

“计算机网络安全”实验报告

——VPN实验

|  |  |
| --- | --- |
| **院 系：** | 网络空间安全学院 |
| **专业班级：** | 网安2104班 |
| **姓 名：** | 邬雪菲 |
| **学 号：** | U202112131 |
| **日 期：** | 2024年6月18日 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **评分项** | **实验报告评分**  **（50%）** | **检查单分数**  **（50%）** | **综合得分** | **教师签名** |
| **得分** |  |  |  |  |

实验报告评分表

|  |  |  |  |
| --- | --- | --- | --- |
| **评分项目** | **分值** | **评分标准** | **得分** |
| 实验原理 | 10 | 10-8：原理理解准确，说明清晰完整  7-5：原理理解基本准确，说明较为清楚  4-0：说明过于简单 |  |
| VPN系统设计 | 25 | 25-19：系统架构和模块划分合理，详细设计说明翔实准确  18-11：系统架构和模块划分基本合理，详细设计说明较为准确  10-0：系统架构和模块划分不恰当，详细设计说明过于简单 |  |
| VPN实现细节 | 25 | 25-19：回答准确清晰，实现方法技术优良，与设计及代码一致  18-11：回答较准确清晰，实现方法一般，与设计及代码有偏差  10-0：文字表达混乱，实现方法过于简单 |  |
| 测试结果与分析 | 20 | 20-15：功能测试覆盖完备，测试结果理想，分析说明合理可信  14-9：功能测试覆盖基本完备，测试结果基本达标，分析说明较少  8-0：功能测试覆盖不够，测试未达到任务要求，缺乏分析说明 |  |
| 体会与建议 | 10 | 10-8：心得体会真实，意见中肯、建议合理可行，体现个人思考  7-5：心得体会较为真实，意见建议较为具体  4-0：过于简单敷衍 |  |
| 格式规范 | 10 | 图、表的说明，行间距、缩进、目录等不规范相应扣分 |  |
| **总 分** | | |  |

目 录

[1 实验原理 1](#_Toc20511)

[1.1 网络拓扑 1](#_Toc27280)

[1.2 通信机制 1](#_Toc28292)

[1.3 加密原理 3](#_Toc4927)

[1.4 认证机制 4](#_Toc21494)

[1.5 多用户并发 4](#_Toc782)

[1.6 进程间通信 5](#_Toc26691)

[2 VPN系统设计 6](#_Toc20126)

[2.1 概要设计 6](#_Toc20814)

[2.2 详细设计 10](#_Toc233)

[3 VPN实现细节 15](#_Toc68)

[3.1 实验环境 15](#_Toc30748)

[3.2 网络配置 15](#_Toc3468)

[3.3 问题1 16](#_Toc31299)

[3.4 问题2 16](#_Toc312)

[3.5 问题3 17](#_Toc18554)

[4 测试结果与分析 18](#_Toc21504)

[4.1 认证VPN服务器 18](#_Toc27772)

[4.2 认证VPN客户端 20](#_Toc5388)

[4.3 加密隧道通信 21](#_Toc7107)

[4.4 支持多客户端 22](#_Toc17176)

[4.5 易用性和稳定性 24](#_Toc9371)

[5 体会与建议 26](#_Toc7100)

[5.1 心得体会 26](#_Toc15570)

[5.2 意见建议 27](#_Toc31188)

# 实验原理

## 网络拓扑

如图1-1为实验的网络拓扑，VPN隧道将创建在本地网关服务器和客户端之间，允许主机通过网关安全地访问专用网络。外网主机HOSTU0和HOSTU1上运行VPN客户端，其IP地址为10.0.2.xxx，使用容器创建。本地虚拟机作为VPN网关，运行VPN服务器（VM），负责数据的转发，其在外网的IP地址为10.0.2.8，在内网的IP地址为192.168.60.1。内网主机HOSTV的IP地址为192.168.60.xxx，使用容器创建。

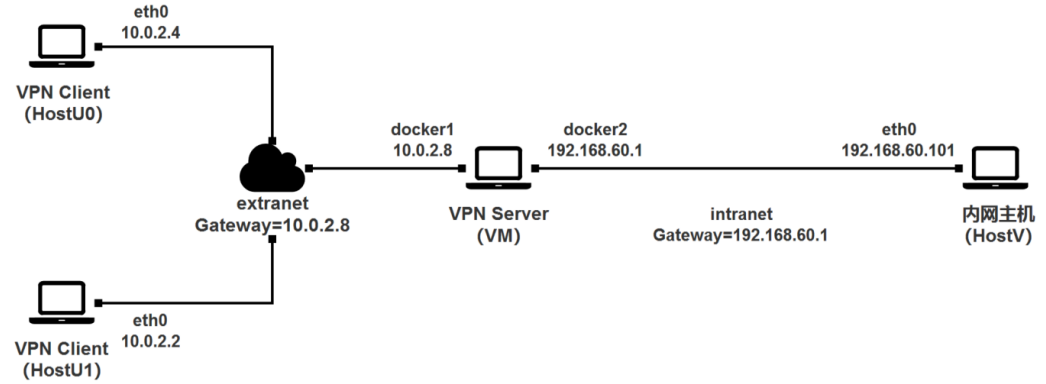


图1-1 实验环境网络拓扑图

## 通信机制

在本次实验中利用隧道技术实现安全的VPN通信。其中TUN技术使虚拟网络内核驱动程序，用来模拟网络层设备，处理第三层IP数据包，允许用户直接与网络层或者数据链路层交互。在VPN中，TUN技术用于在用户和内核网络栈之间创建虚拟网络接口，传送IP数据包。VPN客户端和服务端通过TUN接口建立隧道的通信流程如下图所示：

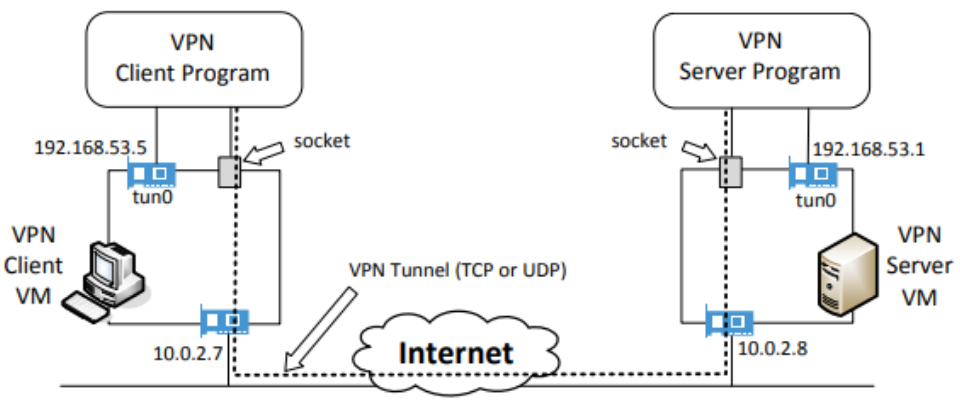


图1-2 VPN隧道图

以外网主机HOSTU向内网主机HOSTV发送数据包为例：

1. 外网用户启动VPN客户端，与VPN网关上的VPN服务器建立TLS连接并由服务器分配外网主机的虚拟IP
2. 外网主机程序数据包发送，路由到tun0接口，则系统首先将HOSTU要发送的报文写进tun0接口中。
3. VPN客户端持续监听tun0虚拟接口，从tun0中读取用户写入的数据报，加密封装后利用TLS连接，通过TLS隧道进行传输。
4. 数据包经过外部网络传输到VPN服务器，服务器收到后将报文解封并写入tun0中，tun0根据解封后报文的目的ip进行路由，docker2将目的地为192.168.60.101的报文转发给HOSTV的eth0接口，接口再转发给HOSTV。

其具体流程如下图所示：

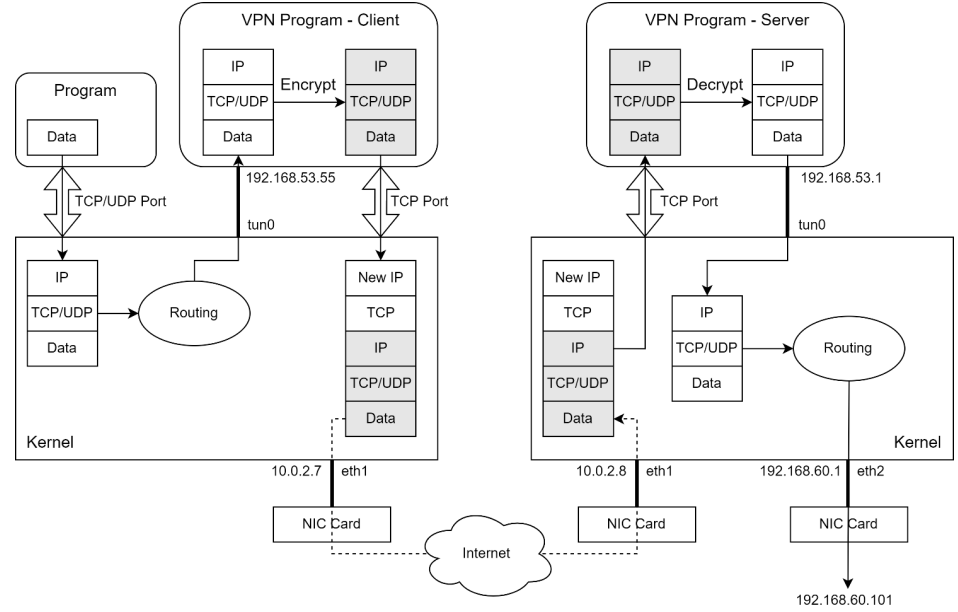


图 1-3 外网主机向内网主机发送数据的通信过程

而从内网主机向外网主机发送数据的通信流程与以上流程相反，在此便不赘述。

## 加密原理

在VPN的实现过程中，使用SSL/TLS加密服务进行加密，TLS加密通信的过程如下图所示：

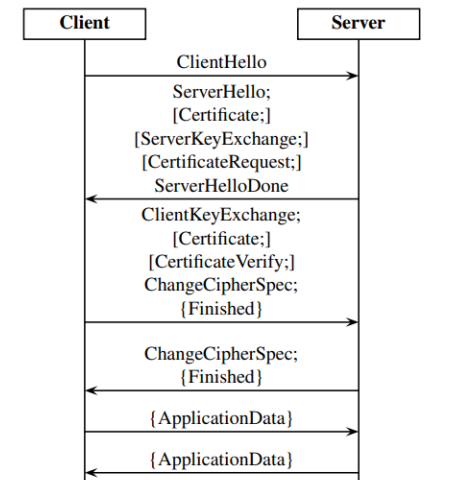


图 1-4 TLS握手过程

1. 客户端问候阶段：当建立安全连接时，客户端首先发出问候信息，称为"客户端问候"。该信息包括：客户端版本、随机数、密码套件列表、扩展功能等。
2. 服务器响应阶段：服务器在接收到客户端问候后，如果能够继续进行协商，则会回复"服务器问候"信息，否则会请求重新协商。"服务器问候"包含：服务器版本、随机数、选定的密码套件、扩展结果等。
3. 证书交换阶段：在确定了双方都能接受的密码套件后，服务器会根据需要发送"证书消息"，其中包含了服务器的证书链。这一步骤的目的是验证服务器的身份，并允许客户端从证书中获取服务器的公钥，以便生成预备主密钥。
4. 服务器问候完成阶段：在RSA密钥协商算法中，服务器在发送完证书消息后，会立即发送"服务器问候完成"消息。
5. 客户端密钥交换阶段：客户端在接收到"服务器问候完成"消息后，计算主密钥，使用RSA算法加密，随后通过"客户端密钥交换"消息发送给服务器。服务器在接收到该消息后，将计算主密钥和会话密钥，而客户端也会在本地进行相同的计算。
6. 更改密码规范和完成握手阶段：客户端在发送"客户端密钥交换"消息后，会连续发送"更改密码规范"消息和"完成握手"消息。服务器也会相应地发送这两个消息。如果"完成握手"消息的校验成功，那么握手过程即告完成。
7. 加密通信阶段：一旦握手过程成功完成，客户端和服务器将使用协商确定的会话密钥对所有后续通信数据进行加密，并将其封装到TCP数据中传输。

## 认证机制

在开始VPN通信之前，需要对客户端和服务器均进行认证以确保连接的合法性和安全性。认证包括两个方向，一个是客户端对服务器进行认证，保证服务器是真实有效的，另一个是服务器需要对客户端进行身份校验，以确定其是否拥有访问专用网络的权限。

对服务器的认证借助公钥证书实现。此过程依靠第三方证书机构，服务器向证书机构申请证书，当客户端向服务器发送连接申请时，服务器利用证书来证明自身是客户端的目标服务器而非伪造的虚假服务器，具体通过判断CA公钥是否经过CA签名且是否在有效期限内来完成对服务器的认证。

对客户端的身份认证一般VPN其实是不包括的，但是可以通过登录账户和用户名进行认证。服务器会验证客户端的账户和密码是否和其内存中的记录情况匹配，如果有相应的记录，则可通过身份认证。

## 多用户并发

在实际部署中，VPN服务器具备处理多个VPN隧道的能力。该服务器能够同时接受来自多个客户端的连接请求，每个客户端均拥有独立的VPN隧道，并且各自维护一个TLS会话。服务器的核心功能包括接收来自不同客户端的数据包，并确保这些数据包能够正确地发送至对应的客户端。

VPN服务器的主进程负责为每个建立的VPN隧道创建一个独立的子进程。当数据包通过隧道传输至服务器时，相应的子进程将捕获这些数据包，并通过TUN接口将其转发。在数据包从专用网络到达TUN接口时，主进程将负责接收这些数据包，并决定其应该被转发至哪个VPN隧道。一旦确定了目标隧道，主进程将负责将数据包路由至对应的子进程。

为了高效地管理不同客户端的消息接收和定向转发，我们采用了管道技术。每个子进程将监控其对应的管道接口，以便在数据到达时立即读取。处理完数据后，子进程将通过其管道接口将响应消息发送回相应的客户端。

具体过程如下图所示：

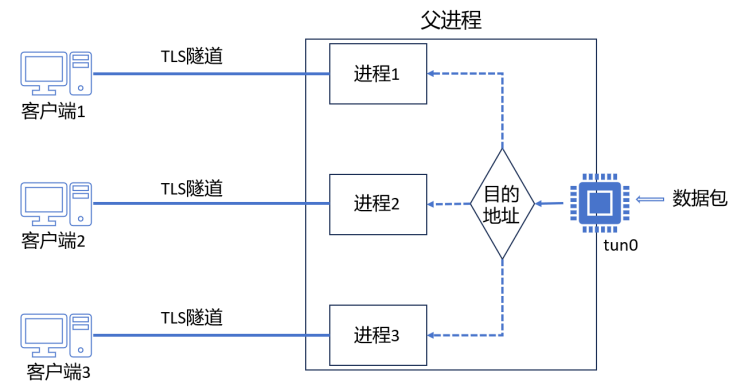


图 1-5 VPN多用户并发

## 进程间通信

为了实现多用户并发，服务器父进程需要为每个用户创建一个子进程，并涉及进程间通信机制，通过管道pipe实现父进程和子进程之间的数据传递。本实验具体使用了命名管道，用给用户分配的虚拟IP地址为每个用户的管道命名，由此可以确定数据包应该通过什么路由到目的地。但是，这个管道是单向的，我们只允许该管道将数据从父进程发送到子进程，而父进程不会从管道中读取任何内容，这通过关闭输出端fd[1]实现。

# VPN系统设计

## 概要设计

VPN系统架构包括客户端和服务器的实现。

### 服务器架构

服务器采用多进程多线程架构，主要包括四个模块：主控模块、tun通信模块、管道通信模块、TLS连接模块，具体如下图所示：

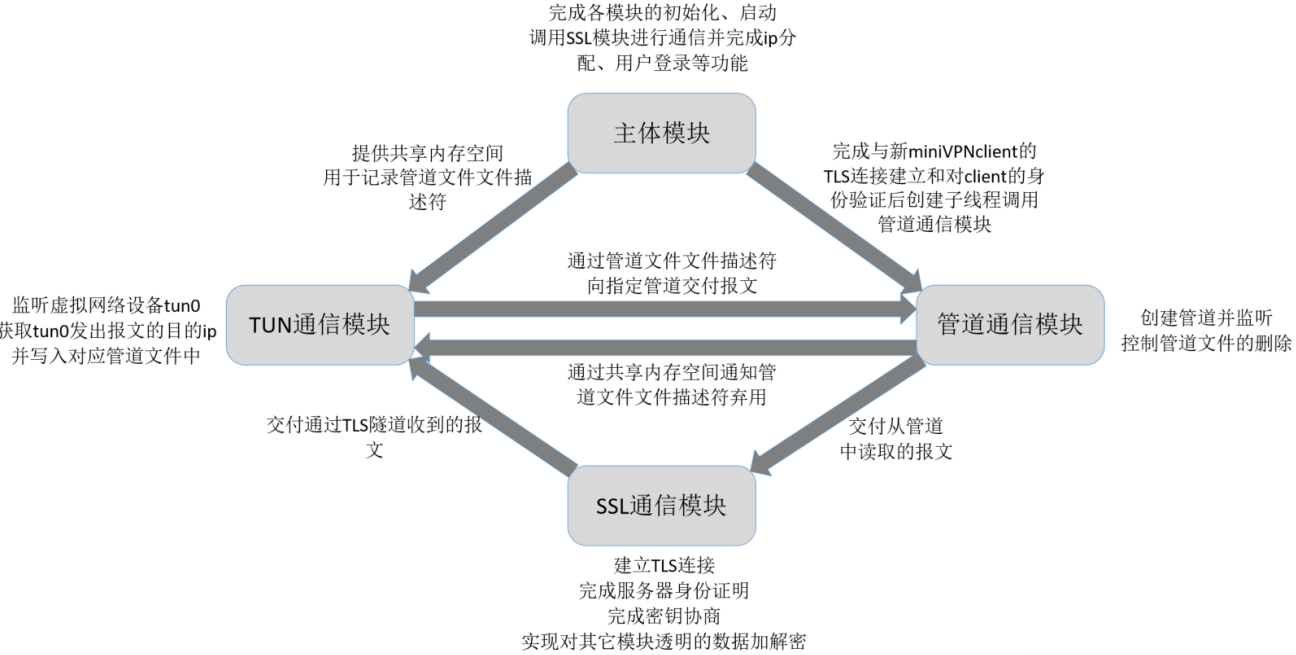


图 2-1 服务器架构

主控进程负责执行初始化任务，包括从TUN设备读取数据并将其分发到相应的命名管道。该进程持续监听客户端的连接请求。一旦检测到新的客户端尝试与服务器建立连接，主控进程即启动一个新的子进程。该子进程专门负责与该客户端的交互，提供必要的服务和数据处理。这种设计允许系统高效地处理多个并发连接，同时确保每个客户端都能获得独立的服务实例。

服务器在与新的客户端建立连接后，将启动一个专用的子进程来为该客户端提供服务。子进程的工作流程，包括完成连接建立的后续步骤，具体包括：建立TLS安全连接；执行客户端身份认证；协商虚拟IP地址；创建与客户端通信的对应命名管道。

子进程执行的任务与VPN客户端类似。客户端发送至内部网络的数据包，通过SSL连接传输至VPN服务器的对应子进程，子进程负责将这些数据包转发至TUN设备，随后由操作系统内核根据路由表将数据包发送至目标内部主机；

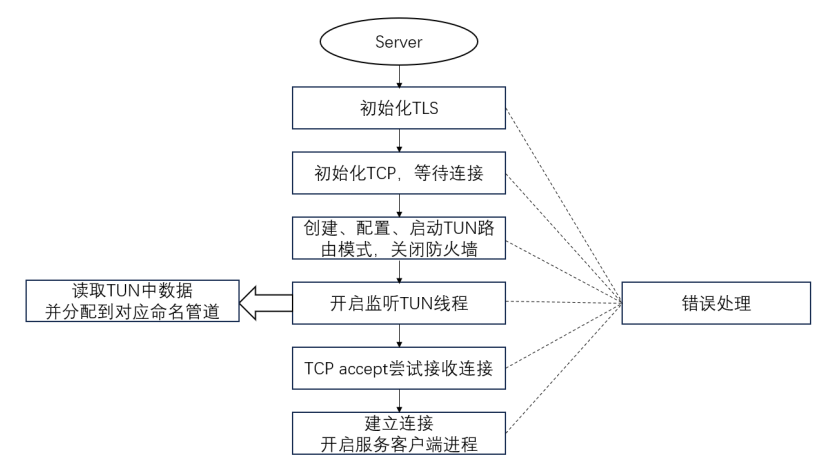


图 2-2 父进程工作流程

内部网络主机发送至VPN客户端的数据包，首先由主进程从TUN设备接收，然后根据数据包的目的地址，将其写入相应的命名管道。子进程从命名管道中读取这些数据包，并通过SSL连接将其发送至VPN客户端。

为了实现对多个客户端的服务，服务器利用命名管道功能，为每个客户端根据其虚拟IP创建一个独立的命名管道。这样，VPN客户端发送至内部网络的数据包只需通过SSL连接由VPN服务器转发至TUN设备，即可通过内核路由至目标主机。对于内部网络主机发送至外部网络VPN客户端的数据包，服务器主进程在接收到这些数据包后，根据其目的地址，将其写入相应的命名管道。负责与目标外部主机通信的子进程将从命名管道中读取数据包，并通过SSL连接将其发送至目标外部主机。

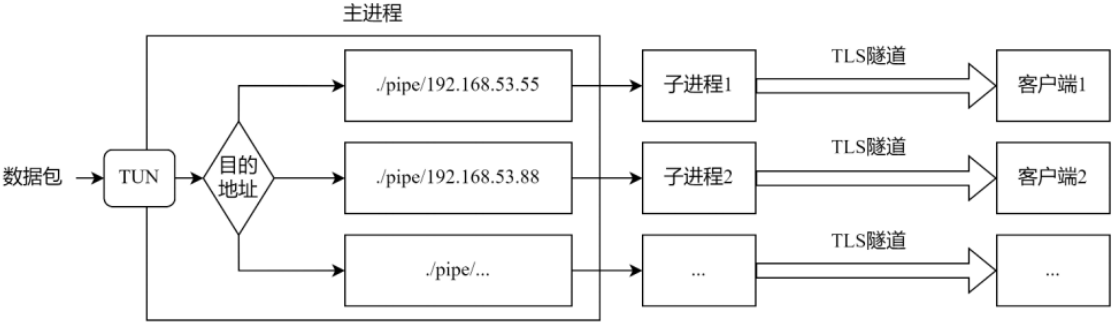


图 2-3 多用户并发VPN服务器

### 客户端架构

VPN客户端可分为主体模块、TUN通信模块、TLS连接模块。

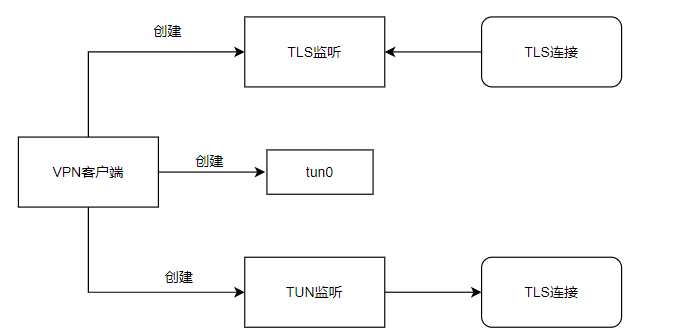


图 2-4 VPN客户端架构

其工作流程遵循单一进程和单一线程的架构模式。一旦VPN连接被正式确立——这包括TCP连接建立、TLS握手完成、服务器端TLS身份认证、客户端身份验证、虚拟IP地址协商以及本地网络配置——客户端生成的、目的为内部网络的数据包将被路由至TUN设备。VPN客户端程序随后从TUN设备读取这些数据包，并通过SSL安全层进行封装，然后将其传输至VPN服务器。

对于来自内部网络、目标为本机的数据包，VPN服务器将通过SSL安全通道将它们发送至相应的VPN客户端。VPN客户端程序将从SSL连接中提取这些数据包，并将其转发至TUN设备，随后由操作系统内核将数据包传递至目标应用程序。因此，VPN客户端的主要任务是监控TUN设备接收的数据流以及SSL连接接收的数据。

为了有效管理这两种数据流，VPN客户端采用了基于select系统的多路输入监听机制。该机制允许客户端程序同时监听多个数据输入源，从而确保数据的实时处理和响应。程序将持续运行，直到SSL连接正常断开或遇到其他错误导致程序终止。这种设计确保了VPN客户端在维护数据传输的安全性和效率的同时，也能够灵活地处理各种网络事件。

### 工作机制

首先是连接建立机制：

1. 在虚拟机(VM)上启动VPN服务器程序。
2. 在主机(HostU)上启动VPN客户端软件，输入VPN服务器的端口号，并根据提示输入认证所需的用户名和密码。随后，指定用于建立TCP连接的端口号和主机名，以对VPN服务器进行身份验证。验证成功后，启动虚拟网络接口tun0，并输入PEM密钥的密码短语以完成认证。
3. 认证成功后，将在内部网络(intranet)和外部网络(extranet)之间建立VPN通信隧道。此时，HostU能够访问extranet中的内网资源。
4. 同时，在另一台主机(HostU2)上启动VPN客户端，并尝试连接至VPN服务器。VPN服务器将为每个客户端创建独立的线程，并将用户信息记录在命名管道(/pipe)中。

外网客户端发送数据机制：传入数据的流程始于客户端程序发起连接请求，数据通过虚拟网络接口tun0获取虚拟IP地址。随后，VPN客户端软件对数据进行处理，并通过物理网络接口发送至VPN网关。VPN网关作为VPN服务器，接收并处理来自客户端的数据。数据被发送至网关的虚拟网络接口tun0，再通过网关的物理内网接口eth1，最终传输至目标内网主机。

内网主机发送数据机制：传出数据的流程从内网主机开始，主机生成数据包并发送至VPN网关。VPN网关对数据进行处理，包括识别与数据包对应的客户端子进程、执行数据加密等操作。处理后的数据通过VPN隧道发送至指定的VPN客户端。VPN客户端程序接收数据，并将其解码后返回给发起请求的应用程序。

## 详细设计

### 服务器设计

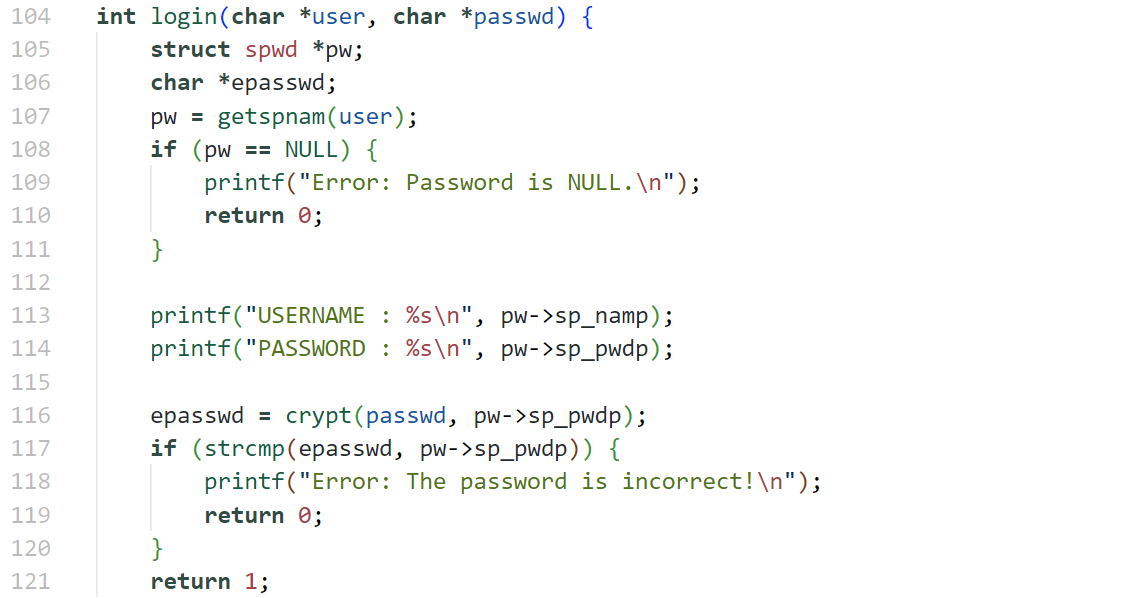
1. **服务器认证客户端**

服务器使用 /etc/shadow文件中的账户信息对客户端的登录身份进行匹配。具体身份验证步骤如下：

1. 获取加密的用户信息：服务器首先调用 getspnam(user\_name) 函数，根据提供的用户名检索对应的加密身份信息。该函数会返回一个包含用户密码散列的结构体。若查询的用户不存在，则返回 NULL。
2. 密码加密与对比：服务器随后使用 crypt() 函数对客户端提交的密码进行加密处理，加密方式与存储在 /etc/shadow 文件中的散列密码保持一致。加密后的密码将与 /etc/shadow 中存储的散列值进行比对，以验证用户名和密码是否匹配。这一过程是服务器验证客户端身份的关键环节，确保只有经过验证的用户才能获得访问权限。

认证结果处理：

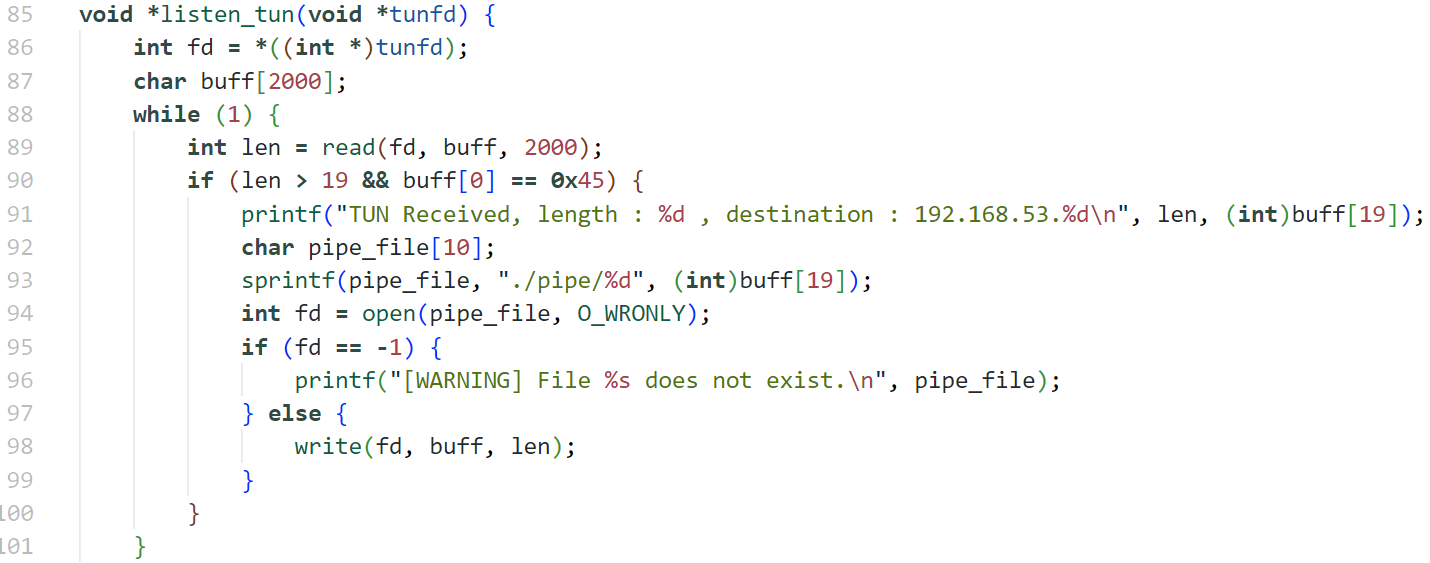
1. 如果认证失败，即用户名不存在或密码不匹配，返回错误原因
2. 如果认证成功，即用户名存在且密码匹配，返回成功提示信息过



1. **多客户端服务**

使用命名管道实现。具体实现步骤如下：

1. 创建子进程和命名管道：客户端发起连接请求时，服务端首先创建一个子进程。接着，服务端根据tun隧道分配给客户端的虚拟IP地址（格式为192.168.53.x），其中x代表IP的最后一字节，依据此值，在/pipe目录下创建一个名为x的命名管道（pipe）。
2. 监听和转发报文：服务端利用之前建立的tun监听线程对tun设备进行持续监听。当监听到报文时，服务端将对那些需要发送回客户端的报文进行处理，而对于客户端向外发送的报文，则只进行必要的转发操作。服务端根据报文中的目标IP地址的最后一字节，确定对应的命名管道，并使用write()函数将监听到的完整数据信息写入该管道。这一监听和转发过程是通过listen\_tun()函数实现的。



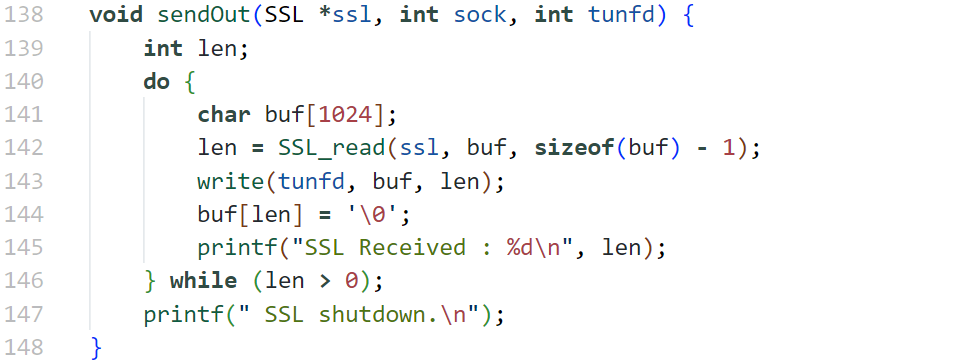
1. **发送数据**

sendOut() 函数是网络通信过程中的关键组件，其核心作用是处理客户端发出的数据并将其有效传输至目标网络。

读取客户端数据：sendOut() 函数首先执行的操作是从客户端接收数据。这些数据可能包含客户端希望服务器代为转发到外部网络的各类请求或信息。

写入 tun 设备：接收到数据后，sendOut() 函数将这些数据写入到服务器上的 tun 设备。tun 设备作为一个虚拟网络接口，提供了一种机制，允许数据包通过服务器的内部网络栈进行路由和转发。

数据转发：一旦数据被写入 tun 设备，它将被自动路由至配置的虚拟网络接口，并最终转发到数据包中指定的目标 IP 地址。这个过程实现了客户端通过网络虚拟私人网络（VPN）连接发送数据到外部网络的需求。



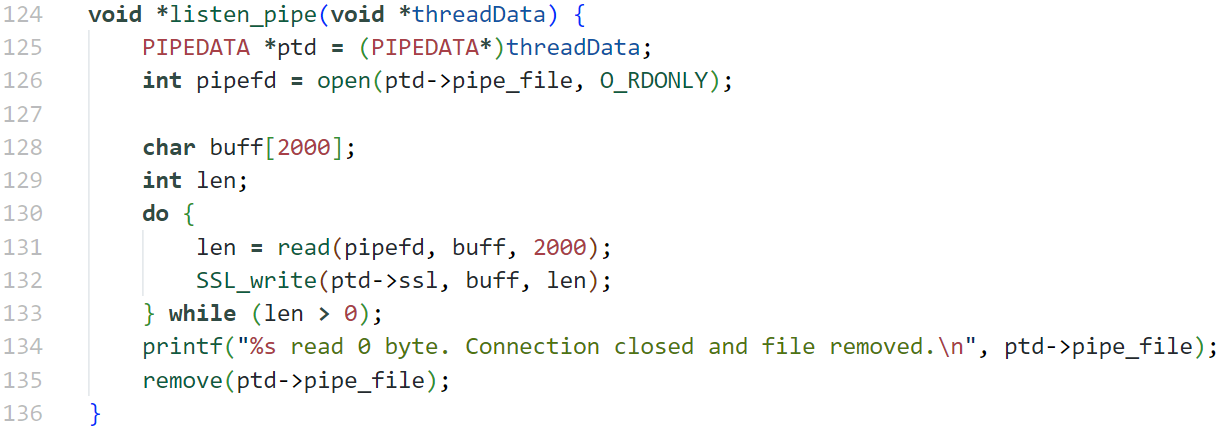
1. **监听管道**

listen\_pipe() 函数是服务器端处理客户端通信的重要组成部分，其主要功能可以概括为：

监听命名管道（Pipe）：该函数使用一个封装了命名管道文件描述符的 PIPEDATA 结构体。通过这个结构体，listen\_pipe() 能够监控并等待特定命名管道中的数据。这种监听是持续的，确保任何发送到管道的数据都能被及时捕获。

读取数据：当 listen\_pipe() 检测到管道中有数据可读时，它会从管道中读取这些数据。这通常意味着客户端通过 VPN 隧道发送了某些数据，现在需要由服务器端进行进一步处理。

发送数据：读取到的数据随后会通过已建立的 SSL 安全连接发送给对应的客户端。这一步骤确保了数据的安全性和完整性，同时实现了从服务器到客户端的数据转发，从而完成了整个通信过程。

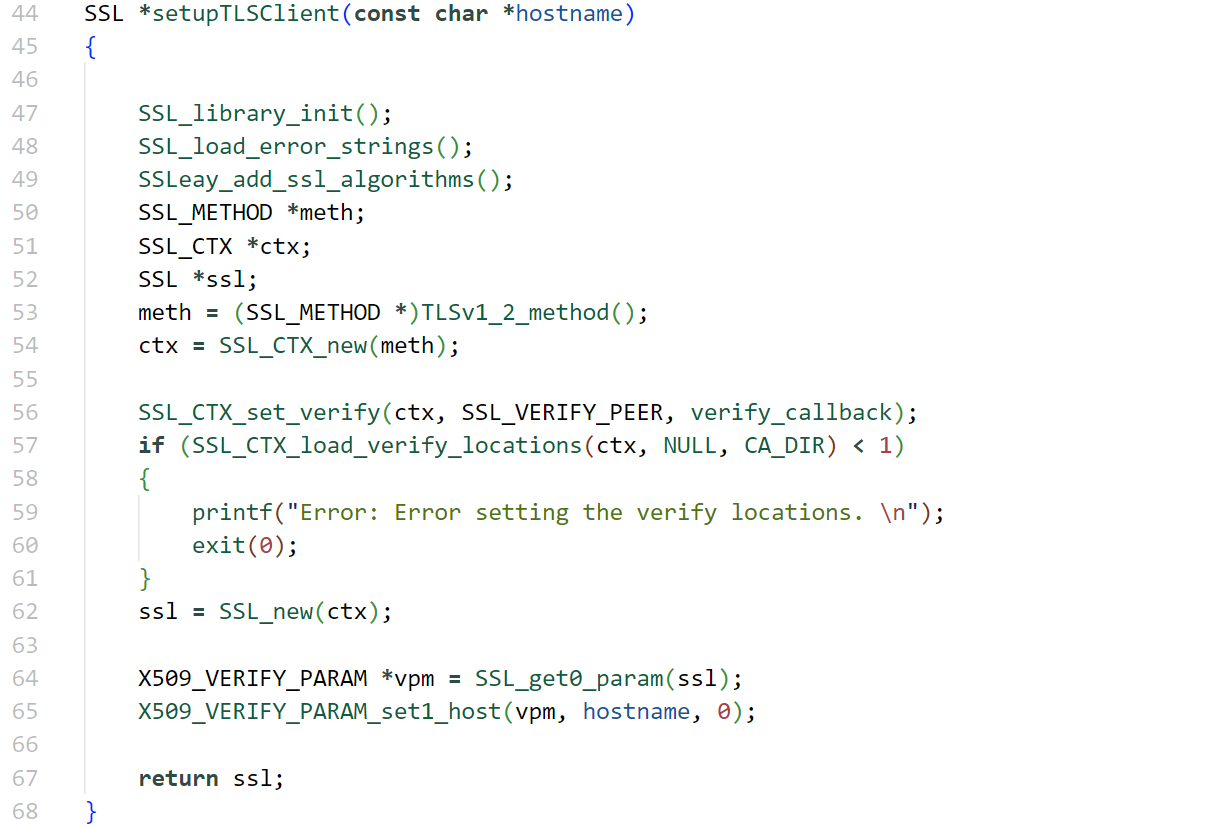


### 客户端设计

1. **客户端认证服务器**

客户端利用 SSL\_CTX\_load\_verify\_locations() 函数实现对服务器身份的验证，该函数通过扫描指定目录"ca\_client"中的证书文件。此函数依据证书文件的哈希值来定位并检索目标证书。随后，客户端使用这些证书数据来执行验证操作。

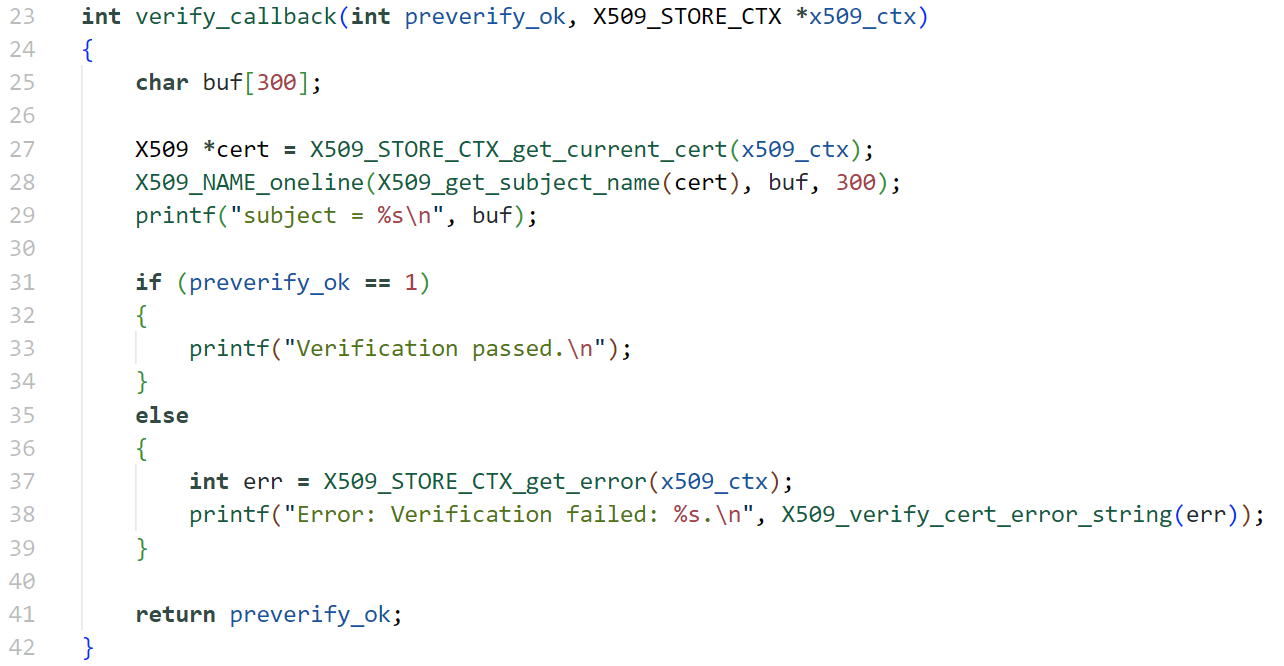
在建立SSL连接的过程中，客户端依据提供的SSL上下文信息来完成验证流程。这一过程确保了服务器的身份得到确认，从而保障了整个连接的安全性和可靠性。通过这种方式，客户端能够验证服务器所提供的证书是否可信，进而确保通信双方之间的数据交换在安全的环境下进行。



在客户端执行服务器认证的过程中，引入了一个回调函数来增强认证的灵活性和兼容性。此回调函数的功能不仅限于展示证书的相关信息，还负责进行深入的证书验证工作。

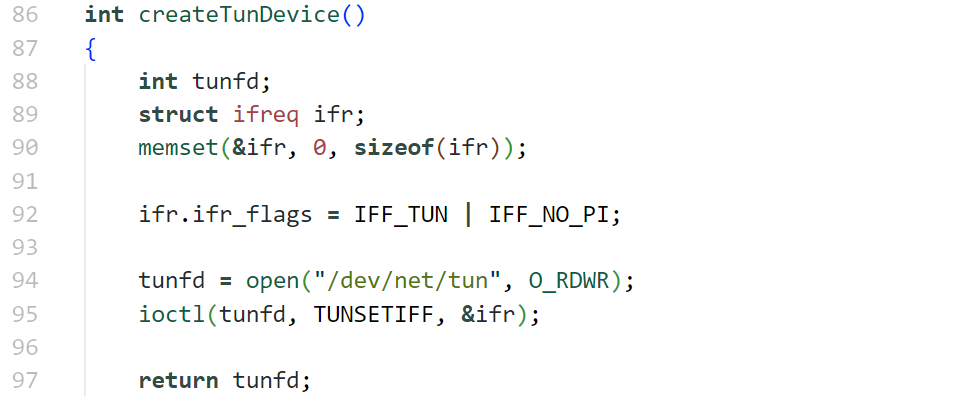
该回调函数在处理证书时，会采用一种更为宽松的策略来应对某些非关键性错误。例如，当遇到主机名与证书中的不完全匹配这类不太严重的错误时，函数将选择忽略这些错误，而不是直接导致认证失败。这种做法允许在确保安全的前提下，提供一定程度的灵活性，使得客户端能够在面对轻微配置差异或预期之外的证书属性时，依然能够完成认证过程。

通过这种设计，客户端在保持对安全标准的严格遵循的同时，也能够适应多样化的部署环境和配置情况，提高了VPN连接的实用性和用户友好性。



1. **TUN创建**

在Linux操作系统中，设备是作为一种特殊的文件类型进行操作的。因此可以通过创建设备目录下的设备来创建Tun0虚拟网卡。



# VPN实现细节

## 实验环境

VMware Workstation 16 Pro 虚拟机。

Ubuntu 16.04 操作系统（SEEDUbuntu16.04）。其中 VPN 网关为 VMware 虚拟机，其余的操作系统均运行在 docker 中

openssl。该软件包含头文件，库函数和命令。该软件包已经安装在上述VM 镜像中。

## 网络配置

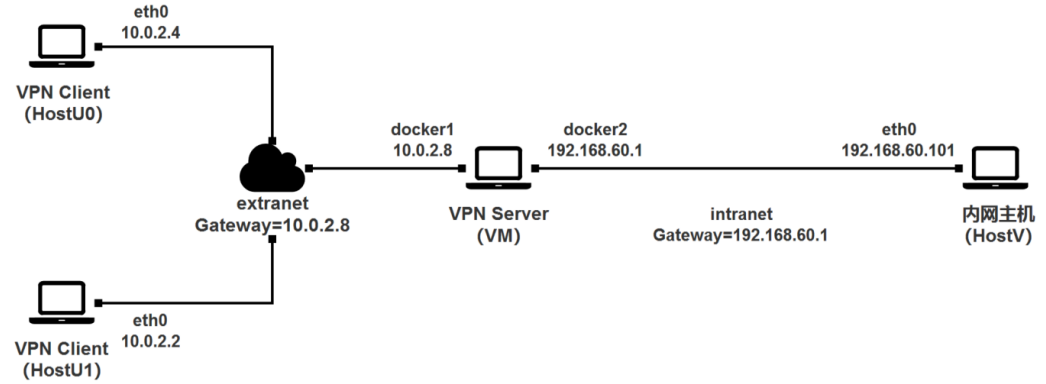


图1-1 实验环境网络拓扑图

root@VM:/$ sudo docker network create --subnet=10.0.2.0/24 --gateway=10.0.2.8 --opt "com.docker.network.bridge.name"="docker1" extranet

root@VM:/$ sudo docker network create --subnet=192.168.60.0/24 --gateway=192.168.60.1 --opt "com.docker.network.bridge.name"="docker2" intranet

sudo docker run -it --name=HostU --hostname=HostU --net=extranet --ip=10.0.2.7 --privileged "seedubuntu" /bin/bash

sudo docker run -it --name=HostV --hostname=HostV --net=intranet --ip=192.168.60.101 --privileged "seedubuntu" /bin/bash

sudo docker run -it --name=HostU2 --hostname=HostU2 --net=extranet --ip=10.0.2.6 --privileged "seedubuntu" /bin/bash

## 

## 问题1

**回答实验指导手册4.2第6步提出的问题：在HostU上，telnet到HostV。在保持telnet连接存活的同时，断开VPN隧道。然后我们在telnet窗口中输入内容，并报告观察到的内容。然后我们重新连接VPN隧道。需要注意的是，重新连接VPN隧道，需要将vpnserver和vpnclient都退出后再重复操作，请思考原因是什么。正确重连后，telnet连接会发生什么？会被断开还是继续？请描述并解释你的观察结果。**

服务器端终止server程序，断开vpn连接，telnet发送无效。重新启动服务器和客户端，telnet生效，且之前没发送出去的命令自动重新发送执行。

而为确保隧道VPN能够正常断开和重新建立连接，必须同时中断服务端和客户端的连接。原因在于，在VPN连接建立时，服务端会启动并等待来自客户端的连接请求。一旦服务端接收到请求，便会初始化服务进程，随后进入持续监听和处理数据包的状态。

若仅断开客户端，服务端由于仍处于数据转发的循环中，将持续监听原连接端口，这会导致它无法响应新的客户端连接请求，从而阻止新客户端建立连接。

相反，如果仅断开服务端，客户端在服务端重启时仍处于原有的隧道处理循环中，由于连接的发送端口保持不变，客户端将无法发出新的连接请求，导致无法重新建立连接。

因此，为了保障VPN服务的连续性和稳定性，服务端和客户端的连接必须同时断开，以便双方均能重新初始化并建立新的连接。

在重新建立隧道连接的过程中，客户端首先向服务端发送连接请求，然后接收并返回服务端的响应数据包。此外，由于telnet服务基于TCP协议，当连接断开后，未发送的数据包会触发TCP的超时重传机制。一旦服务端和客户端重新建立连接，服务端因处于超时重传状态，会发送ACK确认之前未送达的数据包。

客户端在接收到三个冗余ACK后，会触发快速重传机制，重新发送之前未成功传输的报文。服务端在接收到这些重传的报文后，将进行处理并发送响应。客户端根据这些响应完成指令的回显。这一机制使得telnet能够在连接断开后重新连接，并成功回显之前输入的指令，确保了数据传输的一致性和完整性，即使在网络中断或故障恢复后也能维持。

## 问题2

**请介绍你的VPN的登录协议，即在SSL连接建立之后、隧道通信传输之前，SSLVPN客户端与服务器交互了哪些报文？每个报文的格式是怎样的？每个报文的作用是什么？报文内容取值是如何定义的？报文交互的顺序是怎样的？**

在SSL VPN连接建立之初，客户端与服务器之间将进行一系列关键的报文交换，旨在确立双方的连接并为数据传输做好准备。这一过程始于SSL握手，通过此过程，双方交换证书，协商出用于加密通信的对称密钥，确保了通信的安全性与完整性。

完成SSL握手后，客户端向服务器发送一个连接请求。该请求以"Client Hello"报文的形式发出，其目的是表明客户端希望建立VPN连接。在客户端的实现中，通过调用SSL\_write函数，将"Client Hello"字符串发送给服务器，并等待服务器的响应。

服务器在接收到"Client Hello"请求后，会回复一个"Server Hello"响应报文，以纯文本形式确认已接收到客户端的请求，并同意建立连接。服务器端通过类似的SSL\_write调用来发送"Server Hello"。

通过这一系列的报文交换，客户端和服务器能够顺利完成VPN连接的建立，为接下来的隧道通信打下基础。报文的格式简单明了，使用纯文本字符串进行信息传递，易于实现和理解。"Client Hello"和"Server Hello"报文的交互，不仅确立了连接请求和确认的机制，而且为VPN连接的建立和后续数据传输的安全性与稳定性提供了保障。

整个交互过程体现了SSL VPN连接建立的高效性和可靠性，通过简洁的报文格式和明确的通信协议，确保了客户端和服务器能够顺利地建立起VPN连接。

## 问题3

**你的VPN服务器支持多客户端采用的什么技术？VPN服务器收到保护子网主机的应答报文时，如何判断应发送给哪个VPN客户端的隧道？**

采用的是命名管道技术，由此VPN服务器实现了子进程间的高效通信。每个子进程代表一个客户端的SSL连接，并拥有一个与之对应的、唯一命名的管道文件。当主进程在tun0接口接收到来自内核的IP数据包时，它根据数据包的目的IP地址，将IP包转发至相应的管道文件，实现进程间的通信。

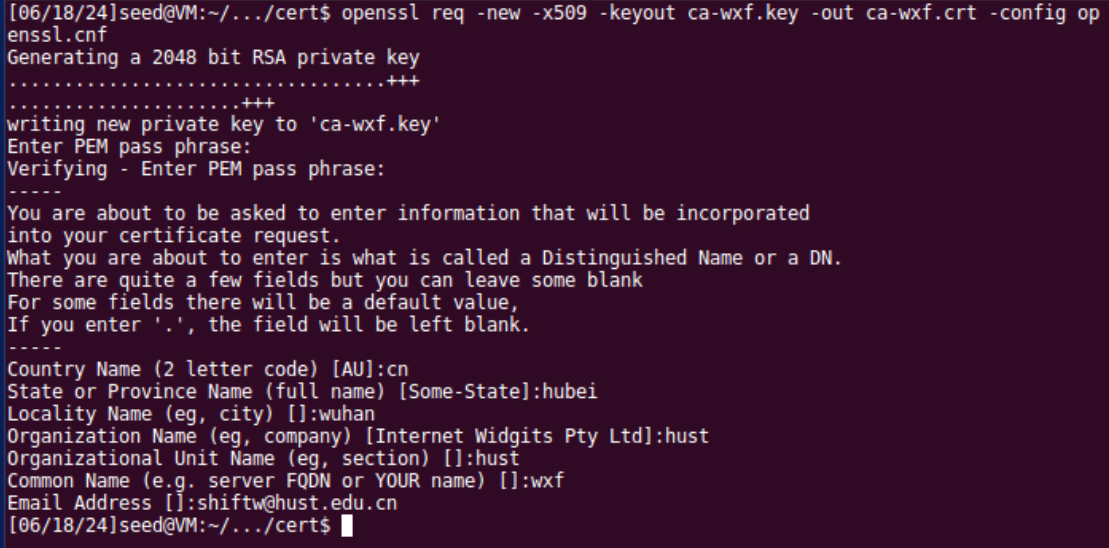
在VPN服务器端，一旦客户端登录成功，服务器便为该客户端分配一个专用的内网IP地址，并基于此IP创建一个FIFO文件。当服务器收到来自保护子网主机的应答报文，它会解析报文的目的IP地址——即之前分配给客户端的内网地址。随后，服务器打开对应IP地址命名的管道，将应答报文写入其中。负责该客户端通信的子线程会持续监听其管道，一旦检测到报文，便通过SSL连接将其转发给客户端。

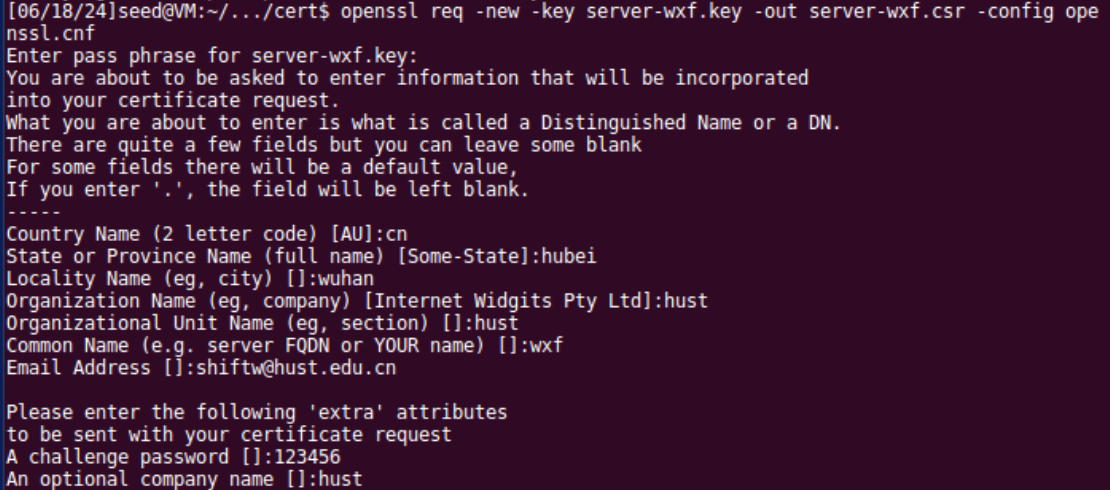
通过这种方式，VPN服务器不仅确保了数据传输的准确性和效率，而且通过为每个客户端分配独立的通信管道，增强了系统的可扩展性和安全性。

# 测试结果与分析

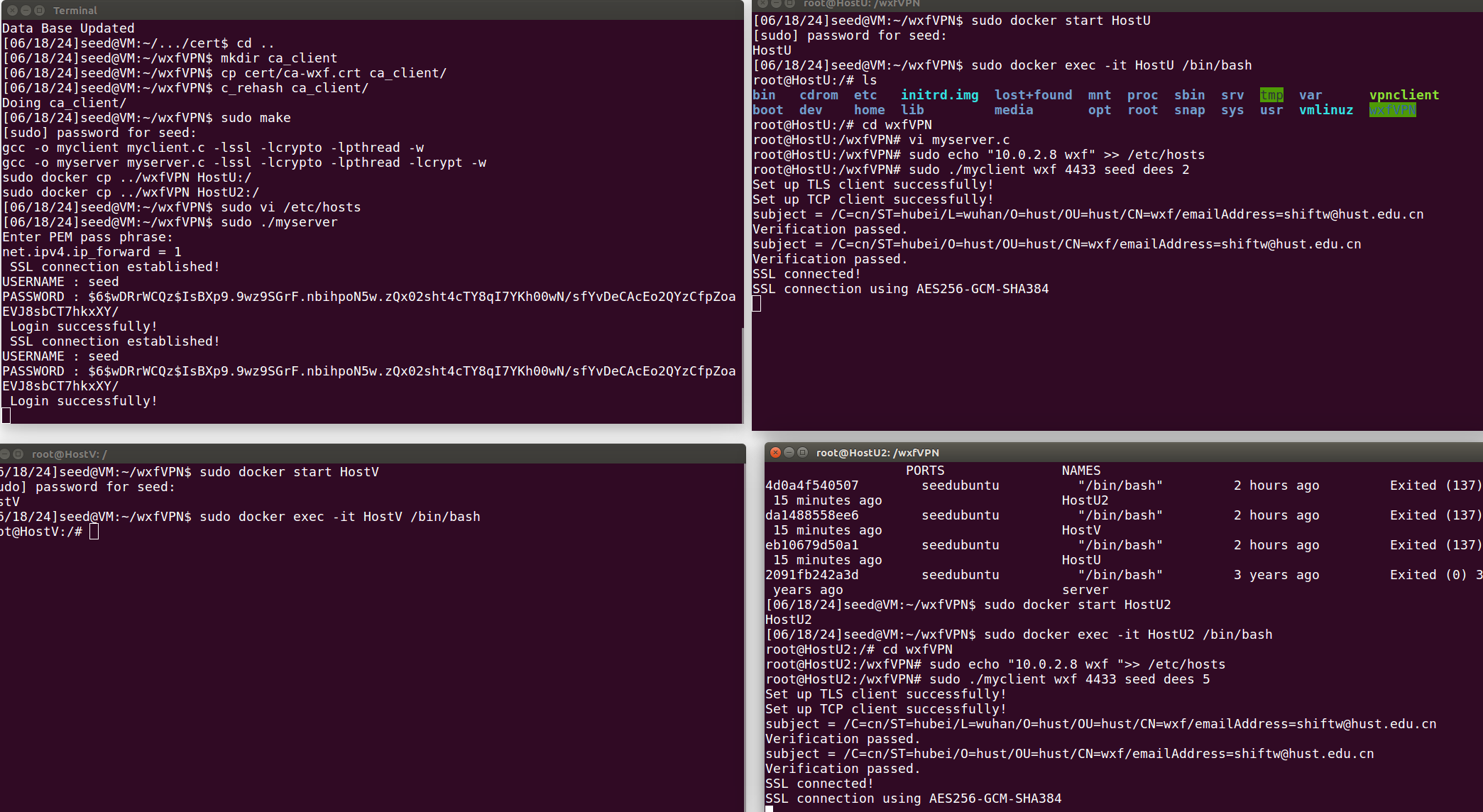
## 认证VPN服务器

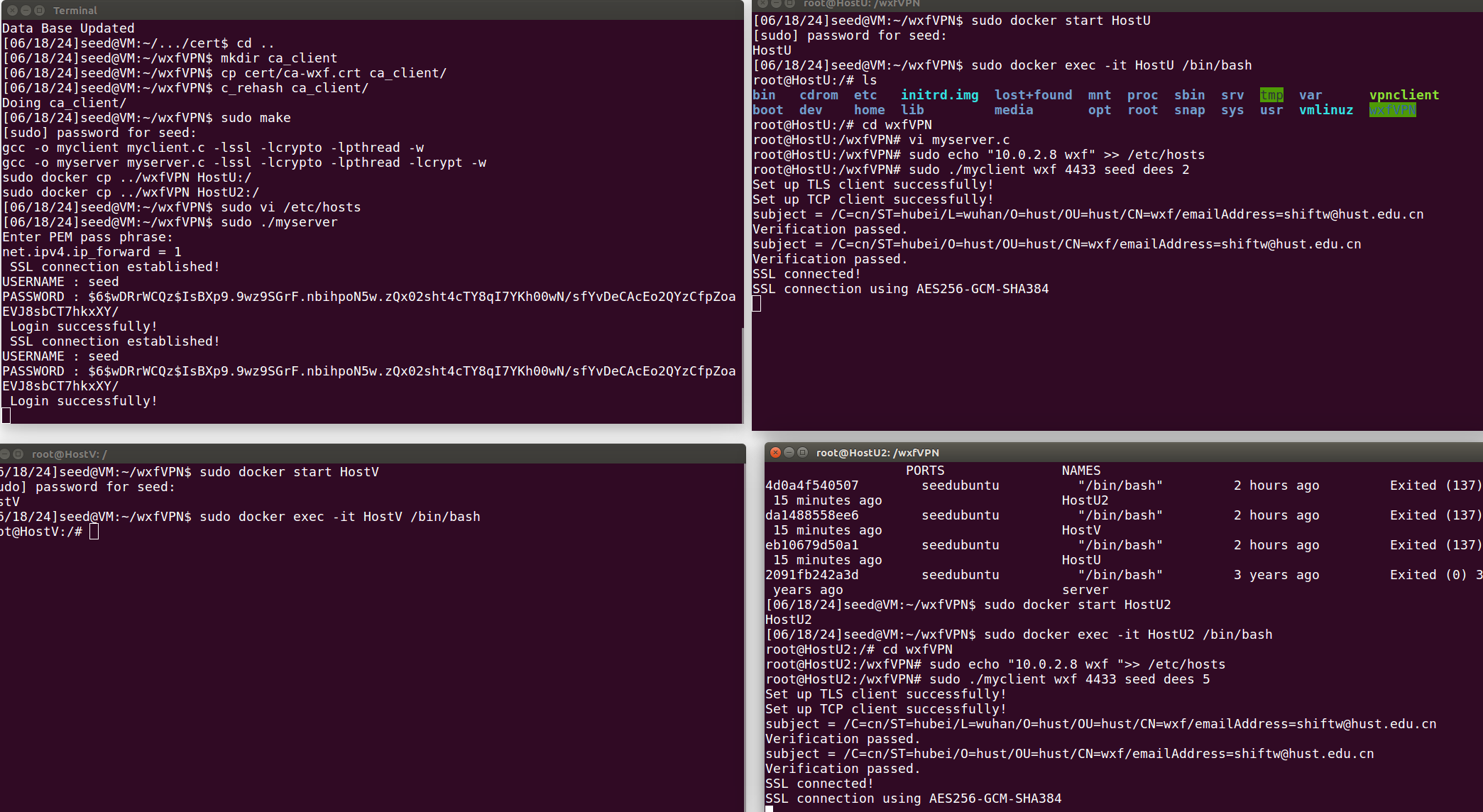
使用openssl生成证书的过程如下图，可以看到分别设置了国家、省份、城市、机构、姓名、邮箱和密码。



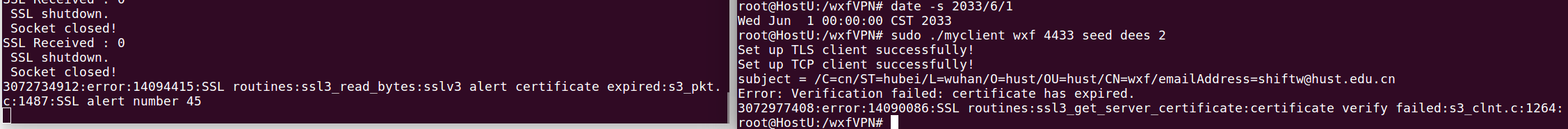


VPN服务器启动，然后打开新终端进入HOSTU的命令行，启动VPN客户端，可以看到服务器端返回登录成功的信息，而客户端返回SSL连接成功的信息，并显示服务器端的证书信息，和之前创建的设置是一致的。

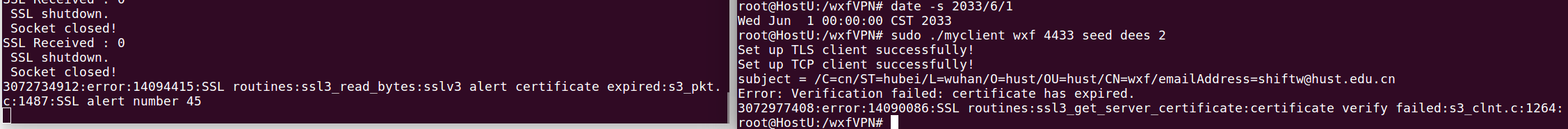




断开客户端的连接，将客户端的时间更新至2033年，由此修改客户端主机时间到VPN服务器证书有效时间之后，然后重新启动客户端连接，客户端界面显示证书信息并提示服务器的证书过期！客户端取消连接直接退出。然后将客户端时间改正回来，重新登录则成功。

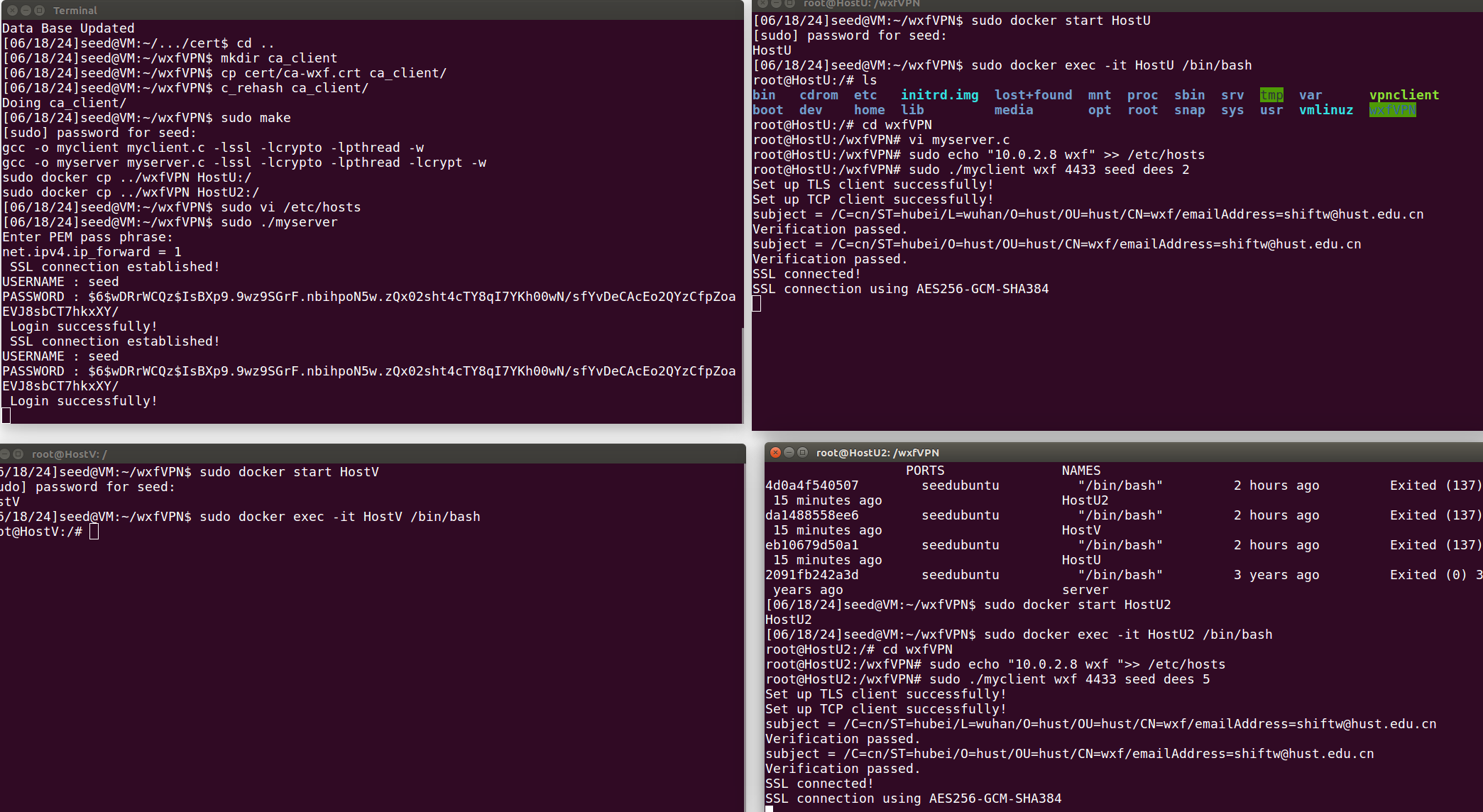


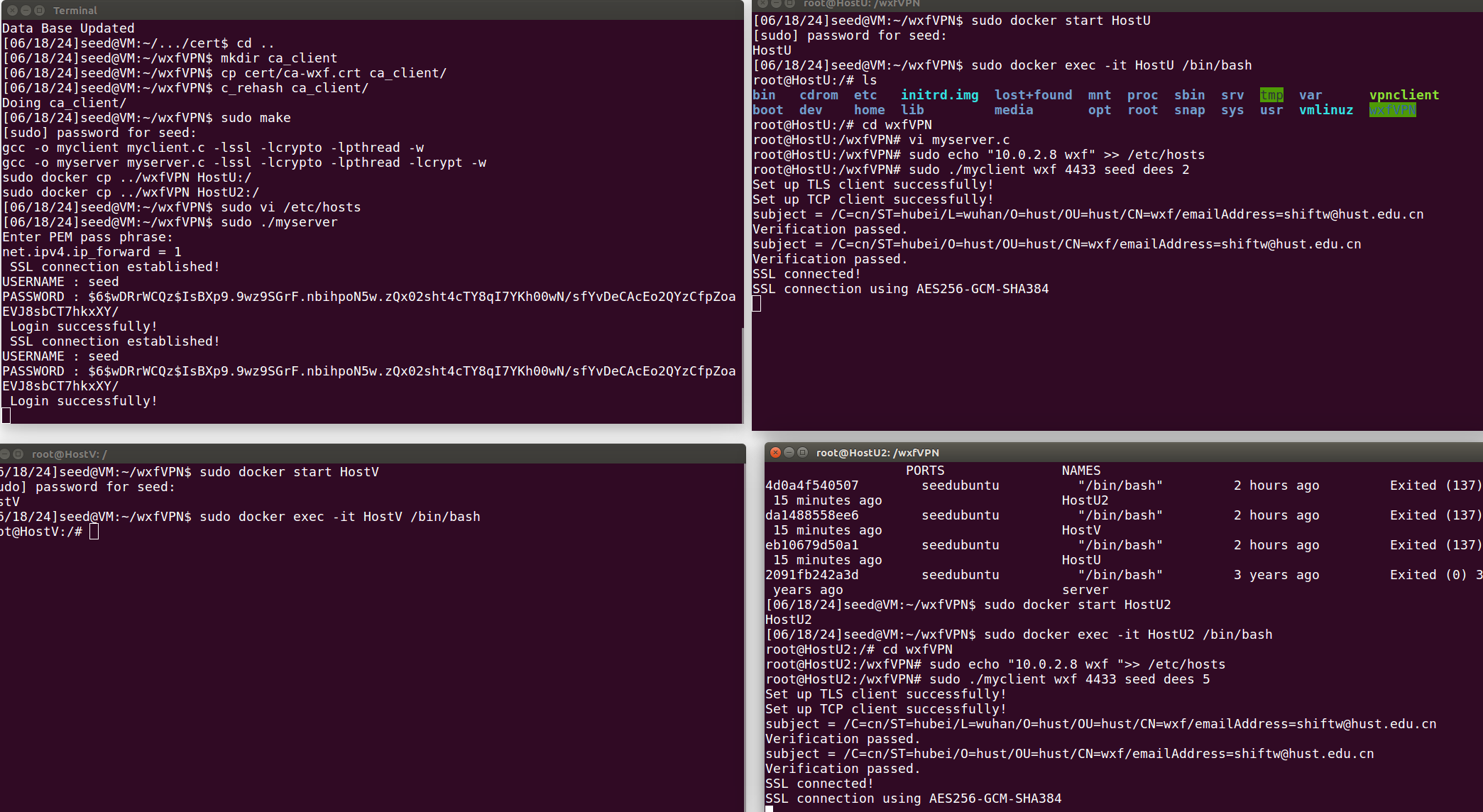
服务器端同样返回证书过期的提示，证明对服务器端的认证设计成功！



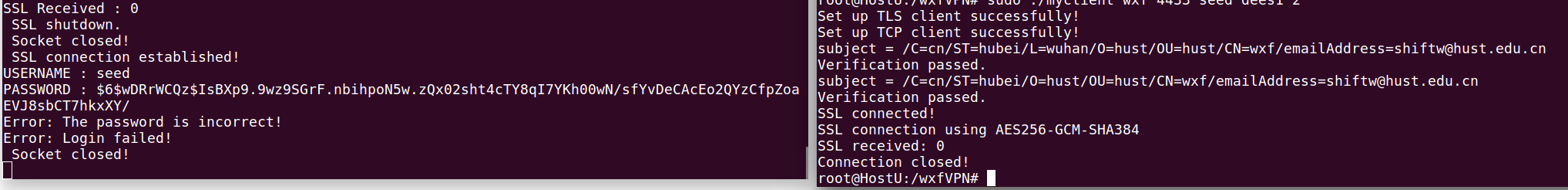
## 认证VPN客户端

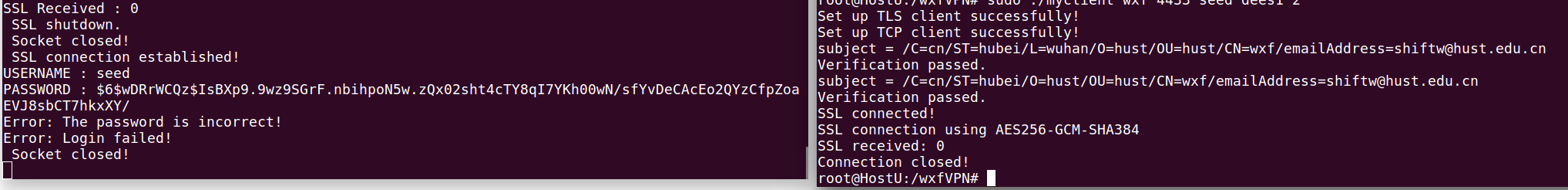
当VPN客户端登录密码正确时，可以正常进行登录，服务器端显示登录成功！





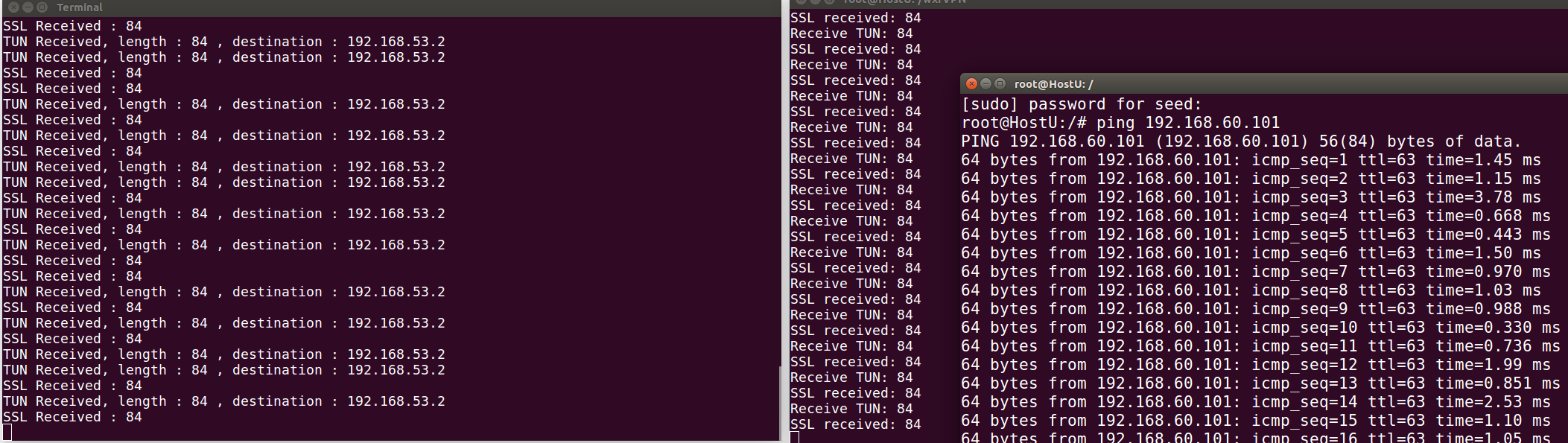
断开客户端，使用错误的口令密码重新尝试登录，发现登录失败！客户端的SSL连接关闭，服务器端返回提示信息：密码不正确，登录失败！可见对客户端的身份认证设计成功。

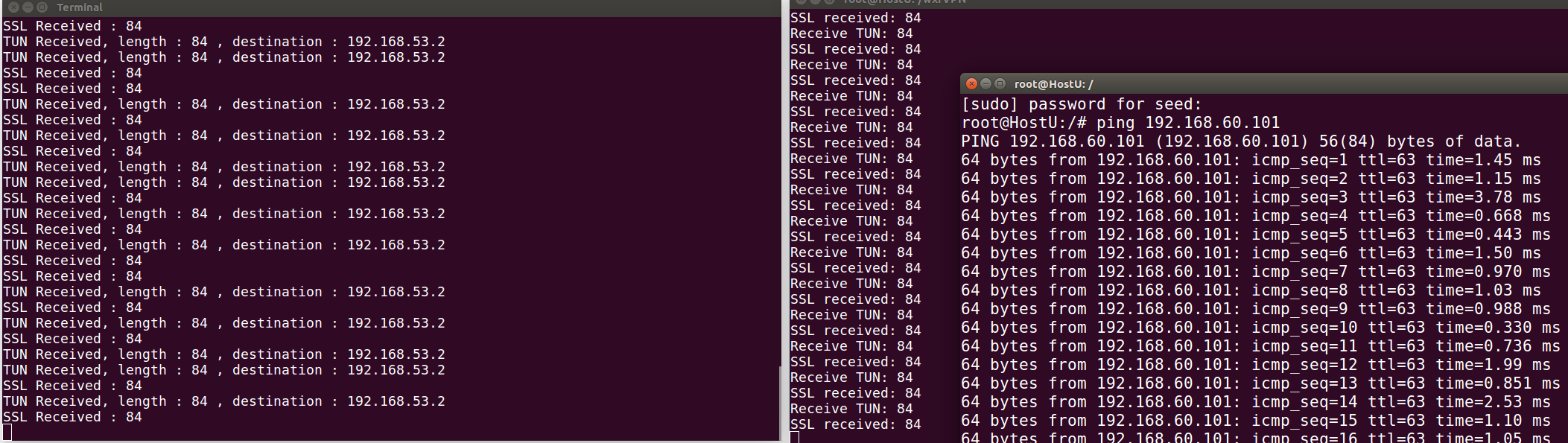




## 加密隧道通信

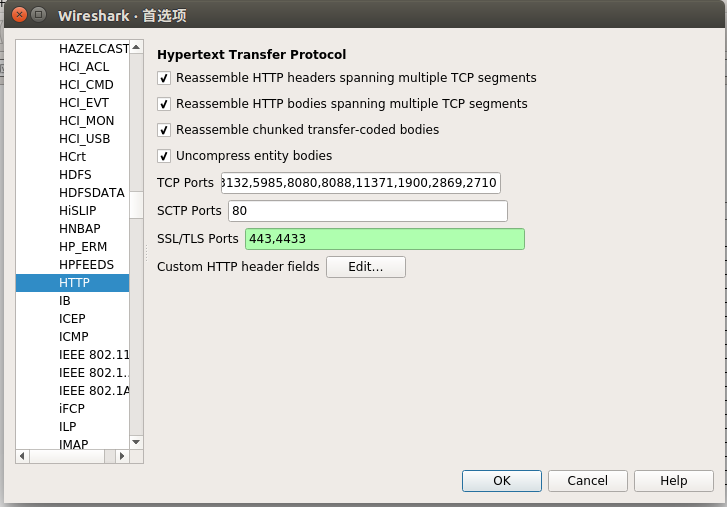
HOSTU启动VPN客户端，然后新建终端ping内网主机192.168.60.101，如下所示，可见成功ping通，返回数据信息。

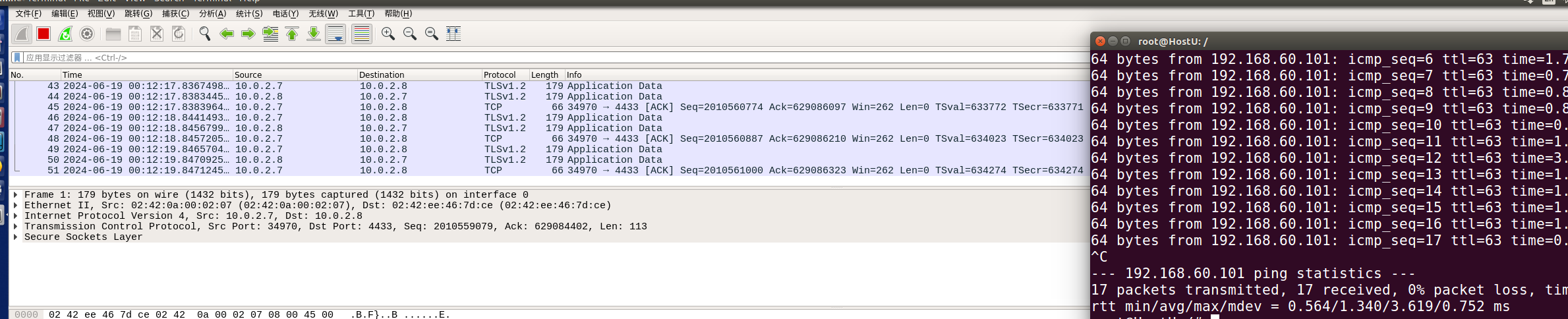




可以看到客户端显示SSL连接和接收数据流信息，而服务器端也显示对应的接收信息。

下一步打开wireshark，设置HTTP中对4433和443端口的监听，标记为SSL/TLS协议。然后启动对docker1的监听，发现wireshark抓到若干TLS数据包。



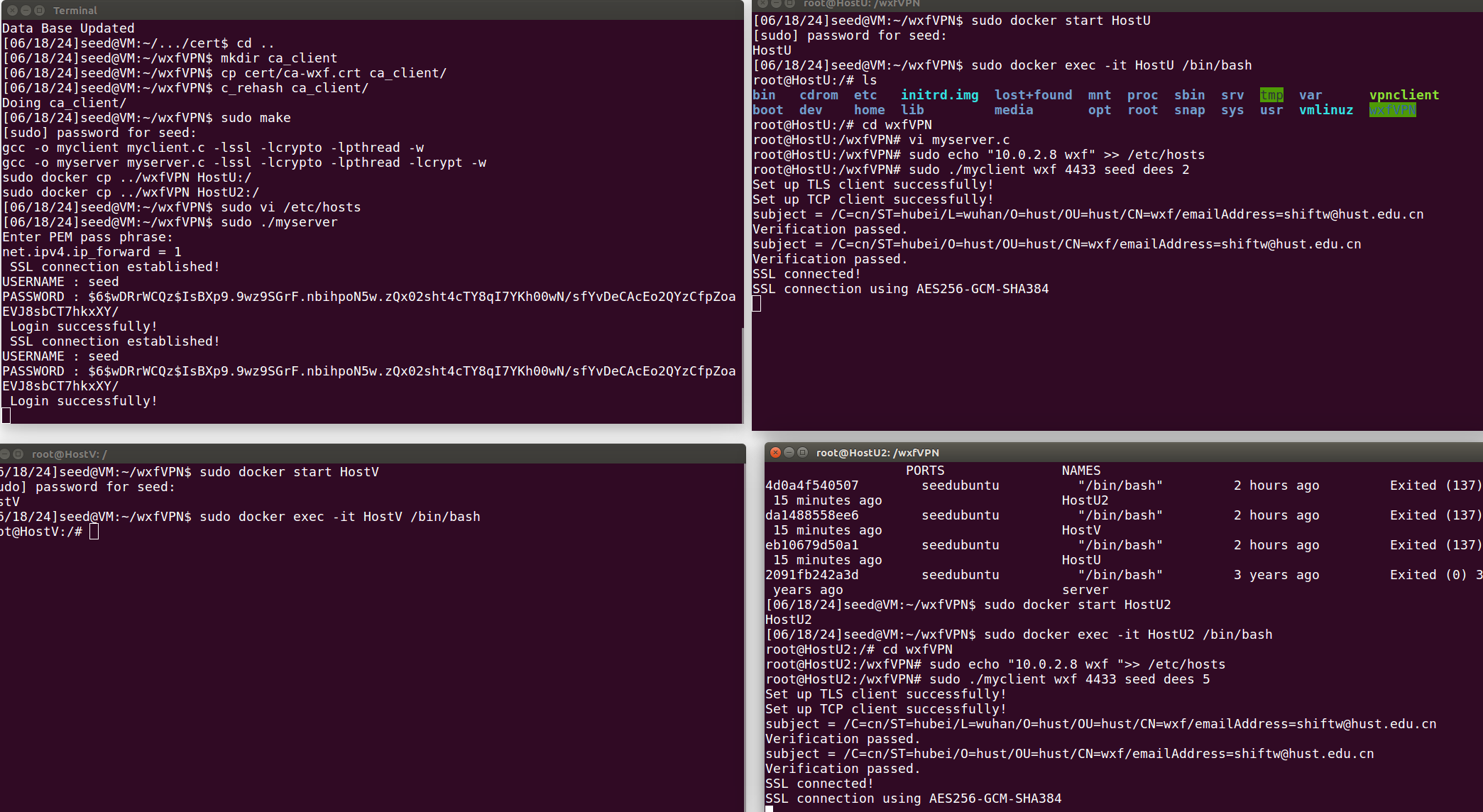


查看数据包已经经过了隧道封装，协议为TLS，说明加密算法设计成功！

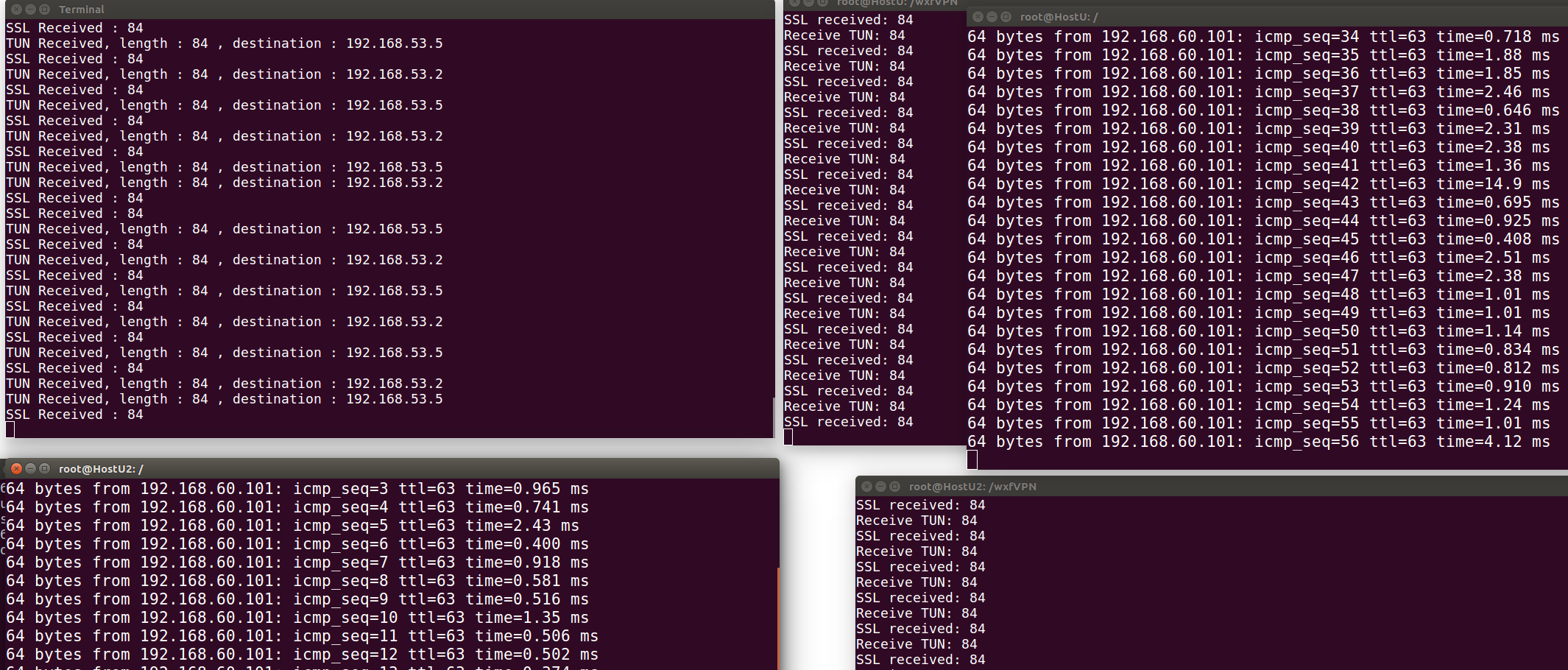
## 支持多客户端

首先测试VPN能否同时支持多个客户端登录。

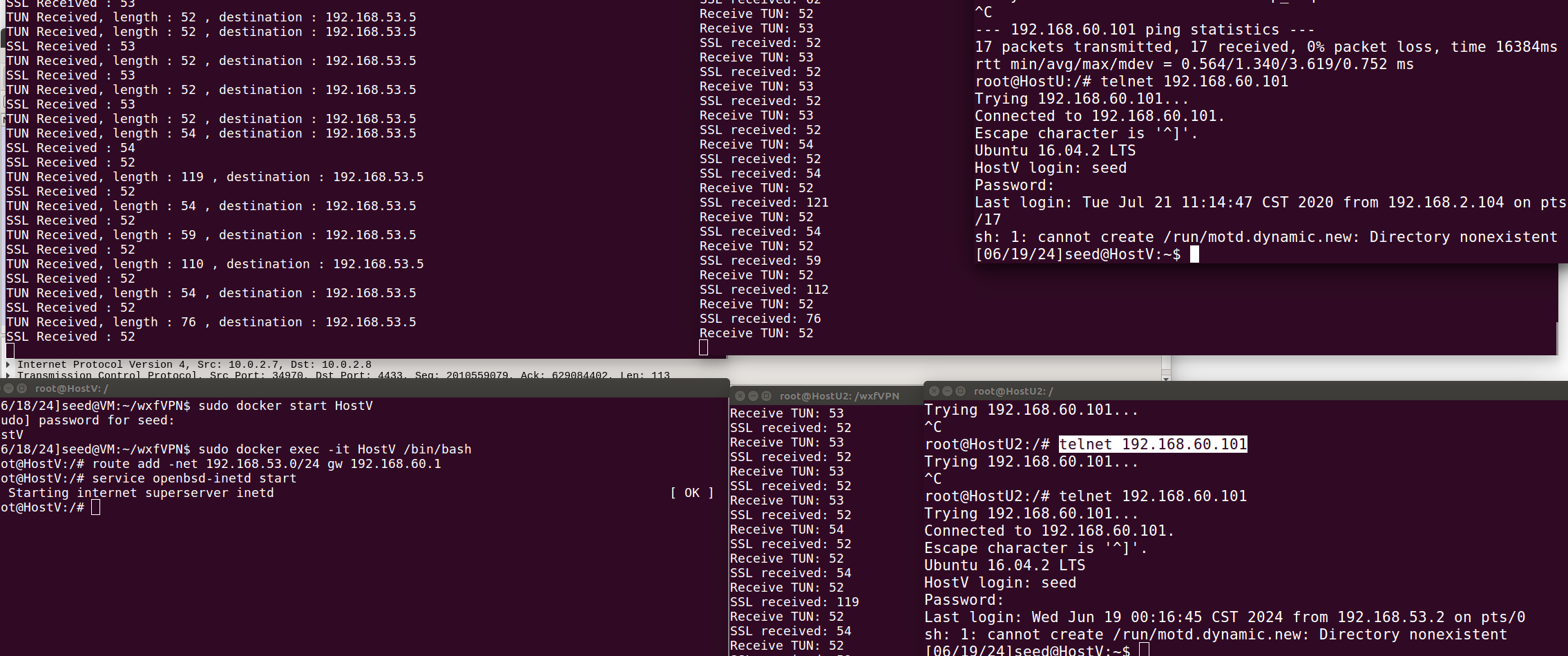
创建两个的外网主机docker HOSTU和HOSTU2并开启，vm网关启动VPN服务器后，先后启动两个VPN客户端，可以看到服务器端显示两个客户端均能正常连接且返回证书信息，说明VPN支持多个客户端同时登录。



下一步让两个外网主机HostU和HostU2都对内网主机发送ping请求，可以看到服务器能同时接收到192.168.53.2和192.168.53.5虚拟IP的数据包，这分别是两台外网主机发送的，说明服务器的并发性良好。



下一步测试telnet连接，两台外网主机HostU和HostU2均使用telnet登录同一台内网主机HOSTV，输入账号密码后回车，可以看到命令行的标头已经变成了HOSTV，说明登录成功。而且两者均能正常发送远程命令控制内网主机！

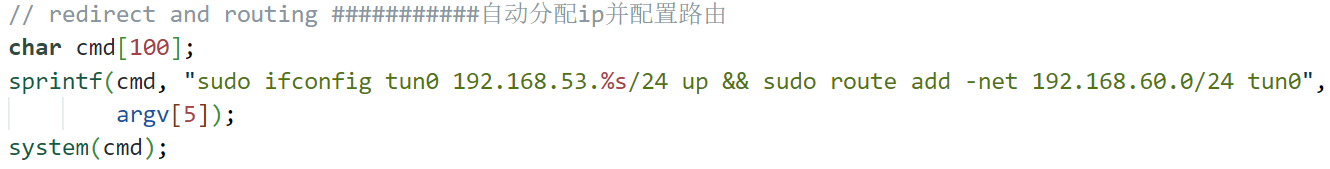


当断开其中一个客户端的telnet连接时，另一个客户端的隧道通信仍能保持连接和数据发送，说明VPN对多用户并发的支持良好！

## 易用性和稳定性

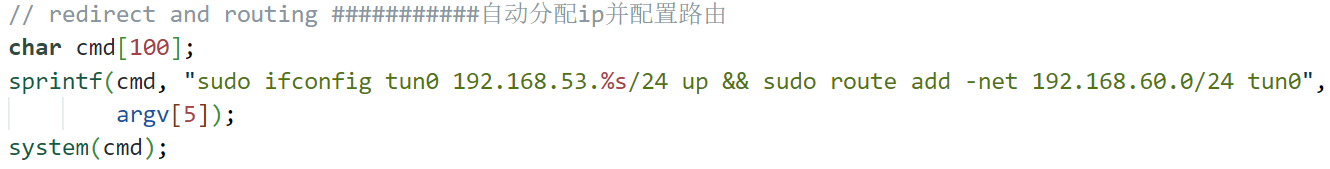
1. VPN客户端虚拟IP获取

采用半自动分配，由用户客户端设置虚拟ip的最后一部分，然后自动生成192.168.60.0/24网段的虚拟地址，最多可以分配253个地址。



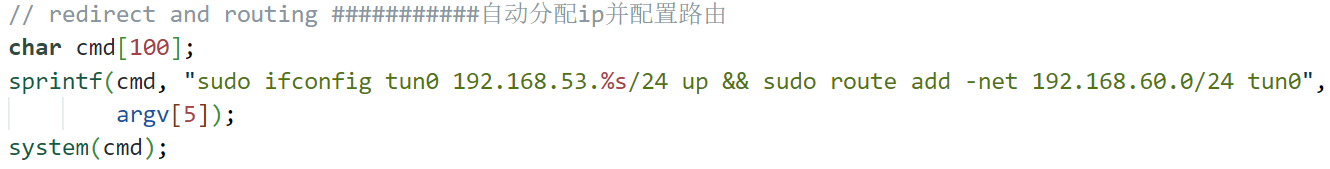
1. VPN客户端虚拟IP配置

自动配置tun0接口为分配的IP地址，直接一键运行客户端即可，无需手动配置



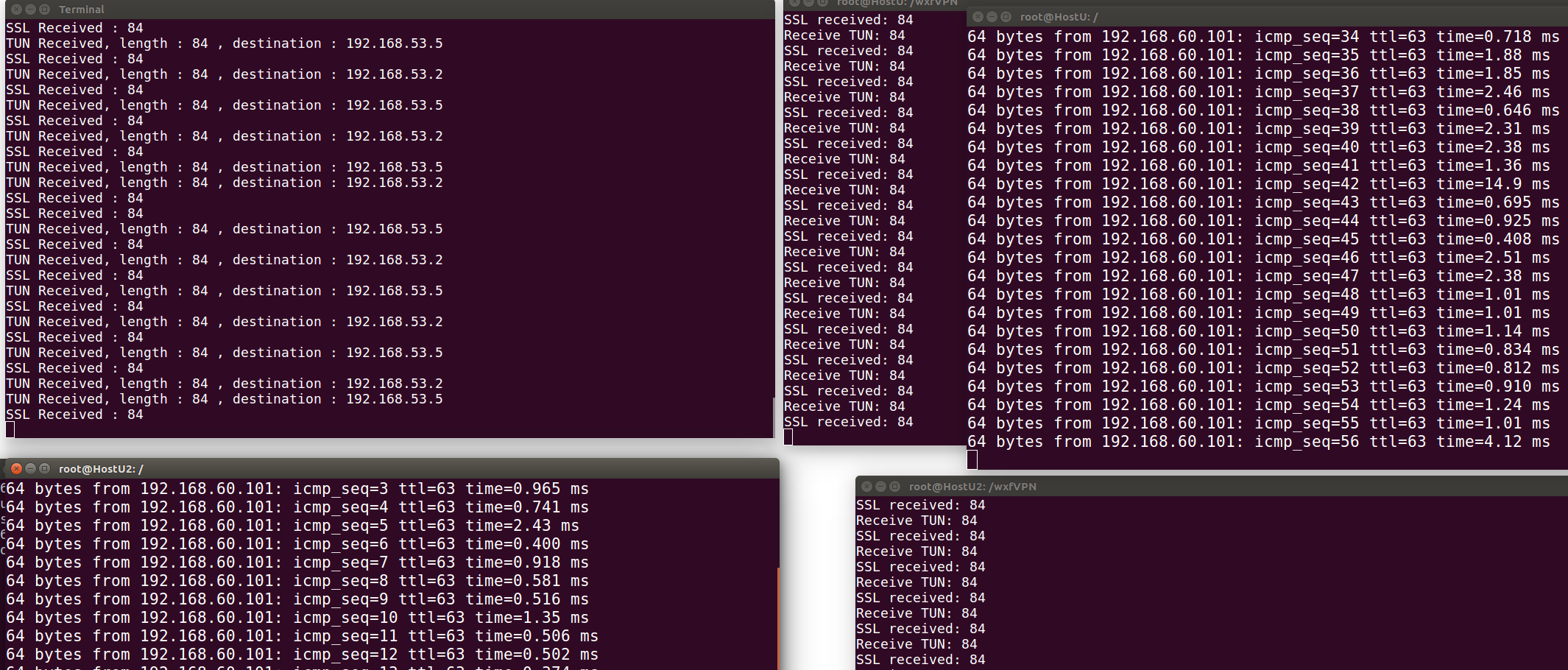
1. VPN客户端内网路由配置

自动添加tun0的路由，客户端启动即可自动配置，无需手动。



1. 正常使用时的稳定性：

支持多客户端连接，且响应稳定迅速。



# 体会与建议

## 心得体会

本次实验让我对VPN的具体实现有了更深的理解，而不是只停留在理论知识层面。通过编程实现一个简单的miniVPN，我了解了VPN的技术原理，如何通过建立隧道实现内网与外网主机的通信，又如何对通信进行加密和认证等等。这些让我对VPN数据的流通有了进一步的认识，也学习了如何使用C语言代码进行实现。除此之外，在解决多客户端问题时，还运用到了Linux多线程与select函数多路复用的知识，是在linux实现上对操作系统知识的又一步巩固，总体上来说收获颇丰。

从利用Docker构建网络架构、生成证书，到应用TUN/TAP虚拟网络设备、shadow认证、管道技术、select机制等实现多客户端VPN，指导材料均提供了详尽的说明和示例代码，为初学者提供了清晰的学习路径。

而且实验中的一些问题让学生多了很多思考空间。例如，在问题一中探讨的telnet连接恢复问题，通过指导书的最后图表，我理解了TCP的可靠连接恢复机制，从而加深了对服务工作原理的认识。

在实验过程中，我还亲身实践了使用OpenSSL库创建CA（证书颁发机构）的过程，从理论到实践的转变使我体验了整个证书申请、签名和验证的流程。此外，通过使用Wireshark追踪加密流量，我在网络工具的应用上变得更加熟练。

在编写多进程VPN服务器代码的过程中，我面临了并发处理和资源管理的挑战。通过运用PIPE技术，我实现了父子进程间的通信。最终，我没有直接使用指导书中的pipe()函数创建管道，而是在理解了管道原理后，采用了mkfifo()进行命名管道的创建，掌握了一种新的进程间通信技术。这一过程不仅加深了我对进程间通信机制的理解，也锻炼了我的编程逻辑能力。我认识到，高效的资源分配和错误处理机制对于保持服务器的稳定性至关重要。

实验中的代码部分与示例代码相似，这是在实现VPN程序时不可避免的现象，因为重点在于功能的实现而非算法创新。许多操作，如虚拟设备的创建和shadow口令认证，都是按照相对固定的步骤完成的。在这部分，我采用了借鉴、理解、掌握的学习方式。

总体而言，通过深入研究代码，我对VPN通信机制及其应用技术有了更具体和深刻的认识。学习过程激发了我的兴趣，我期待未来能够更深入地研究现有的VPN软件。

## 意见建议

运行示例代码的时候问题较多，建议总结一下每一届常出现的问题做个合集，给出解决方案和提示，避免同学们花费太多时间。