网络技术挑战赛

4over6 原理验证系统
开发文档

Contents

第 1 章. 项目简介

1.1 项目概述

IPV4 over IPV6，简称“4over6”是 IPV4 向 IPV6 发展进程中，向纯 IPV6 主干网过渡提出的一种新技术，可以最大程度地继承基于 IPV4 网络和应用，实现 IPV4 向 IPV6 平滑的过渡。该项目通过实现 IPV4 over IPV6 隧道最小原型验证系统，以实现在安卓手机上，通过 IPv6 网络访问 IPv4 网站的功能。

1.2 客户端项目概述

在安卓设备上实现一个 4over6 隧道系统的客户端程序，内容如下：

- 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)

1.3 服务器项目概述

在 linux 系统下，实现 4over6 隧道系统服务端程序，内容如下：

- 实现服务端与客户端之间控制通道的建立与维护
- 实现对客户端网络接口的配置
- 实现对 4over6 隧道系统数据报文的封装和解封装

第 2 章. 项目原理

2.1 面向 Android 终端的隧道原理

IPv6 网络中的用户在希望访问 IPv4 服务时，将 IPv4 包封装在 IPv6 包中发送给一台能够直接访问 IPv4 服务的服务器。该服务器将进行实际 IPv4 资源的访问，然后再将得到的 IPv4 响应封装在 IPv6 包中发送回用户。为了能够通过 IPv4 地址将 IPv4 响应返回给相应的用户，IPv4 over IPv6 服务器需要维护一个 IPv4 和 IPv6 地址的映射表并为需要 IPv4 over IPv6 服务的用户分配 IPv4 地址。本次项目中的 IPV4 over IPV6 隧道原理如下图所示：

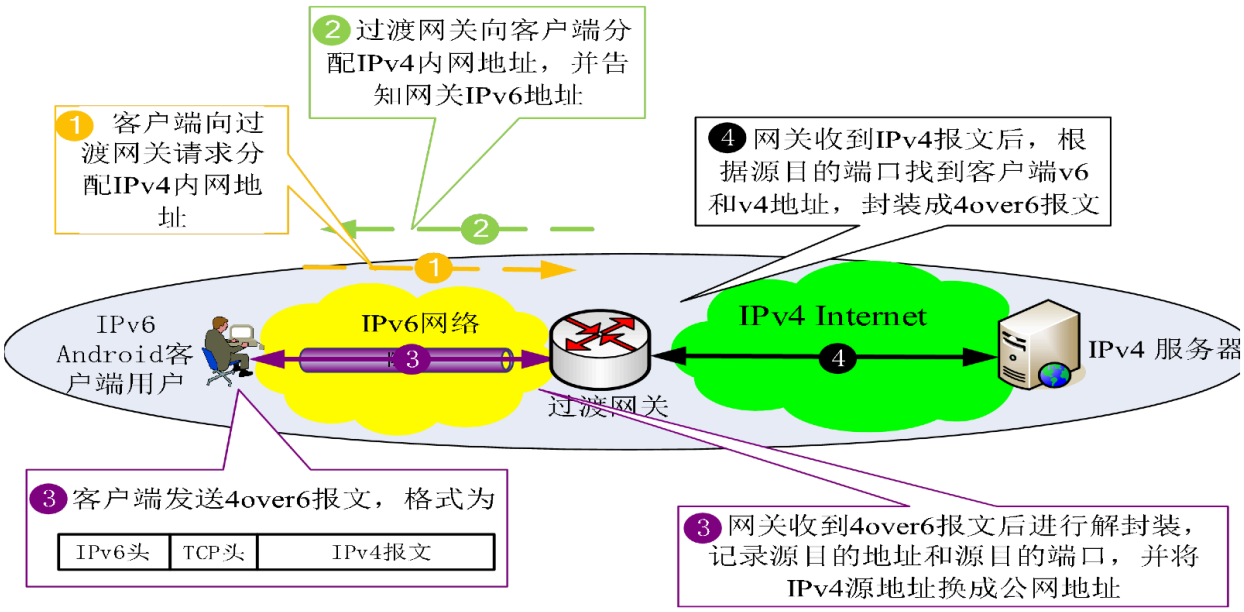

Figure 2.1: 面向 Android 终端的隧道原理

在本次项目中，我们用到的隧道原理主要是面向 Android 终端的隧道原理，在面向

Android 终端的隧道原理中，大致流程为：安卓客户端首先向过渡网关请求分配 IPv4 内网地址；过渡网关向客户端分配 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 IPv4 over IPv6 报文；网关收到报文后解封装，记录源目的地址和源目的端口，并将 IPv4 源地址换成公网地址。当过渡网关收到来自 IPv4 服务器的报文之后，会根据之前记录的映射关系，重新封装成 IPv4 over IPv6 报文，发送给对应的内网用户，这样就完成了数据的的转发和接收，用户便可以成功接收到 IPv4 数据，从而实现通过 IPv6 网络访问 IPv4 的功能。

2.2 项目网络拓扑

本次项目中用到的网络拓扑如下图所示：

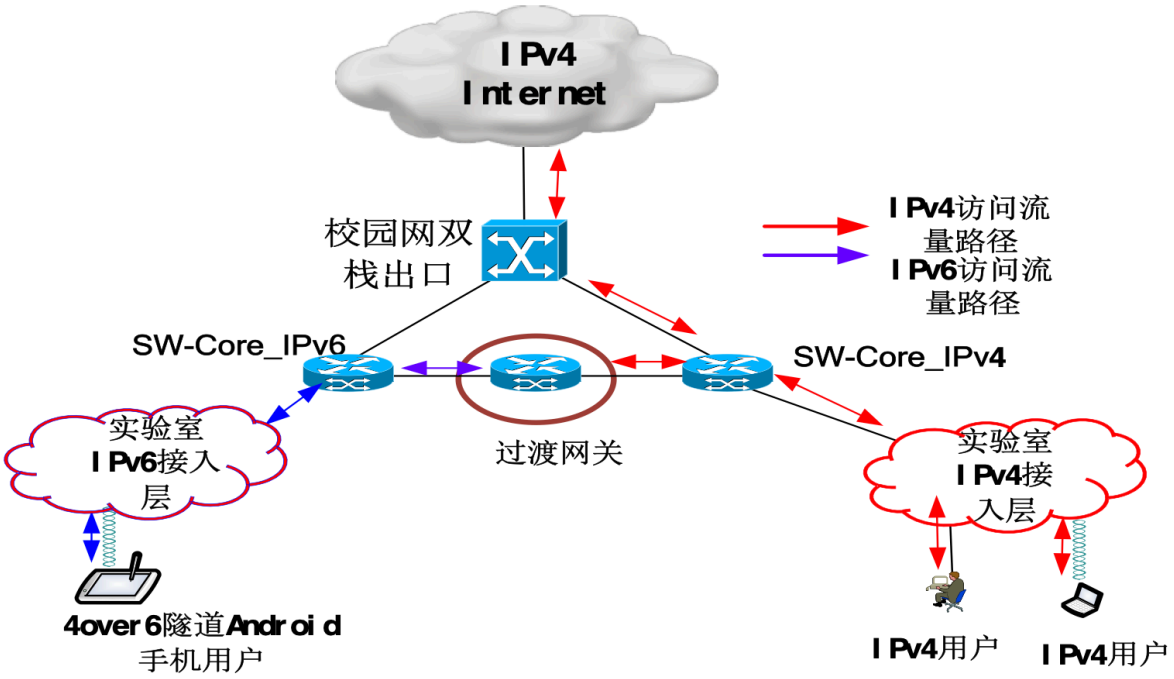

Figure 2.2: 项目网络拓扑图

2.3 安卓 VPN Service 原理

VPN Service 是一个 android 自带的 vpn 服务类，它可以在你不 root 手机的情况下，实现对你手机流量控制。可以通过设置对指定应用流量的拦截，也可以做全局操

作。如果你的应用是想要拦截网络，或者想要，获取网络数据，转发网络数据，就可以通过这个去做。VPN Service 会监控所有的网络进程，并可以进行 IP 隧道处理。当我们发送数据时，可以使用 IPv6 包装要发送的报文并发送给网关。当我们接收数据时，可以将 IPv4 的报文从数据包中剥离出来。本次项目中，安卓客户端需要完成 IPv4 over IPv6 报文的封装和解封装，我们采用了 VPN Service 来进行实现。VPN Service 是安卓的一套 API 接口，方便编程人员创建 VPN 服务。打开服务之后，安卓系统将所有的应用程序发送的 IP 包都根据 iptables 使用 NAT 转发到 TUN，其端口为 tun0。当打开 VPN Service 之后，我们可以获取 tun0 的文件描述符，这样就可以读取或写入数据以实现发送或者接收数据。

Android 设备上，已经使用了 VpnService 框架，建立起了一条从设备到远端的 VPN 链接

- 应用程序使用 socket，将相应的数据包发送到真实的网络设备上
- Android 系统通过 iptables，使用 NAT，将所有的数据包转发到 TUN 虚拟网络设备上去，端口是 tun0
- VPN 程序通过打开/dev/tun 设备，并读取该设备上的数据，可以获得所有转发到 TUN 虚拟网络设备上的 IP 包
- VPN 数据可以做一些处理，然后将处理过后的数据包，通过真实的网络设备发送出去

2.4 NDK 和 JNI

在本次项目中，由于经常要和以字节为单位的数据打交道，并且需要精细地管理内存，因此我们采用 C 来实现 IPv4 over IPv6 的核心功能。要在以 Java 为语言的安卓环境中使用 C 来进行编程，因此我们需要使用 JNI 和 NDK。

NDK 是 Native Develop Kit 的含义，从含义很容易理解，本地开发。大家都知道，Android 开发语言是 Java，不过我们也知道，Android 是基于 Linux 的，其核心库很多都是 C/C++ 的，比如 Webkit 等。那么 NDK 的作用，就是 Google 为了提供给开发者一个在 Java 中调用 C/C++ 代码的一个工作。NDK 本身其实就是一个交叉工作链，包含了 Android 上的一些库文件，然后，NDK 为了方便使用，提供了一些脚本，使得更容易的编译 C/C++ 代码。总之，在 Android 的 SDK 之外，有一个工具就是 NDK，用于进行 C/C++ 的开发。一般情况，是用 NDK 工具把 C/C++ 编译为.co 文件，然后在 Java 中调用。

JNI，全称为 Java Native Interface，即 Java 本地接口，JNI 是 Java 调用 Native 语言的一种特性。通过 JNI 可以使得 Java 与 C/C++ 机型交互。即可以在 Java 代码中调用 C/C++ 等语言的代码或者在 C/C++ 代码中调用 Java 代码。由于 JNI 是 JVM 规范的一部分，因此可以将我们写的 JNI 的程序在任何实现了 JNI 规范的 Java 虚拟机中运行。同时，这个特性使我们可以复用以前用 C/C++ 写的大量代码 JNI 是一种在 Java 虚拟机机制下的执行代码的标准机制。代码被编写成汇编程序或者 C/C++ 程序，并组装为动态库。也就允许非静态绑定用法。这提供了一个在 Java 平台上调用 C/C++ 的一种途径，反之亦然。

第 3 章. 项目内容

3.1 客户端

客户端前台

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

客户端后台

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
- VPN 程序通过打开/dev/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

3.2 服务器端

服务端在 linux 环境下运行，主要有下面几个功能：

- 创建 IPV6 TCP 套接字，监听服务器和客户端之间的数据通信
- 维护虚接口，实现对虚接口的读写操作
- 维护 IPV4 地址池，实现为新连接客户端分配 IPV4 地址
- 维护客户信息表，保存 IPV4 地址与 IPV6 套接字之间的映射关系
- 读取客户端从 IPV6 TCP 套接字发送来的数据，实现对系统的控制消息和数据消息的处理
- 实现对数据消息的解封装，并写入虚接口
- 实现对虚接口接收到的数据报文进行封装，通过 IPV6 套接字发送给客户端
- 实现保活机制，监测客户端是否在线，并且定时给客户端发送 keeplive 消息

第 4 章. 项目设计与实现

在本次项目中，我们的项目主要分为两大部分，客户端和服务器端。

客户端可以分为两大部分，前台和后台，前台是安卓界面的显示，后台是创建 IPV6 套接字，数据的封装与解封装，以及与 4over6 隧道服务端的通信。本次项目中的客户端整体流程如图 4.1 所示：

Figure 4.1: 客户端整体流程

- 程序启动后，先检查网络状态
- 获取 IPV6 地址
- 开启后台线程
- 开启前台定时器刷新界面 (间隔 1 秒)

前台是 java 语言的显示界面，后台是 C 语言的数据交互，前后台之间通过管道进行通信，这里创建了两个管道，一个是 IP 信息管道，一个是流量信息管道，两个管道分开处理相对简单，通信的详细流程如图 4.2 下所示：

Figure 4.2: 前后台通信流程

- 后台获取到 4over6 服务器发送来的 IP 地址等信息
- 把这些信息解析出来，写入 IP 信息管道
- 前台从 IP 信息管道读取到 IP 地址等信息
- 开启安卓 VPN 服务
- 把安卓虚接口描述符写入 IP 信息管道
- 后台从 IP 信息管道获取安卓虚接口文件描述符
- 对该虚接口进行读写操作

- 记录读写长度和次数
- 在后台定时器线程，定时写入流量信息管道
- 前台从流量信息管道获取流量信息
- 定时刷新界面的流量信息

服务器端主要可以分为三部分：主进程循环、读取虚接口线程、keeplive 线程。主框架流程图如图 4.3 下所示：

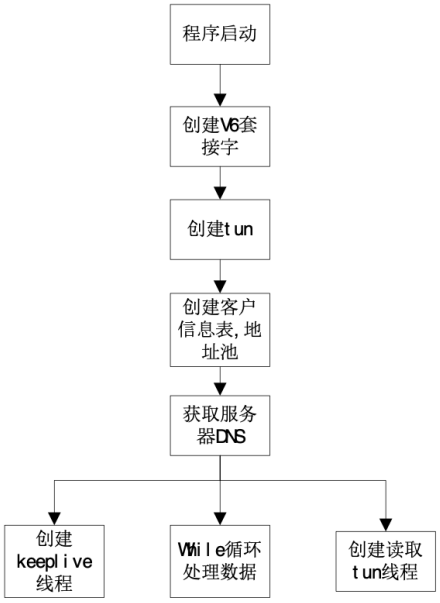

Figure 4.3: 服务器端整体流程

- 创建 IPV6 套接字，把该套接字加入 Select 模型字符集
- 创建 tun 虚接口
- 创建客户信息表和地址池
- 获取服务器 DNS 地址
- 创建 keeplive 线程
- 创建读取虚接口线程
- 主进程中 while 循环中数据处理

4.1 客户端前台流程设计

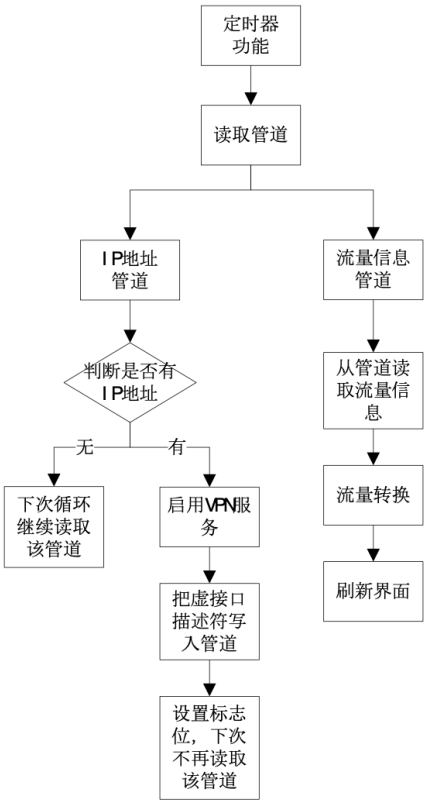

Figure 4.4: 客户端前台流程设计

前台的详细流程图如图 4.4 所示，前台定时器主要功能为定时刷新显示界面，显示流量信息，定时器功能解析如下：

- 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
- 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
- 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并显示到界面
- 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

4.2 客户端前台具体实现

4.2.1 UI 主界面

主界面采用了 ScrollView 嵌套 LinearLayout 的布局方式，将连接 VPN 前后的不同界面通过一个 Layout 展示，通过控制不同组建的可见显示，实现界面的切换。程序界面主要分为连接 VPN 前和连接 VPN 后两个部分。

程序启动时主界面属于连接 VPN 前界面。其包含两个可输入窗口，分别用于动态设置服务器 IPV6 地址以及服务器端口号。另一个展示栏显示当前本机连接状态以及本机 IPV6 地址。另外两个按钮分别实现连接 VPN 功能以及退出程序功能。当本机网络环境满足需求时，即处于 IPV6 网络环境，可以通过点击【连接 VPN】跳转到连接信息展示界面，此时隐藏初始界面的元素，并将信息展示界面元素显现以实现页面转换。

连接前程序界面如图 4.5 所示：

Figure 4.5: 连接前程序界面

连接 VPN 后的界面可以展示当前程序运行时间，本机分配的 IPV4 地址，上网速度，总共接收发送包大小等信息。一个【断开 VPN】按钮可以将本机与服务器断开，并回到初始化的界面。

4.2.2 MainActivity

MainActivity 是本程序的主线程，其主要完成程序的初始化工作，以及启动多个线程以支持程序工作。当程序启动时，MainActivity 首先完成对于界面的初始化，为按钮注册监听事件，并每隔 2s 检查本机的网络环境，调用 NetChecker 的 getIpv6Address 尝试获取本机的 IPV6 地址。只有成功获取了本机的 IPV6 地址，才能继续后续的 VPN 服务。

当点击【连接 VPN】按钮后，首先检查是否允许开启 VPN 服务，满足条件会启动 C 后台线程以及前端的定时器线程。C 后台线程见后端具体实现。

前端的定时器线程同前述工作流程。首先连接 IP 信息的管道，接收 C 后台线程传来的 IPV4 地址，DNS 等信息，然后启动 VPN 服务。通过继承 java.util.TimerTask 实现连接 C 后台线程创建的流量信息管道，读取信息并实现 UI 界面信息的更新。

当点击【断开 VPN】按钮后，会结束 C 后台线程以及前端的定时器线程。

4.2.3 MyVpnervice

MyVpnService 是一个继承了 VpnService 的类，其与主线程 MainActivity 之间通过 Intent 进行数据交换。其需要完成的主要功能是：根据传入的 IPV4 地址，DNS 服务器等参数，初始化 VPN 发服务，并尝试获取虚接口的文件描述符。当获取描述符后，通过管道将数据传给 C 后台。

4.2.4 NetChecker

NetChecker 类主要实现调用 Android 接口，检查本机当前网络环境，并尝试获取 IPV6 地址的功能。isWIFIConnected 负责检查 Wifi 连接状态，getIpv6Address 负责获取本机 IPV6 地址。主线程 MainActivity 只有通过了 NetChecker 类成功获取到本机的 IPV6 地址，才允许前端开启 VPN 服务。

4.3 客户端后台流程设计

后台流程图如图 4.6 所示，详细流程解析如下：

- 创建 IPV6 套接字

- 连接 4over6 隧道服务器
- 开启定时器线程（间隔 1 秒）
- 发送消息类型为 100 的 IP 请求消息
- 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

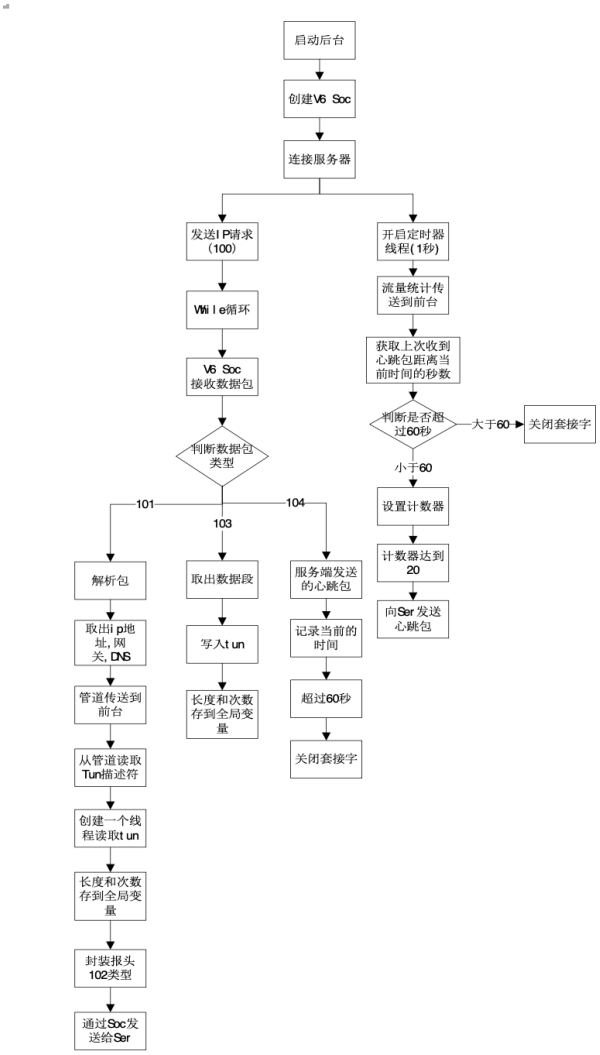

Figure 4.6: 客户端后台流程设计

4.4 客户端后台具体实现

4.4.1 JNI

本次项目需要用 java 代码调用 C 代码，就要用到 JNI。Java Native Interface(JNI) 规定了一套 Java 的原生接口，它提供了若干的 API 实现了 Java 和其他语言的通信 (主要是 C 和 C++)。JNI 是为了本地已编译语言，尤其是 C 和 C++ 而设计的。其支持一个“调用接口”(invocation interface)，它允许把一个 JVM 嵌入到本地程序中。本地程序可以链接一个实现了 JVM 的本地库，然后使用“调用接口”执行 JAVA 语言编写的软件模块。

4.4.2 后台线程

当前端启动 C 后台线程时，会传给后台线程服务器的 IPV6 地址以及端口号信息。后端首先进行初始化工作，并建立虚管道，同时使用已有信息创建客户端 Socket，绑定服务器地址，与服务器并进行 TCP 连接。当连接成功时，开启定时器线程 timer_thread，同时构建 IP 地址请求包，主线程进入 While 循环接收服务器发送来的消息，并对消息类型进行判断，处理数据包。

定时器线程每秒钟会执行一次操作。其主要负责检查与服务器连接是否超时 (60s)，如果超时则退出 VPN 服务。另外将与服务器连接的相关流量信息 (发送/接收包大小，数量) 写入管道，并发送至前台，方便前台进行 UI 界面信息的刷新。同时负责每隔 20s 向服务器发送一次心跳包。

主线程在发送 IP 地址请求包后便会进入循环等待并处理服务器发来的数据包。收包的过程是：先构建并清空一个读取缓存 Buffer，首先读取前 4 个字节，代表整个数据包的长度，然后将长度减去 4 之后，按每次一个字节的顺序，逐次读入缓存 Buffer 中，待读取完毕后，将整个 Buffer 拷贝给 Message 即可。根部 Message 的 type 分类，总共需要处理的包有三类 (IP 地址回应包、上网回应包、心跳包)，以下依次讲解：

- IP 地址回应包：客户端收到服务器发来的 IP 地址回应包，说明与服务器连接成功。此时需要将数据包中内容 (IPV4 地址，DNS 等) 经由管道发送给前端，以开启 VPN 服务。完成上述工作后，开启连两个新的线程，一个线程负责循环读取 tun 文件中内容，构造上网请求包发送给服务器；另外一个线程负责持续监听前端是否发来了结束 C 后台线程的指令，以方便结束后台所以运行的线程

- 上网回应包：当收到服务器发来的上网回应包时，首先需要统计数据包信息，更新和服务器流量变化，同时需要将数据包内容写入 tun 文件。其中在更新信息时，需要进行加锁操作，实现数据在不同线程间的同步互斥
- 心跳包：当收到服务器发来的心跳包，只用更新心跳包收到时间即可，其心跳包发送由定时器线程完成

4.5 服务器端流程设计

4.5.1 主进程循环

主进程 while 循环中主要是 Select 模型监听所有套接字，然后根据套接字收到数据类型，做不同的处理。while 循环如下：

- 在 while 循环中，启用 Select 模型，对所有的套接字进行监听
- 假如监听到服务器套接字，accept 新的连接，把新连接的客户端的套接字描述符加入 Select 字符集
- 假如是客户端套接字，首先用 ioctl 函数判断一下该描述符

  假如 nread 不等于 0，客户端正常连接:

  – 接收数据

  – 对收到的数据解封装

  – 判断数据类型

  假如 nread 等于 0，客户端已经断开:

  – 遍历客户信息表

  – 从信息表中查找该客户端描述符所在节点

  – 取出该节点的 IPV4 地址

  – 用该地址与地址池中地址进行匹配

  – 匹配成功，就把地址池中该地址的标志位置 0

  – 在客户信息表中把该节点删除

– Select 字符集中清除该描述符

while 循环流程如图 4.7 所示：

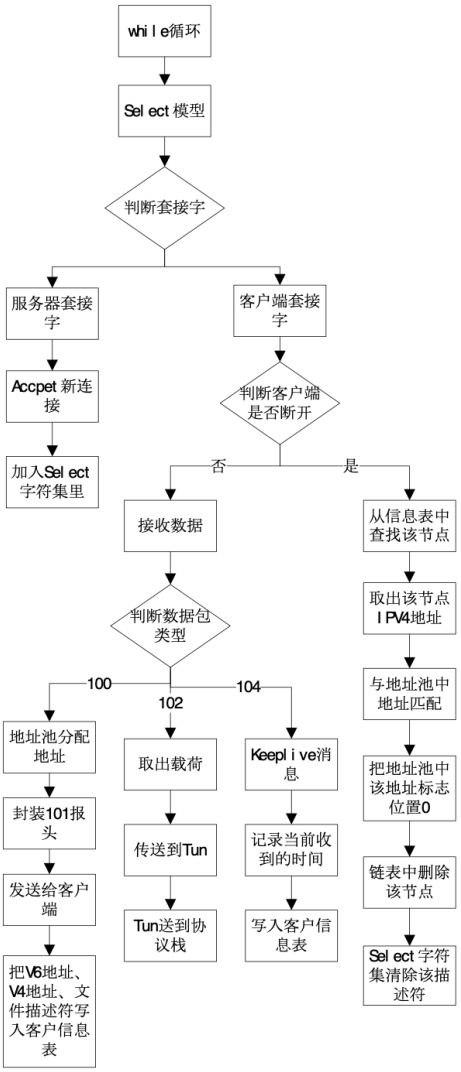

Figure 4.7: 主进程循环流程设计

4.5.2 读取虚接口线程

读取虚接口线程主要功能是从协议栈读取消息，根据消息的目的地址，发送给相应的客户端，主要功能如下：

- 在 while 循环中，从虚接口读取消息

- 取出该消息的 ip 头, 获取目的 ip 地址
- 遍历客户信息表，查找该目的 ip 所在的节点
- 取出该节点的套接字描述符
- 把从虚接口读取到的消息封装 103(上网回应) 报头
- 通过刚才查找到的套接字描述符发送给相应的客户端

读取虚接口线程流程如图 4.8 所示：

Figure 4.8: 读取虚接口线程流程设计

4.5.3 keeplive 线程

keeplive 线程主要功能是给所有当前处于连接状态的客户端发送心跳包，主要功能如下：

- sleep 一秒钟
- 遍历客户信息表
- 链表中的每个节点的 count 字段减 1
- 当该节点的 count 字段等于 0 时，获取该节点的套接字描述符
- 通过套接字描述符向该节点所在客户端发送 104(心跳包) 类型消息

- 发送完成，重新把该节点的 count 字段赋值为 20(每隔 20 秒发送一次心跳包)
- 判断每个节点的 secs 字段的值是否大于 60
- 假如 secs 大于 60，则说明该客户端已经超过 60 秒没有给服务器发送心跳包，获取该节点的 IPV4 地址
- 遍历地址池，找到该地址在地址池中所在位置，把该地址的状态字段置为 0，回收该地址
- 从 Select 字符集中把该节点的套接字描述符清除，关闭该套接字描述符，删除该节点

keeplive 线程流程如图 4.9 所示：

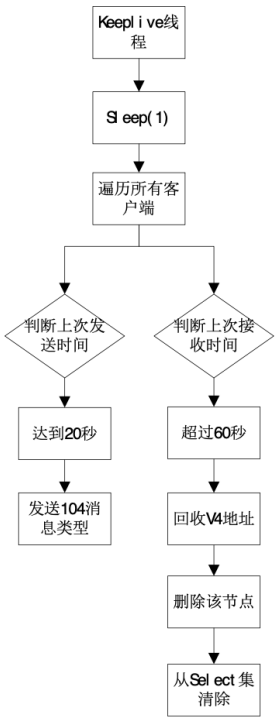

Figure 4.9: keeplive 线程流程设计

4.6 服务器端具体实现

服务器端大致思路在上述流程设计中已经叙述的足够清楚，在此不再赘述。本部分仅对关键技术要点（NAT、epoll 和 socket）进行叙述和讲解。

4.6.1 NAT

服务器端使用 iptables 做 NAT，以此实现私有网段和公网地址之间的转换。当一个客户端发起网络请求时，服务器端的 iptables 会进行”(客户端 IP 地址, 端口) 即 (Src IP, Src Port) -> (服务器端 IP 地址，端口) 即 (Dst IP, Dst Port)” 之间的映射。当公网设备收到上述报文时，服务器端的 iptables 会根据端口号找到对应客户端的 IP 地址和监听端口，对该报文的目的地址和目的端口进行替换，这其实是一个逆映射过程。这样，我们就完成了客户端到服务器端的 NAT，实现了私有网段和公网地址之间的转换。

4.6.2 epoll 和 socket

和网上许多开源的服务器架构一样，我们的项目中，服务器端一样使用了非常便利好用的 epoll 函数，使用 epoll 函数同时监听所有客户端对应的 socket。当 epoll 函数监听到一个可用有效的 socket 的时候，我们首先会根据这个 socket 的描述符找到这个 socket 对应的客户端，然后执行相应的处理函数。当 epoll 函数监听到其中任意一个 socket 状态为 “可写” 时，则从该客户端对应的数据队列里取出数据写入此 socket，持续调用 “write()” 系统调用直到其返回 “EAGAIN”（表明继续写入会导致阻塞），则停止写入。若 epoll 函数监听到其中任意一个 socket 状态为 “可读” 时，则持续调用 “read()” 系统调用直到其返回 “EAGAIN”（表明当前缓冲区里所有数据都已被读取）。若读到完整的消息，则去掉头部后直接写入隧道。我们的隧道同样是使用 epoll 函数进行监听，不同的是根据实现，每次对隧道调用 “read/write” 时都会 “读/写” 一个完整的 IP 包。从隧道中读到 IP 包后，则根据 IP 包头中的 Dst IP 找到对应的客户端，将该 IP 包放入对应客户端的数据队列中，待该用户的 socket 可用后发送。通过上述实现，我们就完成了客户端与服务器端之间的 socket 连接与信息传输。

第 5 章. 项目中遇到的问题

5.1 客户端遇到的问题

我们客户端的开发在项目过程中发现一个问题，即服务器可能向客户端发送不止一个 IP 地址回应包，并且包内容相同。然而在项目中我们仅仅想对 IP 地址回应包进行唯一的一次处理，因为涉及新线程的开启等工作。故我们对于 IP 地址回应包是否进行处理除了判断其数据包类型外，还需要判断是否以前收到了 IP 地址回应包，如果收到了，则将此包丢弃。如果以前没有收到，则处理此包。

5.2 服务器端遇到的问题

服务器端开发我们遇到的最大的问题就是，一开始我们天真的以为客户端和服务器端的开发是完全解耦合的，于是负责客户端开发的同学和负责服务器端开发的同学一开始就是各自为战。但当客户端和服务器端真正一起跑起来之后，才发现并不是像我们想象的那么简单，最大的问题还是服务器端和客户端之间互相通信的问题，服务器能够接收到客户端发来的包，但却无法识别。也就是说服务器端和客户端的包的定义不一致，这也就是说服务器端和客户端并不是完全解耦合的。这一 bug 的调试极大程度地拖慢了我们的进度，当然最终我们还是成功实现了我们自己开发的服务器端和客户端的通信，并可以成功使用我们的客户端通过我们的服务器进行 IPv4 over IPv6 的操作。

第 6 章. 运行方式及效果

6.1 客户端运行方式

- 开发环境：macOS Mojave 10.14.5
- Android Studio 版本：3.4
- 项目运行手机版本：小米 MIX 2S

6.2 服务器运行方式

- 开发环境：macOS Mojave 10.14.5
- 运行环境：整个服务器项目在具有 v4/v6 双栈网络的 Centos 服务器上编译运行
- 运行方式：采用 Cmake 编译运行方式

6.3 最终项目效果

连接服务器前后界面如图 6.1，6.2 所示：

Figure 6.1: 连接服务器前界面

Figure 6.2: 连接服务器后界面

连接宿舍 IPv6 网络 (未认证账户), 使用客户端连接服务器之后访问 IPv6 网站效果如下:

Figure 6.3: 访问 IPv6 网站

连接宿舍 IPv6 网络 (未认证账户), 使用客户端连接服务器之后访问 IPv4 网站效果如下:

Figure 6.4: 访问 IPv4 网站