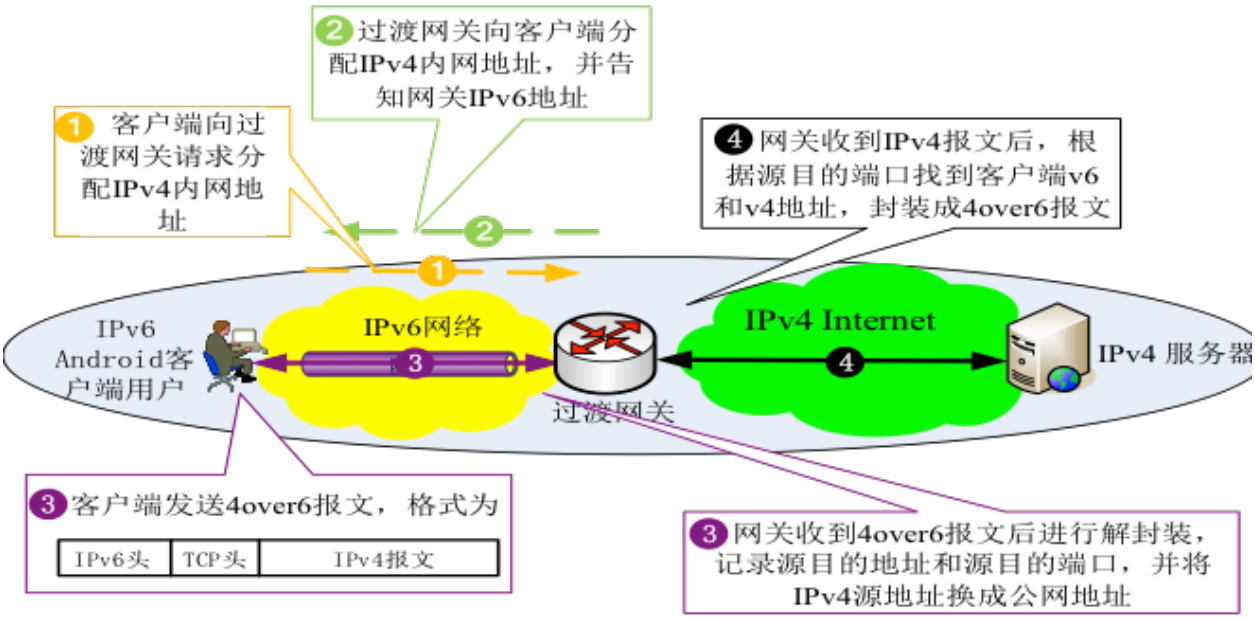

图 1: 4over6 隧道原理

在 4over6 隧道中，客户端首先向过渡网关请求分配 IPv4 内网地址；过渡网关分配 IPv4 内网地址，并提供对应的 IPv6 网络地址；接着，安卓客户端发送 4over6 报文，过渡网关接受报文并进行分析，得到源地址和目的地址，将 IPv4 地址转化为公网地址，发送到公网之中；当过渡网关收到公网的 IPv4 报文之后，根据记录好的映射关系，重新封装成 4over6 报文，发给对应的内网用户，完成数据的转发和接受。

本实验中分别完成了过渡网关和客户端的功能，本报告主要阐述客户端的相关实现。

客户端用户处于 IPV6 网络环境，通过过渡网关完成 IPV4 网络的访问，过渡网关横跨 IPV4 和 IPV6，提供地址转换和数据包的分发。

4.2 VPN Service 原理

在客户端实现过程中，使用了 VPNService API，打开 VPN 服务后，Android 系统通过 iptables 使用 NAT 将所有数据包转发到 TUN 虚拟网络设备，通过mInterface.getFd();可以获取到虚接口描述符，从而通过读虚接口获取系统的 IP 数据包，再将 IPV4 数据包封装经 IPV6 socket 转发给服务器端即可；服务器端根据网络请求信息完成访问，并将结果通过 IPV6 socket 发回，后台解析取出 IPV4 数据包，写入虚接口以实现数据接收。

5 具体实现

客户端实现主要分前台和后台，前台是 Java 实现的 Android 客户端显示界面，后台主要是 C++ 实现的客户端与服务器端的数据交互。前后台通过读写管道以及 JNI 函数调用实现交互。总体流程如下：

图 2: 总体流程图

5.1 前端实现内容

5.1.1 前端流程及完成的工作

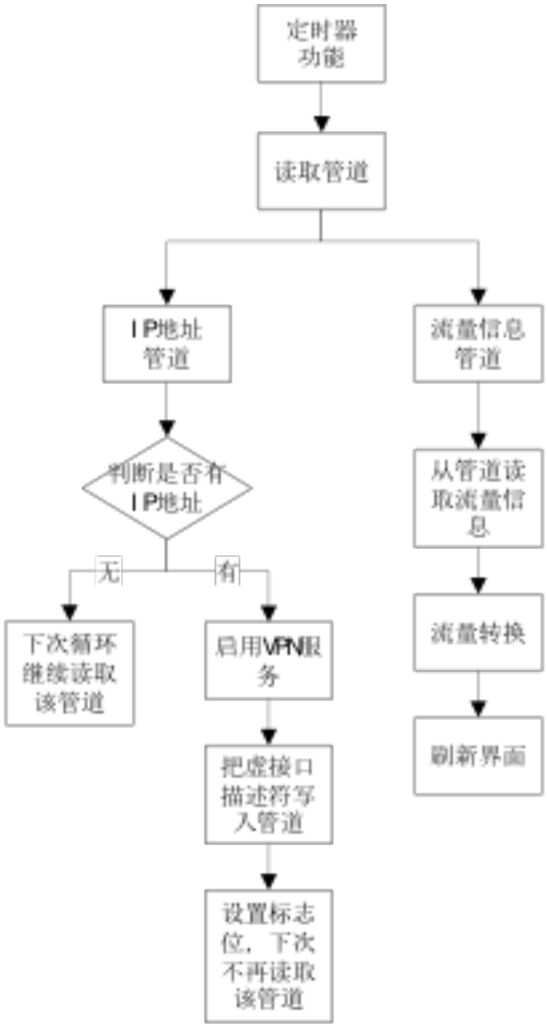

图 3: 前端详细流程

前台流程如上图所示，主要完成以下操作

- 开启定时器之前，创建一个读取 IP 信息管道的全局标志位 flag，默认置 0;
- 开始读取管道，首先读取 IP 信息管道，判断是否有后台传送来的 IP 等信息;
- 假如没有，下次循环继续读取;
- 有 IP 信息，就启用安卓 VPN 服务 (此部分在后面有详细解释);
- 把获取到的安卓虚接口描述符写入管道传到后台;
- 把 flag 置 1，下次循环不再读取该 IP 信息管道;
- 读取流量信息管道;
- 从管道读取后台传来的实时流量信息;
- 把流量信息进行格式转换;

- 显示到界面；
- 界面显示的信息有运行时长、上传和下载速度、上传总流量和包数、下载总流量和包数、下联虚接口 V4 地址、上联物理接口 IPV6 地址。

5.1.2 具体实现

5.1.2.1 界面

主界面采用了 ScrollView 嵌套 LinearLayout 的布局方式，界面中中有 2 个输入框，以及一个 Button，分别用于输入服务器端的 IPV6 地址，端口号，以及点击按钮链接 VPN。且有一个文本框用于显示当前 IPV6 地址（若无，提示无 IPV6 网络访问权限）链接 VPN 之后，IPV6 地址和端口输入框消失，显示一个 TextView（用于显示运行时长、上传和下载速度、上传总流量和包数、下载总流量和包数、下联虚接口 V4 地址、上联物理接口 IPV6 地址等信息）和断开连接按钮。


20:37
0.48 K/s
80
IPv4_over_IPv6
Server Address:
2402:f000:5:8601:942c:3463:c810:6147
Server Port:
6666
Local IPv6:
无IPv6访问权限，请检查网络
连接VPN
无IPv6访问权限，请检查网络, 4s后重试...

图 4：未连接界面


20:44
1.29 K/s
82
IPv4_over_IPv6
IPV4: 10.0.0.3
Router: 0.0.0.0
DNS: 8.8.8.8 202.38.120.242 202.106.0.20
已运行: 00:00:00
已发送: 407 B/2 个数据包
已接收: 417 B/1个数据包
共出入: 824 B/3个数据包
上传: 0.40 KB/s
下载: 0.41 KB/s
上联V6: 2402:f000:5:8601:8e23:e6e3:b7:3110
断开VPN
Top VPN is connecting...

图 5: 连接VPN 后界面

5.1.2.2 UI 主线程

- 客户端开启后，即检查当前网络环境是否支持 IPV6 访问，若不可访问 IPV6 网络则用户点击【链接 VPN】按钮无效。检查代码如下：

```
static String getIPv6Address(Context context) {
    if(! isWIFIConnected(context)) {
        return null;
    }
    try {
        final Enumeration<NetworkInterface> e =
           NetworkInterface.getNetworkInterfaces();
        while (e.hasMoreElements()) {
            final NetworkInterface networkInterface = e.nextElement();
            for (Enumeration<InetAddress> enumAddress =
                networkInterface.getInetAddresses();
                 enumAddress.hasMoreElements(); ) {
                InetAddress inetAddress = enumAddress.nextElement();
                if (!inetAddress.isLoopbackAddress() &&
                    !inetAddress.isLinkLocalAddress()) {
                    return inetAddress.getHostAddress();
                }
            }
        }
    } catch (SocketException e) {
        Log.e("NET", "无法获取IPV6地址");
    }
    return null;
}
```

- 为【链接 VPN】按钮注册监听服务，若可访问 IPV6 网络（即有 IPV6 地址），则用户点击按钮后，启动 VPN 服务，并将用户填入的服务器 IPV6 地址和端口号通过 Intent 传入
- 启动 VPN 服务后，主界面定时刷新，通过 JNI java 调用 C 函数方式读取流量收发信息并显示在主界面。

5.1.2.3 VPNService

继承一个 VpnService 的类，启动后，先读取 MainActivite 中传入的 Intent，获取服务器端的 IPV6 地址和端口号，通过 JNI java 调用 C 函数的方式将该数据传给 C 后台，C 后台自动与服务器端建立 IPV6 Socket，然后 VPNService 循环查询 C 后台是否获取到服务器发回的 101 数据信息，得到 C 后台传入的 IP 地址，DNS，路由，以及 IPV6 Socket 标识信息，据此初始化 VPN 服务并启动，获取到 TUN 虚接口的文件描述符后，通过 JNI 函数调用传给 C 后台，至此前端的任务完成。

```
// 2. 开始读取管道，首先读取IP信息管道，判断是否有后台传送来的IP等信息
// 3. 假如没有，下次循环继续读取；
```

```
while(!isGet_ip()) {
}
// 4. 有IP信息，就启用安卓VPN服务
String ip_response = ip_info();
Log.e(TAG, "GET IP " + ip_response);
String[] parameterArray = ip_response.split(" ");
if (parameterArray.length <= 5) {
    throw new IllegalStateException("Wrong IP response");
}

// 从服务器端读到的IP数据
ipv4Addr = parameterArray[0];
router = parameterArray[1];
dns1 = parameterArray[2];
dns2 = parameterArray[3];
dns3 = parameterArray[4];
// sockrt 描述符
String sockfd = parameterArray[5];
Builder builder = new Builder();
builder.setMtu(1500);
builder.addAddress(ipv4Addr, 32);
builder.addRoute(router, 0); // router is "0.0.0.0" by default
builder.addDnsServer(dns1);
builder.addDnsServer(dns2);
builder.addDnsServer(dns3);
builder.setSession("Top Vpn");
try {
    mInterface = builder.establish();
}catch (Exception e) {
    e.printStackTrace();
    Log.e(TAG,"Fatal error: " + e.toString());
    return;
}
// 5. 把获取到的安卓虚接口描述符写入管道传到后台
int fd = c
send_fd(fd, PIPE_DIR);
if (!protect(Integer.parseInt(sockfd))) {
    throw new IllegalStateException("Cannot protect the mTunnel");
}
start = true;
Log.e(TAG, "configure: end");
```

5.2 后台实现内容

5.2.1 后台实现流程及内容

1. 创建 IPV6 套接字;
2. 连接 4over6 隧道服务器;
3. 开启定时器线程（间隔 1 秒）:
    1. 读写虚接口的流量信息写入管道;
    2. 获取上次收到心跳包距离当前时间的秒数 S;
    3. 假如 S 大于 60，说明连接超时，就关闭套接字;
    4. S 小于 60 就每隔 20 秒给服务器发送一次心跳包。
4. 发送消息类型为 100 的 IP 请求消息;
5. while 循环中接收服务器发送来的消息，并对消息类型进行判断;
    1. 101 类型（IP 响应）:
        1. 取出数据段，解析出 IP 地址，路由，DNS;
        2. 把解析到的 IP 地址，路由，DNS 写入管道;
        3. 从管道读取前台传送来的虚接口文件描述符;
    2. 创建读取虚接口线程:
        1. 持续读取虚接口;
        2. 记录读取的长度和次数;
        3. 封装 102(上网请求）类型的报头;
        4. 通过 IPV6 套接字发送给 4over6 隧道服务器。
    3. 103 类型（上网应答）:
        1. 取出数据部分;
        2. 写入虚接口;
        3. 存下写入长度和写入次数。
    4. 104 类型（心跳包）:
        1. 记录当前时间到一个全局变量。

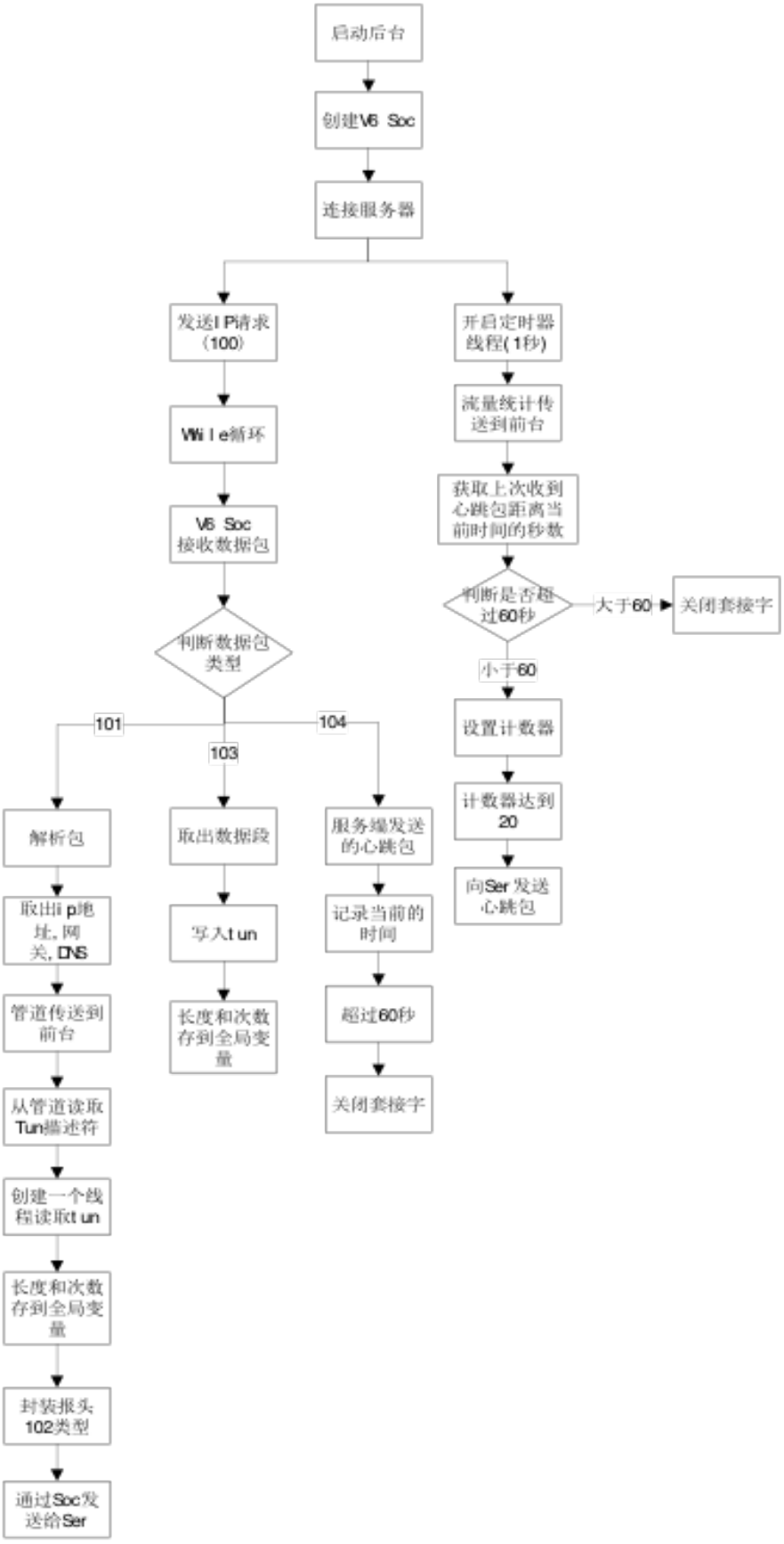
启动后台
创建V6 Soc
连接服务器
发送IP请求(100)
Wile循环
V6 Soc接收数据包
判断数据包类型
101
103
104
解析包
取出ip地址,网关,DNS
管道传送到前台
从管道读取Tun描述符
创建一个线程读取tun
长度和次数存到全局变量
封装报头102类型
通过Soc发送给Ser
取出数据段
写入tun
长度和次数存到全局变量
服务端发送的心跳包
记录当前的时间
超过60秒
关闭套接字
开启定时器线程(1秒)
流量统计传送到前台
获取上次收到心跳包距离当前时间的秒数
判断是否超过60秒
大于60
关闭套接字
小于60
设置计数器
计数器达到20
向Ser发送心跳包

图 6：后台实现流程

5.2.2 具体实现

5.2.2.1 与服务器建立 Socket 连接

后台使用 C++ Socket 编程实现，用户点击【连接 VPN】按钮之后即启动 MyVPNService 服务，同时调用 C 后台的start_VPN函数，该函数是后台主函数，MyVPNService 服务通过public native int send_addr_port(String addr, int port);函数向 C 后台传递服务器端 IPV6 地址和端口，后台检测到取得 IPV6 地址信息后即完成与服务器的 Socket 连接及绑定。

```
if ((sockfd = socket(AF_INET6, SOCK_STREAM, 0)) < 0) {
    LOGE("can't create socket");
}
server.sin6_family = AF_INET6;
server.sin6_port = htons(SERVER_PORT);
Inet_pton(AF_INET6, SERVER_IPV6, &server.sin6_addr);
int temp;
if ((temp  = connect(sockfd, (struct sockaddr *) &server, sizeof(server)) )== -1){
    __android_log_print(ANDROID_LOG_ERROR, TAG, "can't access server %s",
        strerror(errno));
    return -1;
}
```

后台主线程中新建了 3 个线程manage_data，readTun以及send_heart，分别用于处理数据请求，读取虚接口并发送 102 上网请求，以及发送定时心跳包

5.2.2.2 manage_data 线程

在实验中我们重新定义了 Message 结构体，每次从 socket 读取数据时均先读取Msg_Hdr，然后根据 length 字段长度读取相应的 data 字段

```
struct Msg_Hdr {
    uint32_t length; // payload 长度,不包括type，注意协议切割
    char type; //
};

struct Msg{
    struct Msg_Hdr hdr;
    char data[MAX_MESSAGE_LENGTH];
};
```

只要与服务器 socket 连接保持，则manage_data线程循环运行，在manage_data线程中主要完成以下操作：

- 判断心跳包是否超时，超时则修改对应状态，并关闭 socket，通知 java 前端 VPN 已断开连接
- 从 socket 中读取信息结构体的 Msg_Hdr 部分数据，根据读取到的 type 和 length 字段决定是否继续读取 data

- 根据 type 字段做对应的处理

```
n = read(fd, &msg, needbs);
if(n < 0) {
    LOGE("read sockfd %d error: %s \n",fd,strerror(errno));
    kill_myself();
    return -1;
}
else if(n == 0) {
    LOGE("recv 0 byte from server, close sockfd %d \n",fd);
    kill_myself();
    return -1;
}
else if(n == needbs){
    process_payload:
    char* ipv4_payload = msg.data;
    if(msg.hdr.type != 100 && msg.hdr.type != 104) {
        n = read(fd, ipv4_payload, msg.hdr.length);
        if(n != msg.hdr.length) {
            LOGE("read payload error, need %d byte, read x byte\n",msg.hdr.length);
            if(n <= 0) {
                LOGE("读取data出错，关闭");
                kill_myself();
                return -1;
            }
        }
        while(n < msg.hdr.length)
            n += read(fd, ipv4_payload + n, msg.hdr.length-n);
    }

    switch(msg.hdr.type){
        case 101:
            LOGE("get 101");
            recv_ipv4_addr(&msg);
            break;
        case 103:
            LOGE("get 103");
            recv_ipv4_packet(&msg);
            break;
        case 104:
            // 心跳包,记录接收时间
            s = time(NULL);
            LOGE("%s %ld", "收到心跳包104", s);
            break;
        default:
```

```
            return -1;
        }
    }
    else {// 读到长度小于头长度说明可能出错(也有可能粘包,继续读取)
        while (n < needbs)
            n += read(fd, ((char*)&msg) + n , needbs-n);
        goto process_payload;
    }
```

- 读取到 103 上网回应包recv_ipv4_packet: 当收到 103 数据包，就代表收到了服务器转发的数据，这时候直接将其 data 字段写入 tun 虚接口之中即可，并更新统计信息。

```
int write_tun(char* payload, uint32_t len) {
    // 写虚接口，收到信息，写入虚接口
    byte_in += len + 8;
    packet_in += 1;
    total_byte += len + 8;
    total_packet += 1;
    Write_nByte(tun_des, payload, len);
    return 0;
}

void recv_ipv4_packet(Msg* msg) {
    write_tun(msg->data, msg->hdr.length);
    debugPacket_recv(msg, msg->hdr.length);
}
```

- 收到 104 心跳包时，更新收到心跳包的实际即可

6 实验结果及分析

实验测试阶段，我们使用魅族和华为安卓手机进行了测试，网络环境为宿舍的 Tsinghua 无线网（可访问 IPV6）以及实验室的 DIVI 网络，开始 VPN 后，可以正常连接，测试了多种网络传输对象和环境：

- 浏览器打开网页，网页加载速度流畅
- 斗鱼直播，视频播放流畅
- 微信，qq 文字消息，表情包消息等发送接收正常

经过测试可以验证实现基本正确，且我们也测试过其稳定性，在 IPV6 网络稳定条件下，VPN 服务不会异常中断，且手动断开 VPN 后可重新点击连接 VPN 按钮正常连接。

7 遇到的问题

7.1 连接 VPN 之后无法与服务器之间进行通信

C 后台请求到 IPV4 地址信息之后，前端建立好 VPN 服务之后读取虚接口向服务器发送数据流量信息时服务器收不到相应的数据包，经过检查发现在 VPN 的建立过程中未对 C 后台 Socket 数据进行保护，导致后台与服务端之间的 ipv6 socket 链接也转发到了 VPN 的虚接口，这样就导致 102 的上网请求数据包无法发送给服务器，修改方案是利用 VPNService 类里的 protect 方法保护自己的 socket。

```
if (!protect(Integer.parseInt(sockfd))) {
    throw new IllegalStateException("Cannot protect the mTunnel");
}
```

如上所示，其中socketfd即为 C 后台与服务端链接时的 Socket，通过 JNI 或者管道方式从 C 后台读取即可。

7.2 数据包读取不完整

在一开始的实现中有时候会发现会读取到非预设类型的报文，也就是说这些报文是读取不完整的，在传输过程中被截断或者读取时未读取完整，导致数据混乱，查阅相关资料后与服务器端一同重新定义 Message 结构体（见【manage_data 线程】一节），并且在读取数据包时确保每次读取sizeof(Msg_Hdr)个字节或者msg_hdr.length个字节 data 字段，从而避免粘包现象。

```
size_t needbs = sizeof(struct Msg_Hdr);
n = read(fd, &msg, needbs);
if(n < 0) {
    LOGE("read sockfd %d error: %s \n",fd,strerror(errno));
    kill_myself();
    return -1;
}
else if(n == 0) {
    LOGE("recv 0 byte from server, close sockfd %d \n",fd);
    kill_myself();
    return -1;
}
else if(n == needbs){
    process_payload:
    if(msg.hdr.type != 100 && msg.hdr.type != 104) {
        n = read(fd, ipv4_payload, msg.hdr.length);
        if(n != msg.hdr.length) {
            /*做出对应处理*/
        }
        while(n < msg.hdr.length)
```

```
            n += read(fd, ipv4_payload + n, msg.hdr.length-n);
    }
    /*
    * 处理读取到的数据
    */
}
else {// 读到长度小于头长度说明可能出错(也有可能粘包,继续读取)
    while (n < needbs)
        n += read(fd, ((char*)&msg) + n , needbs-n);
    goto process_payload;
}
```

7.3 异常退出问题

在测试时发现有时候应用程序会突然崩掉，直接被系统 kill 掉，输出调试信息后发现总是会报如下错误

```
05-10 19:28:45.915 24188-24188/com.example.ipv4_over_ipv6 A/libc: Fatal signal 5
   (SIGTRAP), code 1 in tid 24188 (.ipv4_over_ipv6)
```

Google 之后发现是因为通过 JNI 方式调用的 C 后台的部分函数忘记了 return，而编译 app 时并不会检查出这种错误，而运行时检测出该问题之后则会直接 kill 整个程序，解决方法是添加对应的返回值语句即可。

8 实验心得体会

在这次实验中，我们掌握了 Android 下应用程序开发环境的搭建和使用，理解了 IPV4 over IPv6 的原理，掌握了 Android 下 VPN 的原理及实现，掌握了 JNI 调用原理，并实现了一个测试通过，效果良好，性能稳定的客户端。除此之外，我们对于原理课上所讲的 IPV4 over IPv6 又有了更深的理解和体会，对 IPV6 数据包和 IPV4 数据包的格式和特点有了更深入的认识，加深了对 socket 的理解，特别是对数据包的读取粘包问题的处理，让我们对网络包的传输有了更深入的认识。

感谢老师和助教的悉心指导。