网络专题训练报告：IPv4-over-IPv6 VPN

计 75 班 赵成钢 2017011362
计 75 班 顾煜贤 2017011421

2020 年 3 月

目录

1 实验目的

1. 掌握 IPv4 over IPv6 VPN 隧道的基本原理
2. 掌握 Android 平台构建 VPN 服务的机制
3. 使用 C++ 语言和 Java 语言完成 Android 客户端的构建
4. 掌握 socket 的基本使用方法
5. 掌握虚接口的建立与使用
6. 使用 C++ 语言完成服务端的构建

2 运行方式

2.1 Android 客户端

文件 /app/release/app-release.apk 为编译好的 APK 包，直接拷贝到 Android 手机或者模拟器上即可。如果需要编译，用 Android Studio 或者 Intellij IDEA 打开整个文件夹作为项目即可。

2.2 服务器端

服务器端采用 Makefile 组织编译和运行，直接在对应的代码目录下运行 make run 命令即可编译并运行。

3 实验原理

3.1 基本原理

如图 1 所示，该方法的基本原理为：将本应该 Android 用户端（图中蓝色部分）要直接发送远程服务器（图中黄色部分）的 IPv4 包通过封装技术放在 IPv6 包的数据段发送给中间代理的网关（图中红色部分），由网关通过 IPv4 网络代为访问服务器，并同时进行数据回送等操作。

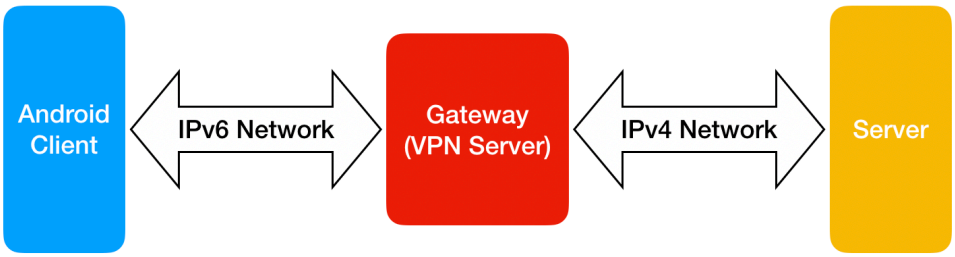


图 1: 基本原理

type 字段	data 字段	备注
100	留空	IP 地址请求
101		IP 地址回应
102		上网请求
103		上网回应
104	留空	心跳

3.2 通信和封装原理

客户端通过直接和网关建立 C IPv6 Socket 进行通信，其中通信传输的数据包的形式如下：

```
// Message
struct Message {
  int length; // includes 'length', 'type' and 'data'
  char type;
  char data[4096];
};
```

其中的 length 段为整个消息的长度，type 段为消息的类型，data 段为数据段。具体而言，根据消息的类型，我们有如下规定：

其中的 IP 地址请求和回应为客户端申请虚拟 IPv4 地址所用的接口；上网请求和回复为客户端通过网关隧道连接正常 IPv4 服务器所用的接口；而心跳则是客户端和服务器端相互发送监测连接状况的接口。

3.3 客户端原理

客户端为 Android 平台，共分为 UI、VPN Service 和 C++ 语言网络后台三个部分。UI 为用户界面负责和用户的交互（如填写地址、连接或者断开）和相关信息（如速度等）的显示；VPN Service 注册到 Android 操作系统，系统将通过一个隧道将全部要发送的数据交给该服务处理；而 C++ 语言网络后台则通过 Raw socket 发送、接受并处理操作系统和服务器的数据。同时，三部分通过 Android Broadcast、Android Intent 或 JNI 接口实现通信。值得一提的是，本实现中没有利用任何管道实现 C++ 和 Java 语言的通信。

3.4 服务器端原理

服务端使用 C++ 语言编写而成。主要提供了建立、删除连接、转发数据包和维护心跳包的服务。当一个用户请求建立连接时，服务端建立对应的 socket，并为客户端分配 IPv4 地址，此后客户端用此地址访问外部 IPv4 网络，同时将用户的 IPv6 地址、IPv4 地址和 socket 关联起来。用户发送数据包给服务端后，服务端将原来封装在 IPv6 数据包中的 IPv4 包取出，写入虚接口发送至外部 IPv4 网络。服务端收到外部 IPv4 数据包后，根据包的目的地址查找用户对应的 socket，并

通过此 socket 关联的 IPv6 地址将包发给对应的用户。服务端同时每隔 20 秒向用户发送心跳包，并接收用户的心跳包。如果用户两个心跳包间隔时间过长，则主动关闭此用户的连接。

4 实验设计

4.1 实验分工

在本实验设计中，赵成钢同学负责了 Android 客户端的编写，顾煜贤同学负责了服务器端的编写。

4.2 客户端设计

同实验原理一节中所述，本客户端分为 UI、VPN Service 和 C++ 语言网络后台三个部分。下面将分别阐述三个部分的设计。

4.2.1 UI (MainActivity)

图形界面的具体视觉设计如图 2 所示，有一个按钮（用于连接和断开）、两个信息栏（用于填写服务器地址和端口）和一个文字显示（用于显示提示信息、连接速度和状态等）。

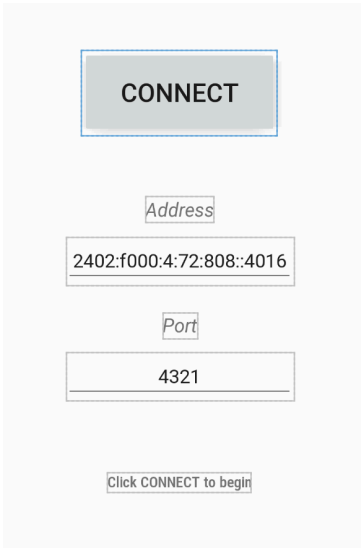


图 2: UI 视觉设计

从逻辑上讲，该部分设计如图 3 所示。用户通过信息输入框和按钮进行交互，而 UI 则通过 Intent 或广播来启动或停止 VPN Service，同时 VPN Service 通过广播来更新 UI 的状态。

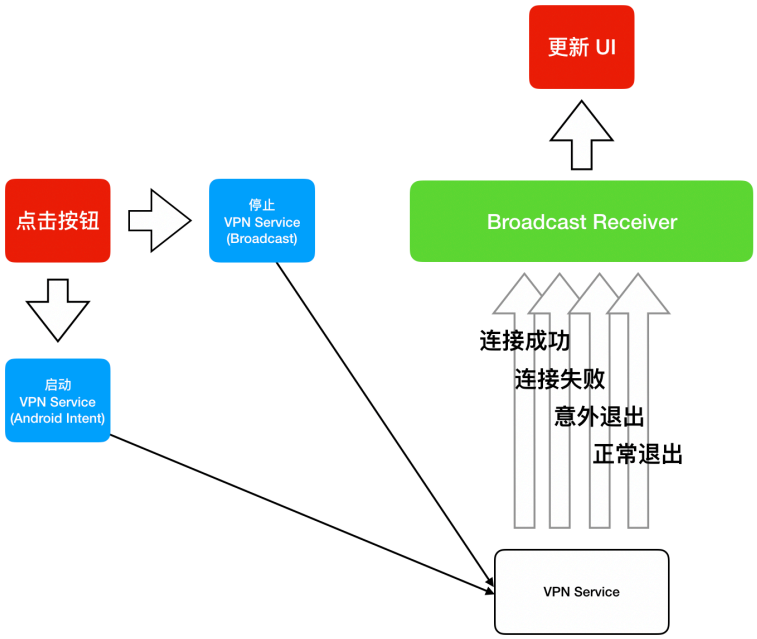


图 3: UI 逻辑

4.2.2 VPN Service

VPN Service 继承自 Android Service，是连接 UI 和网络后台的关键组件，其逻辑设计如图 4 所示。

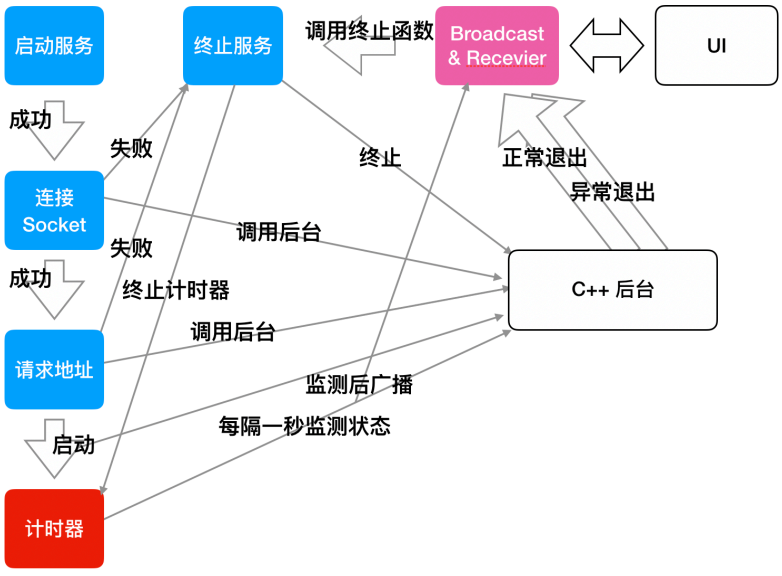


图 4: UI 逻辑

启动时，连接 Socket 和请求虚拟 IP 地址均为串行操作（有超时设置），而且此时后台还没有启动，只是调用了 C++ 后端的连接和请求的两个函数，也不存在多线程的问题。

如果连接失败，则直接终止；否则则启动计时器和后台。

后台的运行通过 Java Thread 包装起来，Thread 完成后会把后台的运行返回情况（正常退出或者异常退出）告诉广播器再通知 UI。

计时器的运行设置为隔一秒来访问后台中的信息统计函数，这里虽然和后台（上文中的 Java Thread）是两个线程，但是计时器读写的都是一些统计信息，通过一些实现的 Trick 也可以不加锁同时保证统计信息的正确。

至此 VPN Service 的运行中只会有两个线程（计时器和后台），同时注意到因为前面启动的逻辑是串行的，后面运行中的逻辑也只是调用，所以这里并没有用到任何 C++ 和 Java 管道通信的技术，实现更为轻量级。

4.2.3 C++ Backend

如图 5 所示，后台的逻辑和 VPN Service 连起来看更是一种调用和返回的关系，其中在建立连接前，VPN Service 通过连接 Socket 和请求 IP 地址两个函数串行完成，之后启动后台核心线程，并用计时器来调用 tik 时钟函数。

后台核心分为发送和接收两个线程，发送线程不断读取隧道并发送到 Socket 中来发送；而接收线程不断接收并写入隧道。同时需要注意的是这里因为是一个线程读一个线程写，所以也不需要用加锁的方式来处理。

对于终止或者异常发送的情况，我这里使用一个变量来控制，发送和接受两个线程则不断通过读取变量来决定是否退出线程，也需要注意的是，这个变量虽然在被三个线程（计时器、发送和接收）同时使用，但是写入只有一种情况，那就是改为 false，所以就算出现了因为冲突，写的内容也一定是一样的，即使出现了读了之后马上写的问题，也只是多运行了一个发生或者接收的周期，不会产生很大影响，综上分析不加任何的锁也没有关系，也就是在本实现中没有使用任何的锁或者管道。

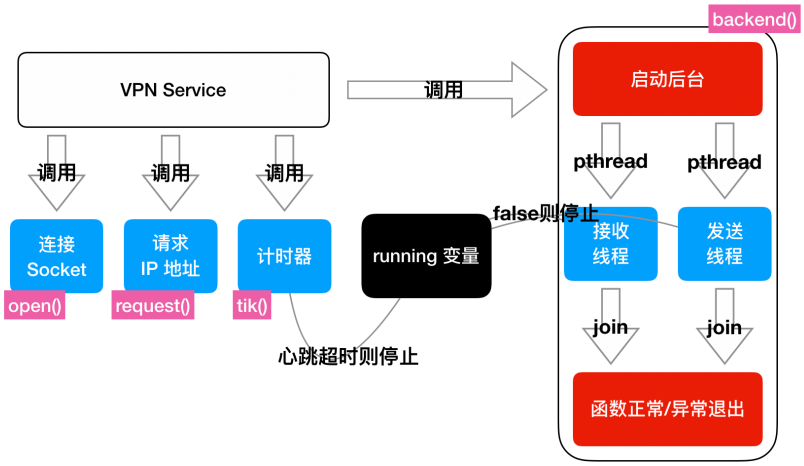


图 5: C++ 后台逻辑

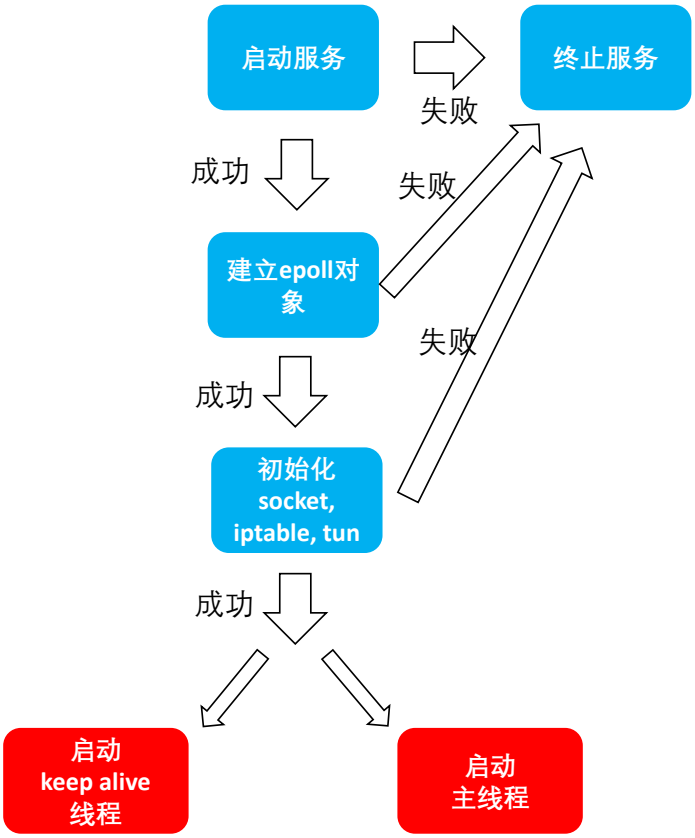


图 6: 服务器启动逻辑

4.3 服务器设计

服务器端的实现没有使用实验指导书中的 Select 模型，而是使用了 epoll 模型。由于 epoll 模型支持 I/O 多路复用，实验指导书中的 While 循环读取数据和 tun 读取数据可以统一到一个线程中处理，都使用 epoll 的事件机制来完成。下面分服务器启动，主线程和 keep alive 线程详细描述服务器设计。

4.3.1 服务器启动

服务器设计逻辑如图 6所示。

在初始化 socket 时，需要将获得的文件描述符注册为一个 epoll 事件。初始化 tun 时，在 /dev/net/tun 位置创建 tun 文件描述符，将此虚接口的设置为开启状态并配置 IP 地址，同时还要将得到的文件描述符注册为 epoll 事件。这样只需主线程中监听 epoll 事件，就可以实现客户端建立连接和 tun 的读取。除了初始化文件描述符，还要设置用户信息表。用户信息表也没有的实现也与实验指导书中的有些许不同：我将 IPv4 地址池和用户信息表实现在了一起，而不是单独用一个链表来实现地址池。用户信息表是一个长度为 MAX_USER 的数组，每个元素类型为 UserInfo

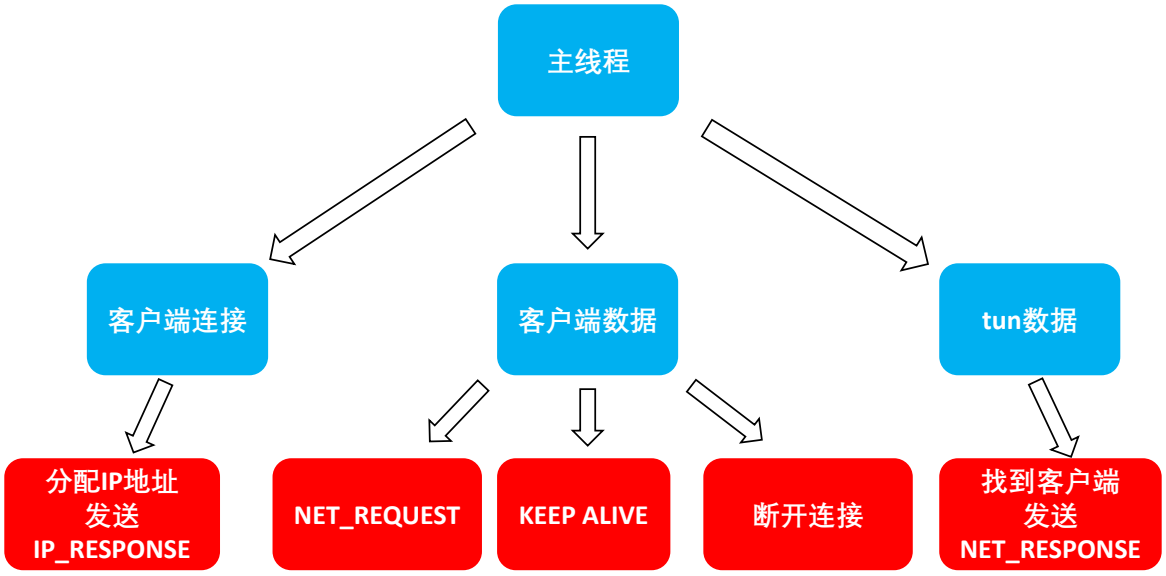


图 7: 主线程设计逻辑

其定义如下，代表一个连接上的用户，初始化时将 v4addr 设置为一组连续的 IP 地址，实现地址池：

```
// UserInfo
struct UserInfo {
    int fd; // 客户端文件描述符
    int count; //距离下一次发心跳包的时间
    int secs; //距离上一次收到心跳包的时间
    struct in_addr v4addr; //客户端的 IPv4 地址
    struct in6_addr v6addr; //客户端的 IPv6 地址
};
```

如果上述初始化过程成功，则创建 keep alive 线程，主线程进入监听 epoll 事件循环。

4.3.2 主线程

服务器主线程的基本逻辑为监听 epoll 事件。当一个 epoll 事件到来时，判断该事件是客户端连接事件，客户端发送数据，还是 tun 数据，主要逻辑如图 7所示。

客户端连接事件 一个客户端请求连接，在当前连接数不超过 MAX_USER 的情况下，接受连接并创建客户端文件描述符，将此描述符注册为 epoll 事件。然后在用户信息表中查找空闲的位置。由于初始化过程时已经将用户信息表的 IPv4 地址设置好，因此找到空闲位置就相当于分配 IPv4 地址了。将客户端文件描述符和其 IPv6 地址存储在找到的表项中。而后通过此描述符发送 IP_RESPONSE 包，其中包含 DNS 地址，分配的 IPv4 地址。

客户端发送数据 通过客户端文件描述符在用户信息表中查找对应表项，客户端发送数据有三种情况：

1. NET_REQUEST 包：将包中的 IPv6 头去掉，剩下的刚好是 IPv4 包，通过 tun 发出。
2. KEEP_ALIVE 包：表项 secs 值置为 0。
3. 使用 ioctl() 函数判断客户端断开连接，则关闭此客户端文件描述符。

tun 数据 tun 数据为外部 IPv4 网络的回应数据。先根据数据包的 IPv4 地址从用户信息表中找到对应的用户表项，然后根据表项中封装的 IPv6 地址封装 IPv6 包，写入客户端的文件描述符。

4.3.3 Keep alive 线程

Keep alive 线程中主要做两件事：每隔 1 秒，对于用户信息表中的每个用户（1）检查 count，如果等于 0，则发送心跳包，并设置回 20，否则减 1。（2）检查 secs，如果和当前时间相差大于 60s，则关闭客户端连接。

5 遇到的问题和解决

5.1 Android 客户端

5.1.1 IP Reply 需要补零

在调试中，我发现有些网站可以登陆而很多网站不能登陆的问题，仔细思考之后发现应该是 DNS 的设置问题，我检查了 IP Reply 的数据，我发现学校和队友返回的字符串结尾都没有结束符 0。这导致了我使用的字符串解析函数没有把最后一个 DNS 地址（在字符串的末尾）正确解析出来。

解决方案为判断是否有结束符，如果没有加一个结束符即可。

5.1.2 UI 和 VPN Service 的通信

在实验过程中遇到的一个很大的困难是解决如何让 UI 和可以独自在后台运行的 VPN Service 通信，这里浪费了我大量的时间，开始采用了 Bind 或者管道的设计，最后都导致了一些关联性的问题，因为在后台运行的时候，UI 线程可能会被销毁等等操作。

最后的解决方案为采用无任何关联的 Broadcast 实现，通过全局广播来使得双方进行通信。

5.1.3 在模拟器上的 IPv6 环境

为了方便调试，我也利用了 QEMU 模拟的虚拟环境，而且我的电脑接入的 PPPoE 网络也不支持 IPv6 环境。

解决方法是把 iPhone 通过 USB 连接的方式插入电脑中作为热点，不过需要注意当 PPPoE 和热点同时存在时，需要把 QEMU 模拟的环境的网络包转发到热点中，而且需要把 QEMU 中的默认 DNS 等等设置改为手机的配置，而不是电脑本身的配置。

5.2 服务器端

5.2.1 为服务器申请 IPv6 地址

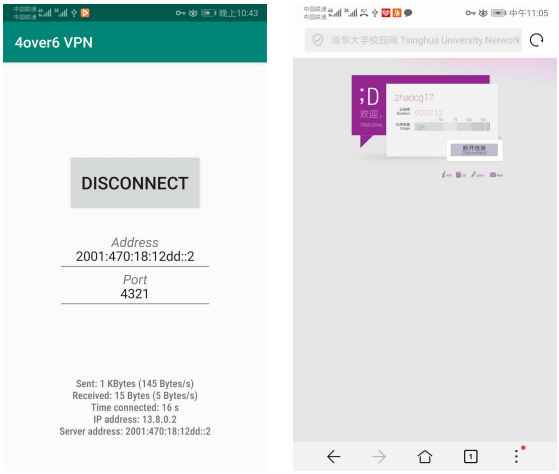
为了完成实验，我们需要一台公网的 IPv6 服务器，但是现在国内很多提供租赁服务器的公司对于 IPv6 的支持都不完善。一开始我打算自己申请一个免费的 IPv6 地址配置到阿里云的 IPv4 服务器上，但是这样配置十分麻烦，并且配置好后网络延迟很大。最后我加入了阿里云的 IPv6 内测，才解决了这个问题。

5.2.2 IP Reply 的补零问题

一开始服务器端传递给客户端的 IP Reply 字符串最后并没有加入结束符 0，导致客户端不能无法解析部分的 DNS 地址。后来我发现发送的数据长度应该是加一，即加入最后的分隔符，这样才解决了这个问题。

6 实验结果

如图 8 分布为软件的截图和上网的截图，本实验将服务器程序部署在助教提供的清华服务器上并通过 HUAWEI P10 手机（Android 版本为 9.1.0），成功访问了在学校的 net.tsinghua.edu.cn 等网站，并通过清华上网服务访问了外网，日常应用（如微信、视频等）使用正常，通过 IP 检测服务也显示是教育网。



(a) 软件截图　　(b) 上网截图

图 8: 运行截图

同时，本实验中把服务器部署到自行租赁的阿里云服务器，并用客户端连接到对应的地址，也可以进行比较流畅的上网、使用日常应用等等一系列操作，IP 检测服务对应显示的地址也是阿里云的地址，这验证了客户端和服务器端的实现符合实验预期结果。

7 总结

本次实验顺利实现了 IPv6 over IPv4 的 VPN 隧道，分工实现了 Android 客户端和服务器端。同时，值得一提的是，除了在本训练课程中收获的知识，在疫情期间，在家中通过自己搭建的 VPN 隧道来访问清华的内网，并通过校园网服务来上网是一件非常令人激动和高兴的事情。

同时，很感谢老师和助教的耐心讲解、答疑和指导书撰写。这是一门思路清晰并有实质性收获的一门课。

另外，综合自己的体验，我们也想提出几个建议：

1. 在客户端部分，开发的一大部分时间花费在了 Android 编程上，为了良好的用户体验，必须学习很多界面和 Android 服务相关的知识，而且 UI、框架间通信等内容花费的时间也比最核心的网络部分多，这在“网络专题训练”为题的课程下，应该进一步改进，可以通过提供基础 Android 框架来实现；

2. 客户端实验指导书中提出的方法实现起来比较冗杂，个人认为 C++ 和 Java 的通信通过管道来实现不是一个很优雅的做法，更推荐本报告中提出的框架或更轻量级的方法来解决；而且也有可能完全使用 Java 中的 socket 来做这件事情；

3. 服务端部分，很大一部分时间在配置服务器上。国内许多提供租赁服务器的公司对于 IPv6 的支持都不完善，导致找到或配置一台有 IPv6 地址的服务器花了很多时间。因此还是希望学校可以提供实验用的服务器。同时因为很多与网络相关的操作都需要拥有 sudo 权限，如果提供服务器的话，如何同时保证学生有足够权限同时又不会对机器系统造成破坏是需要考虑的。