

“计算机网络安全”实验报告

**题目： VPN实验**

院 系 网络空间安全学院

专业班级 信安1805班

姓 名 吴 锦 添

学 号 U201810398

日 期 2022 年 5 月 30

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 评分项 | 实验报告评分  （40%） | 检查单分数  （60%） | | 综合得分 | 教师签名 |
| 得分 |  | |  |  |  |

**实验报告评分标准**

|  |  |  |  |
| --- | --- | --- | --- |
| **评分项目** | **分值** | **评分标准** | **得分** |
| 实验原理 | 20 | 18-20系统流程清晰，报文处理过程描述清楚；  15-17系统流程比较清晰，报文处理过程描述比较清楚；  12以下 描述简单 |  |
| 实验步骤 | 30 | 25-30 实验步骤描述详细、清楚、完整，前后关系清晰；  18-24 实验步骤描述比较清楚，关键步骤都进行了描述  18分以下，实验步骤描述比较简单或不完整 |  |
| 结果验证与分析 | 20 | 16-20，任务完成，针对任务点的测试，对结果有分析  10-15，针对任务点的测试截图，没分析  10分以下，测试很简单，没有覆盖任务点 |  |
| 心得体会 | 10 | 8-10有自己的真实体会  4-7 真实体会套话  3分以下，没有写什么体会 |  |
| 格式规范 | 10 | 图、表的说明，行间距、缩进、目录等，一种不规范扣1分 |  |
| 实验思考 | 10 | 思考题的回答，以及其它的简介 |  |
| **总 分** | | |  |

**目 录**

[实验三 VPN实验 1](#_Toc73540815)

[1 实验目的 1](#_Toc73540816)

[2 实验环境 1](#_Toc73540817)

[3实验内容 2](#_Toc73540818)

[4 实验步骤及结果分析 3](#_Toc73540819)

[5 实验思考 19](#_Toc73540820)

[心得体会与建议 21](#_Toc73540821)

[1 心得体会 21](#_Toc73540822)

[2 建议 22](#_Toc73540823)

# 实验三 VPN实验

## **1 实验目的**

本实验的学习目标是掌握 VPN 的网络和安全技术。为实现这一目标，要求实现简单的 TLS/SSL VPN。虽然这个 VPN 很简单，但它包含了 VPN 的所有基本元素。TLS/SSL VPN 的设计和实现体现了许多安全原则，包括以下内容：

• 虚拟专用网络

• TUN/TAP 和 IP 隧道

• 路由

• 公钥加密，PKI 和 X.509 证书

• TLS/SSL 编程

• 身份认证

## **2 实验环境**

**实验机器的操作系统：**

VMware Workstation 虚拟机。

Ubuntu 16.04 操作系统（SEEDUbuntu16.04）。

**实验的网络配置：**

使用VMware Workstation虚拟机的NAT模式。

实验中所需的内网、外网，在相应的docker中进行网络配置即可。

**系统软件组件：**

本次实验需要使用 openssl 软件，该软件包含头文件，库函数和命令。该软件包已经安装在我们上述 VM 镜像中。

**网络拓扑：**

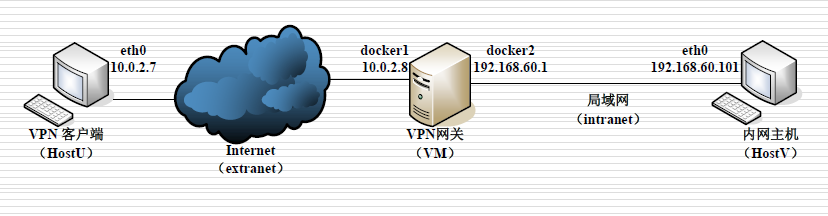


图2.1 网络拓扑图

## **3实验内容**

**任务一：建立主机到主机的隧道**

1. 在计算机（客户端）和网关之间创建 VPN 隧道，允许计算机通过网关安全地访问专用网络。我们使用虚拟机本身作为 VPN 服务器网关（VM），并创建两个容器分别作为 VPN 客户端（HostU）和专用网络中的主机（HostV）。
2. 同时使用指令配置内网和外网网段和相应的网关IP，通过这种设置，HostV 不能直接从Internet 访问，即不能直接从HostU 访问。
3. 随后创建TUN虚拟网络接口，当程序从TUN接口读取数据时，计算机发送到此接口的IP 数据包将被传送给程序，程序可以使用标准的read()和write()系统调用来接收或发送数据包到虚拟接口。
4. 最后再配置相对应的路由，当发送数据时根据相应的路由进行转发。

**任务二：实现隧道的加密**

我们已经创建了一个IP 隧道，但是我们的隧道没有受到保护。只有在我们保障了这个隧道的安全之后，才能将其称为VPN 隧道。这就是我们在这项任务中要实现的目标。为了保护这条隧道，我们需要实现两个目标，即机密性和完整性。使用加密来实现机密性，即通过隧道的内容将被加密。完整性目标确保没有人可以篡改隧道中的流量或发起重放攻击。使用消息验证代码（MAC） 可以实现完整性。可以使用传输层协议（TLS）实现这两个目标。

**任务三：认证VPN 服务器**

在建立VPN 之前，VPN 客户端必须对VPN 服务器进行身份认证，确保服务器不是假冒的服务器。

VPN 服务器需要首先从证书颁发机构（CA）（例如Verisign）获取公钥证书。当客户端连接到VPN 服务器时，服务器将使用证书来证明它是客户端预期的服务器。Web 中的HTTPS 协议使用这种方式来认证Web 服务器，确保客户端正在与预期的Web 服务器通信，而不是伪造的Web 服务器。

**任务四：认证VPN 客户端**

在建立VPN 之前，VPN 服务器必须认证客户端（即用户），确保用户具有访问专用网络的权限。当用户尝试与 VPN 服务器建立 VPN 隧道时，将要求用户提供用户名和密码。服务器将检查其影子文件（/etc/shadow）；如果找到匹配的记录，则对用户进行认证，并建立 VPN 隧道。如果没有匹配，服务器将断开与用户的连接，因此不会建立隧道。

**任务五：支持多客户端**

真实应用中，一个 VPN 服务器通常支持多个 VPN 隧道。也就是说，VPN服务器允许多个客户端同时连接到它，每个客户端都有自己的 VPN 隧道（从而有自己的 TLS 会话）。MiniVPN 应该支持多个客户端。

## **4 实验步骤及结果分析**

#### 4.1 建立主机到主机的隧道

**1. 配置环境**

在计算机（客户端）和网关之间创建VPN 隧道，允许计算机通过网关安全地访问专用网络。使用虚拟机本身作为VPN 服务器网关（VM），并创建两个容器分别作为VPN 客户端（HostU）和专用网络中的主机（HostV）。网络设置如图4.1 所示：



图4.1 网络设置图

VPN 客户端（HostU）和 VPN 服务器网关的外网口通过Internet 连接，在实验中将这两台机器直接连接到同一docker网络“extranet”，模拟 Internet。HostV 是内部局域网的计算机。Internet主机 HostU 上的用户希望通过 VPN 隧道与内部局域网的主机 HostV 通信。为模拟此设置，我们使用 docker 网络“intranet”将 HostV 与 VPN 服务器网关的内网口连接，模拟内部局域网。

在VM 上创建docker 网络intranet和extranet，使用指令如下，执行后的路由表中出现了相应的接口和网关如图4.2所示：

|  |
| --- |
| 在 VM 上创建 docker 网络 extranet  $ sudo docker network create --subnet=10.0.2.0/24 --gateway=10.0.2.8 --opt "com.docker.network.bridge.name"="docker1" extranet  在 VM 上创建 docker 网络 intranet  $ sudo docker network create --subnet=192.168.60.0/24 --gateway=192.168.60.1 --opt "com.docker.network.bridge.name"="docker2" intranet |

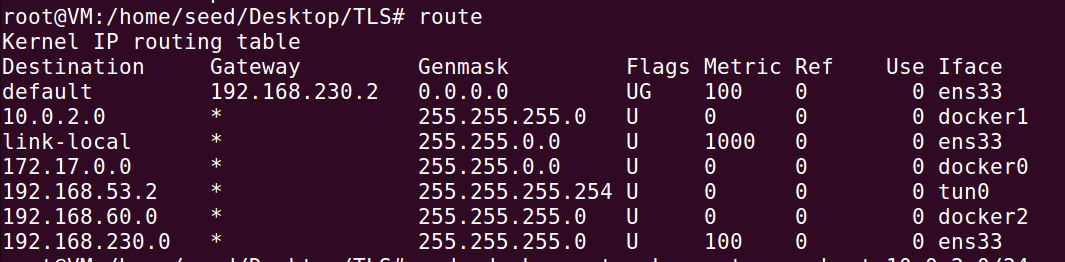


图4.2 路由表

在VM 上新开一个终端，创建并运行容器HostU ，同样的方式创建HostV，使用docker ps 查看创建的容器：

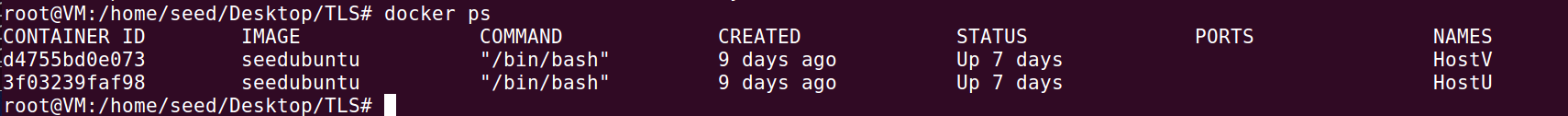


图4.3 docker创建

避免默认路由的干扰，在容器HostU 和HostV 内分别删除掉默认路由，如图4.3所示：

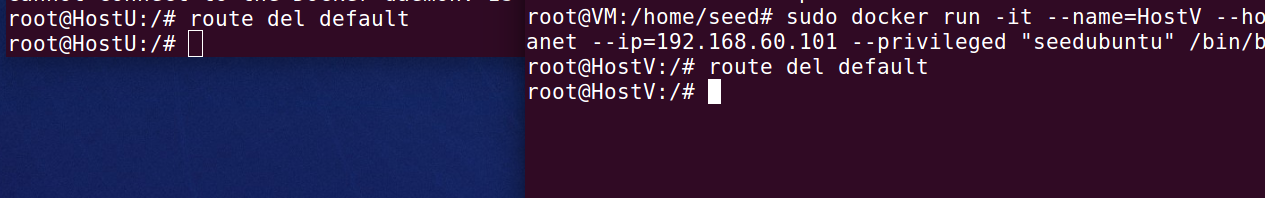


图4.3 删除默认路由

**2. 使用TUN技术创建隧道**

用户空间程序通常访问 TUN 虚拟网络接口。操作系统通过 TUN 网络接口将数据包传送到用户空间程序。另一方面，程序通过 TUN 网络接口发送的数据包被注入操作系统网络栈，在操作系统看来，数据包是通过虚拟网络接口的外部源进来的。程序可以使用标准的read()和 write()系统调用来接收或发送数据包到虚拟接口。隧道的拓扑如图所示：

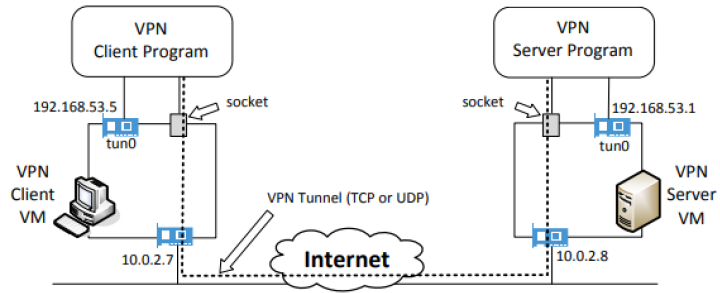


图4.4 隧道连接拓扑

在VM 上运行VPN 服务程序，在程序的createTunDevice()函数中使用tunfd = open("/dev/net/tun", O\_RDWR) ，打开了tun设备，同时tun的编号是系统内核自动顺序分配的，在后续实现多客户端时创建多tun实现中因为没注意这出现过问题。创建tun的具体代码如下：

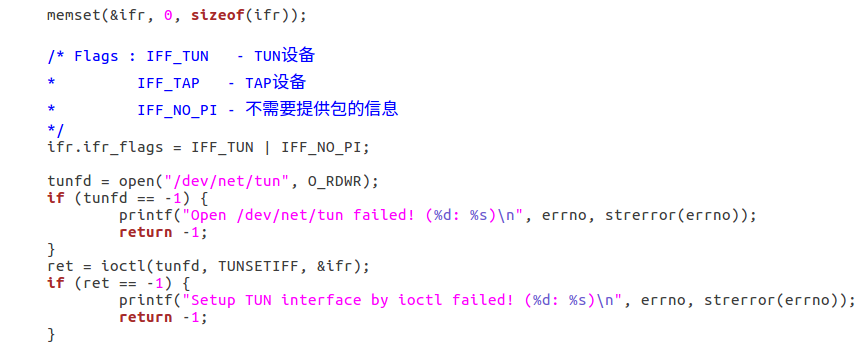


图4.5 tun创建

在启动server程序后，tun已经被创建，正在等待客户端的连接，但这个时候即使运行了客户端建立连接，也无法完成通信。因为tun没有被激活，路由也没有配置。

使用指令sudo ifconfig tun0 192.168.53.2/31 up来激活在server程序中创建的虚拟网卡并配置虚拟IP：



图4.6 虚拟网卡的路由

除非特别配置，否则计算机将仅充当主机，而不充当网关。VPN Server 需要在内网和隧道之间转发报文，因此需要作为网关。我们需要为计算机启用IP 转发，使其行为类似于网关。由于VM 上的iptables 规则可能阻断转发报文，我们还需要清除iptables 规则。指令如下：

|  |
| --- |
| sudo sysctl net.ipv4.ip\_forward=1  sudo iptables -F |

在主机HostU上运行的客户端程序创建了tun，也需要对该tun分配虚拟IP同时激活。但是仅仅完成和服务端一样的配置是不能通过隧道进行通信的，因为没有从主机转发到tun虚拟IP的转发路由。在HostU 上，我们需要将所有进入专用网络（192.168.60.0/24）的数据包定向到 tun 接口，从该接口可以通过 VPN 隧道转发数据包。如果没有此设置，我们将无法访问专用网络。我们可以使用 route 命令添加路由条目。使用指令route add -net 192.168.60.0/24 tun0后的路由表：

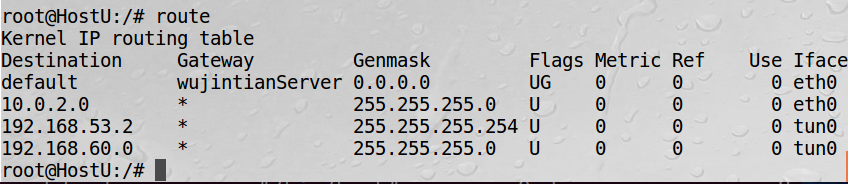


图4.7 HostU的路由

这样便完成了隧道的建立同时数据可以在隧道上传输，但是外网主机却仍旧无法和内网主机HostV通信，因为HostV主机的回复报文没有转发路由，转发到服务器端进行加密，而后再经过隧道传输。

于是添加目标地址为外网虚拟IP网段的数据转发到docker2的路由，添加指令和结果如下：

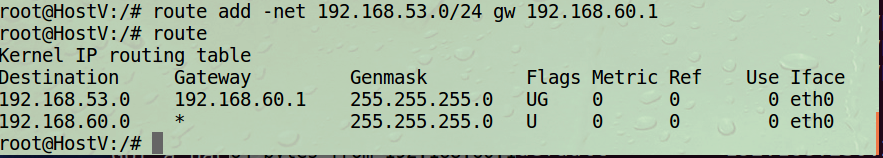


图4.8 HostV的路由

**3. 测试VPN隧道**

完成好配置后可以使用ping和telnet指令测试是否有数据在隧道上传输，可以通过wireshark查看隧道上的数据。隧道的两端分别为客户端的网关和服务器的网关，10.0.2.7和10.0.2.8。

使用ping指令ping内网主机HostV测试，外网主机HostU收到了回复报文，说明内外网之间可以进行通信：

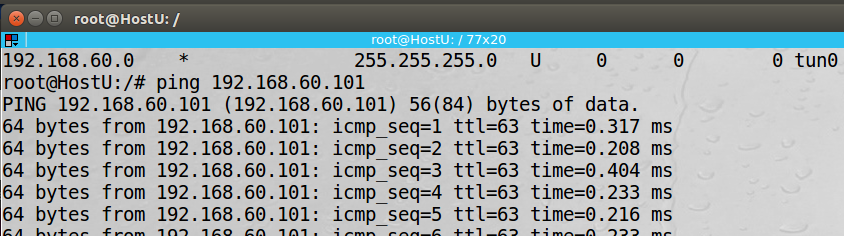


图4.9 HostU与HostV通信

而后在wireshark中的docker1端口，可以看到10.0.2.7和10.0.2.8之间的数据交互：（因为是已经完成加密的通信所以显示的是TCP报文而不是ICMP）

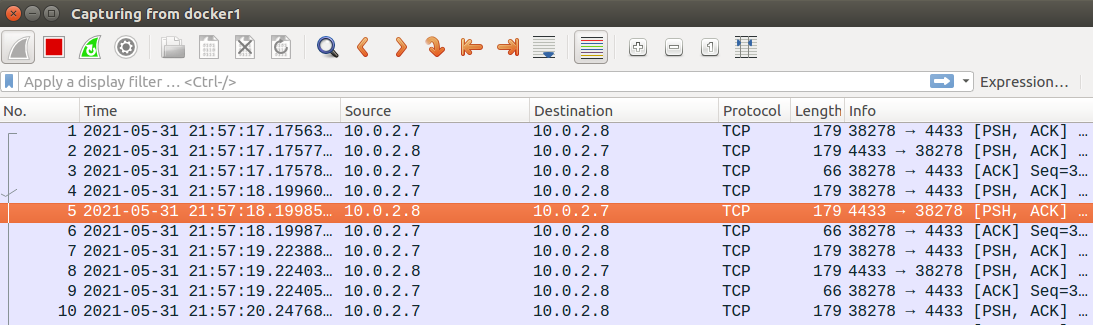


图4.10 隧道流量

使用telnet进行的测试放在思考题中进行展示和说明。

#### 4.2 隧道加密

为了保护这条隧道，我们需要实现两个目标，即机密性和完整性。使用加密来实现机密性，即通过隧道的内容将被加密。完整性目标确保没有人可以篡改隧道中的流量或发起重放攻击。使用消息验证代码（MAC）可以实现完整性。

为了实现这两个目标，在服务端和客户端的程序中使用传输层协议（TLS）实现这两个目标。实现加密的主要流程：

第一步是使用初始化OpenSSL库，使用相关的函数加载ssl上下文ctx，再制定证书验证方法，加载根证书，最后使用SSL\_new(ctx)生成一个SSL\*类型变量。

第二步是创建一个服务器端和客户端的TCP 套接字，生成一个sockfd。TCP套接字的创建流程如下图：



图4.11 tcp套接字创建

最后一步就是使用SSL\_set\_fd(ssl, sockfd);将ssl和相应的套接字连接，在客户端使用SSL\_connect()，在服务端使用SSL\_accept()进行TLS握手协议。相关流程如下图所示：

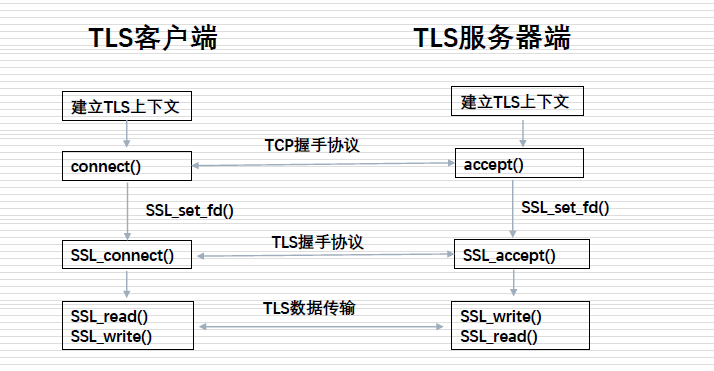


图4.12 TLS连接

完成这些后，就可以使用tunSelected()、socketSelected()函数进行监测，而后使用SSL\_write和SSL\_read函数进行服务端和客户端的通信。

加密隧道的检验，查看docker1端口的ping报文数据是否为加密数据：

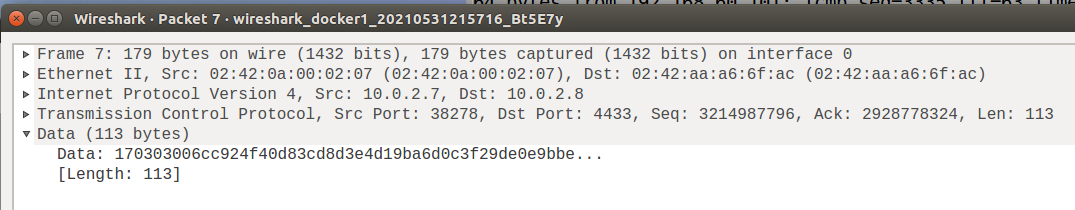


图4.13 加密ping报文数据

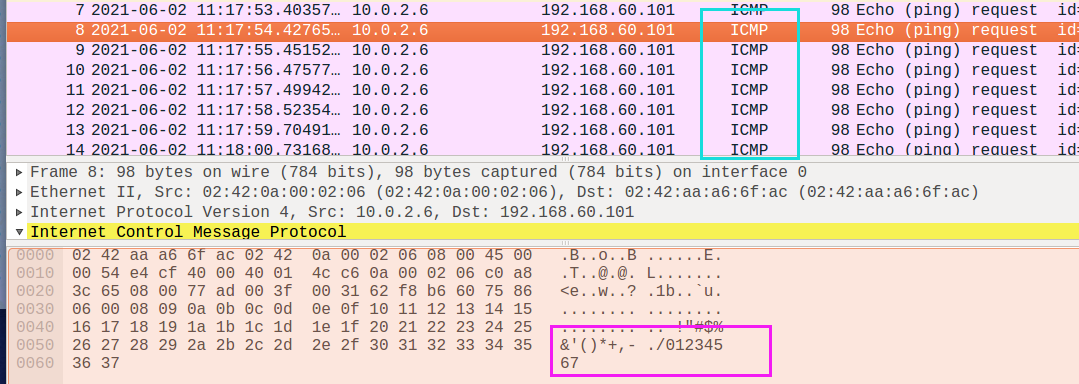


图4.14 未加密ping报文

未加密的ping报文数据是123456如图4.14所示，而此数据并不是，同时wireshark将该报文识别为TCP报文而非ICMP报文，所以加密隧道的加密成功。同时可以在运行程序的输出中查看到数据在tun和tunnel之间的流动：

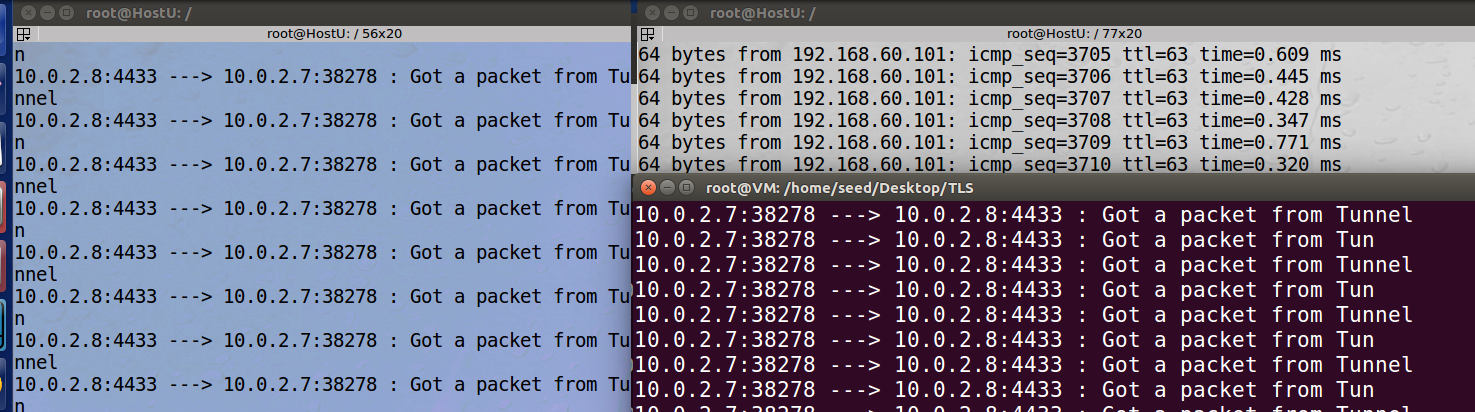


图4.15 程序打印信息

#### 4.3 认证VPN服务器

在建立VPN之前，VPN客户端必须对VPN服务器进行身份认证，确保服务器不是假冒的服务器。

在实验中对服务器的认证是使用公钥证书实现的。VPN 服务器需要首先从证书颁发机构（CA）获取公钥证书。当客户端连接到 VPN 服务器时，服务器将使用证书来证明它是客户端预期的服务器。

使用openssl相关的指令生成CA、服务器证书、客户端证书，将证书文件夹复制到HostU中。在生成证书的过程中需要注意的地方：

1. 要根据openssl.cnf文件设置相应的目录文件
2. 创建的index.txt文件为空文件
3. 创建的serial文件中写入1000

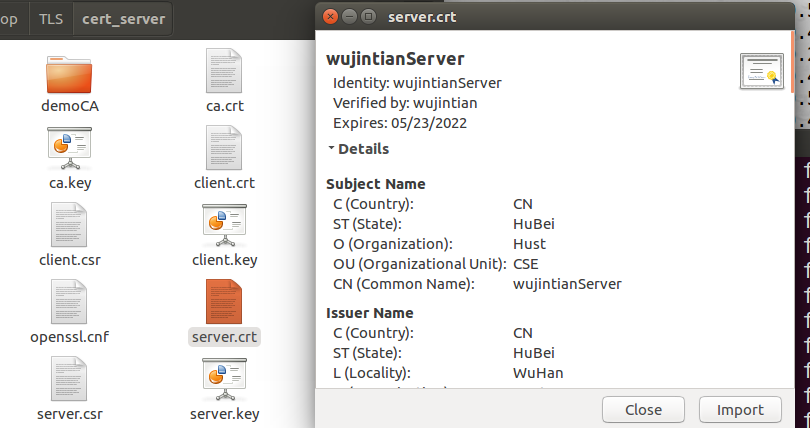


图4.16 证书文件目录

生成了服务器的相关证书后，还需要在程序中将证书的信息加载在一次通话的ssl中，让客户端使用根CA来验证服务端的加载在通话ssl中的证书是否合法。以下解释一下在程序中使用的一些SSL函数：

为了使证书正确的加载，在程序中定义了各证书文件的路径以及文件名：

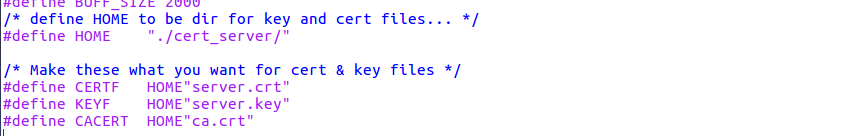


图4.17 文件名

在服务端程序中先使用SSL\_CTX\_set\_verify()函数，函数的第二个参数使用SSL\_VERIFY\_NONE参数表示不进行验证，而在客户端的程序中使用SSL\_VERIFY\_PEER参数表示对对等方的证书进行验证。在函数中使用SSL\_CTX\_load\_verify\_locations()函数将根CA加载在上下文中，客户端加载CA进行验证服务器端证书的合法性。

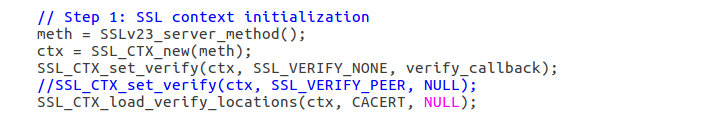


图4.18 server端证书加载

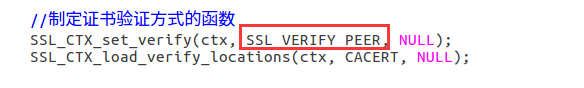


图4.19 client端验证方式

随后运行两端程序即可。验证是否真正的进行了验证，在客户端使用伪造的server.crt，而后运行程序，会有错误提示，如图4.19所示：

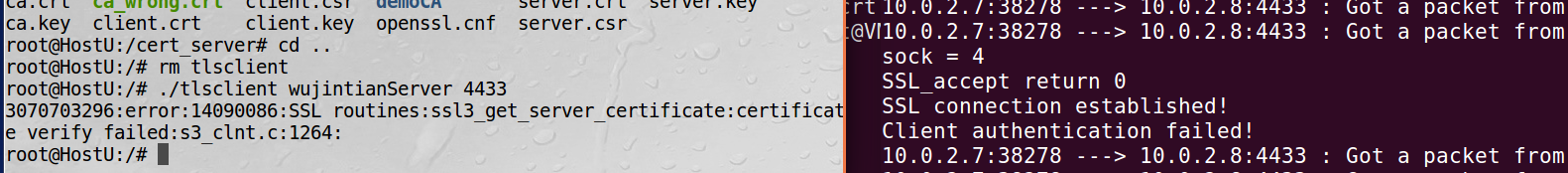


图4.20 服务器证书验证

在实验过程中除开证书生成的错误，还有几个错误值得记录和注意：

1. 在编写服务端程序时，需要使用私钥文件server.key，使用该文件需要输入密码。而当程序中添加daemon(1,1)时，会将执行的服务器程序放到后台运行，而后将控制权转交给bash。如果将daemon(1,1)函数放在SSL\_CTX\_use\_PrivateKey\_file()函数之前，那么当输入私钥文件密码的时候，终端将密码识别为linux的控制台指令，同时还有回显。这样之后，甚至整个终端陷入卡死状态。

改错方法：只需要将daemon(1,1)放在输入之后即可。

1. 运行客户端程序时输入服务器的IP不能正确执行，需要在程序中编写根据主机名获取IP地址的函数。在执行客户端程序时将服务器域名作为参数输入。在程序中主要使用getaddrinfo()函数获取IP地址。同时需要在/etc/hosts文件中添加服务器域名对应的IP。

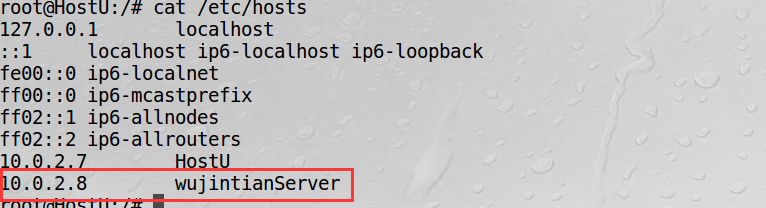


图4.21 /etc/hosts文件

#### 4.4 认证客户端

对于客户端的认证，使用的是存储在 shadow 文件中的帐户信息对用户进行身认证。该程序使用 getspnam()从 shadow 文件中获取给定用户的帐户信息，包括散列密码。然后，它使用 crypt()来散列给定的密码，并查看结果是否与从 shadow 文件中获取的值匹配。如果是，则用户名和密码匹配，并且验证成功。

在服务端实现这个函数是很简单的，函数代码如下所示：

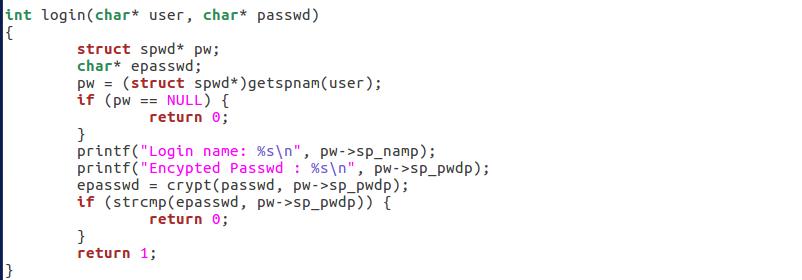
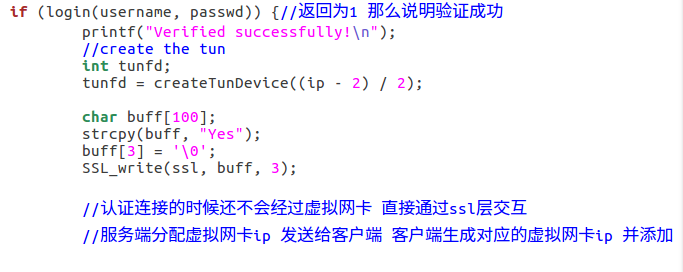
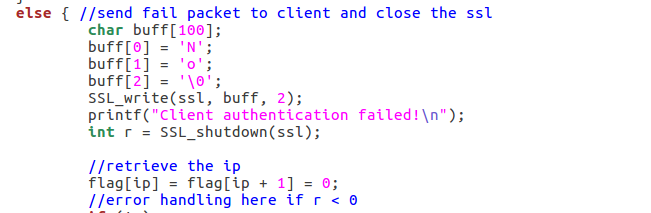


图4.22 login代码

如上图代码所示，当输入密码和用户名匹配时，返回数字1；错误或者不匹配时返回0。在服务端对返回值的判断进行相应的操作，当验证成功时服务器端使用SSL\_write()向客户端发送数据”Yes”，否则发送“No”并且断开该次连接。





在服务器端对客户端进行验证，客户端自然需要将用户名和密码传递给服务器，通过简单的scanf就能获取输入，而后使用SSL\_write将数据发送给服务器即可。

在实验中使用了termios结构体关闭回显，让输入客户密码时密码不被显示。具体代码如下：



图4.23 关闭密码的回显

在getpasswd()函数中，设置标准输入文件描述符不显示输入字符，但显示换行符，在完成输入后，恢复初始的终端设置。以下是客户端输入密码的截图：

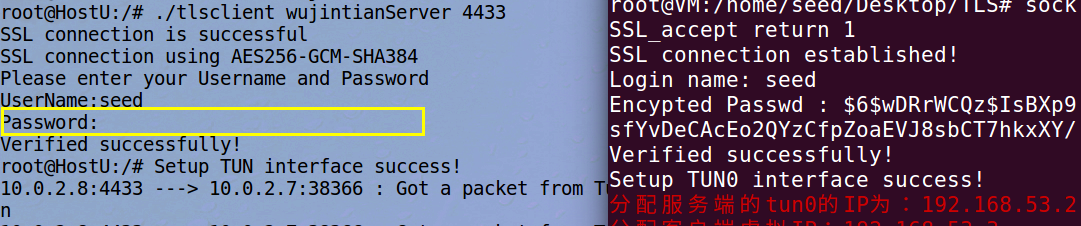


图4.24 关闭回显

验证当客户端输入错误密码时：

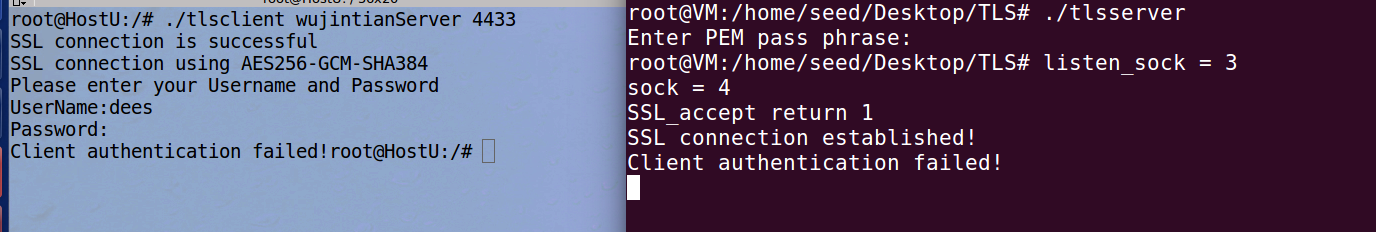


图4.25 输入错误密码

#### 4.5 多客户端实现

在现实情况中通常会有多个客户端同时连接服务器，同时进行数据传输，在实验中也实现了这个功能。

实现该功能的难点在于需要给客户端的tun分配虚拟IP，同时需要知道内网发送的数据要经过哪条隧道发送到目标主机。为了解决这两个问题，在服务端验证完客户端后，每一个连接都为它创建一个tunX(X顺序递加)，同时为对应的客户端分配虚拟IP。

以下展示实现的代码流程以及讲解：

1. 为建立连接的服务端和客户端获取可用的虚拟IP：

在程序中将一个连接的虚拟IP分为一组划分为网络前缀为31，例如：设置让服务端的ip为192.168.53.2 、4 、6 、8 等等，对应客户端的虚拟IP分别为3，5，7，9等。

在服务器程序的accept()函数之后，通过对flag[256]数组的扫描，判断哪个IP未被使用，则准备将这个IP分配给服务端的虚拟网卡tunX(X=(IP-2)/2)，将IP+1分配给对应的客户端。

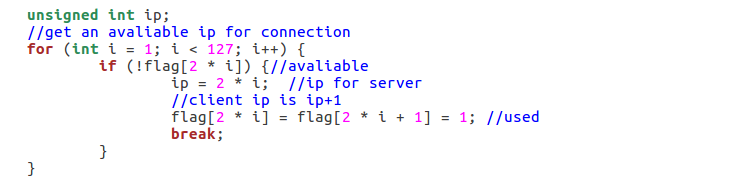


图4.26 get the ip

1. 将获取的可用虚拟IP进行使用：

在1中获取了一组可用的虚拟IP，如果通过了两端的认证，则服务器端根据获取的可用虚拟IP创建对应的tunX(X=(IP-2)/2)，同时在程序中构建字符串指令，利用system()函数激活tunX，以下为相关代码：

双方均验证成功后，将IP分配给对应的客户端，通过SSL\_write()函数直接告诉对端：

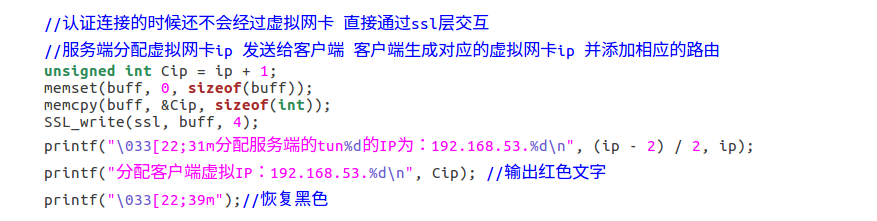


图4.27 分配虚拟IP

在createTunDevice()函数中构建激活tunX的指令，下图中的my\_itoa()函数是自己编写的itoa函数，即将整数转化为字符串：

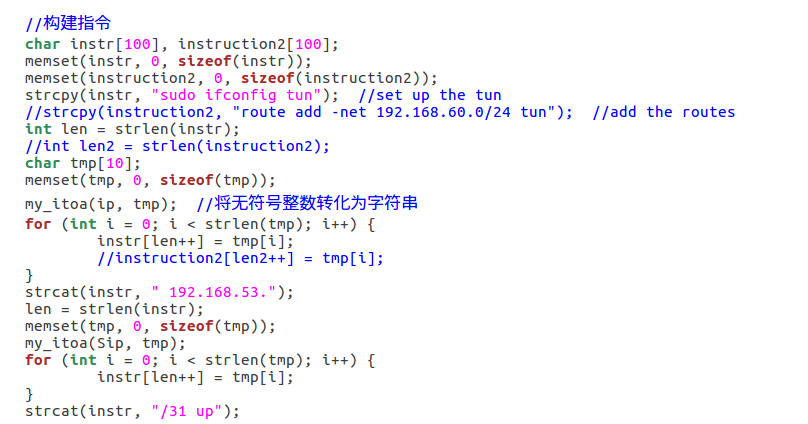


图4.28 激活服务端tunX

在客户端经过SSL\_read读出分配给自己的虚拟IP后，在createTunDevice()函数中创建该虚拟IP的tun0，激活后还需要添加一条转发路由：

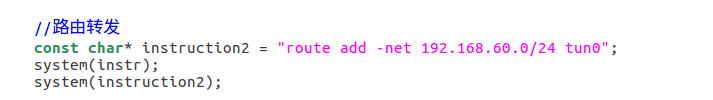


图4.29 添加路由

如果双方没有验证成功，则需要将获取的虚拟IP进行回收：



图4.30 ip回收

1. 完成虚拟IP分配后，客户端可以和服务器端通信：

使用服务器端一个虚拟网卡对应一个客户端的好处在于可以直接的进行通信，和单客户端的读写流程一致。使用管道进行通信需要建立对管道的监听，将管道的数据进行针对性的转发。

1. 虚拟IP的回收：

在进行虚拟IP的回收时利用了一个父子进程的常用函数exit()、以及waitpid()，当客户端结束通信时，通过SSL\_write()将虚拟IP告诉给服务器对应的子进程，子进程也通过exit(Cip)退出，同时将Cip传递给父进程。父进程通过waitpid(-1,&status,WNOHANG)，接收Cip，并将flag[Cip-1]和flag[Cip]置为0，表示为可用。

在accept()函数后调用signal()函数，表示有新连接建立后执行signal中的回调函数：



图4.31 signal函数对子进程处理

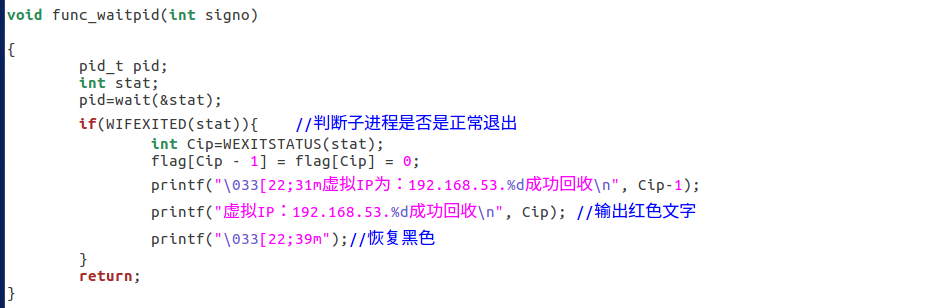


图4.32 信号回调函数

当子进程正常退出时，即客户端与服务器端的连接正常断开连接的时候，在该回调函数中将flag数组的对应位置为0并打印出回收该虚拟IP。以下为回收IP的演示：

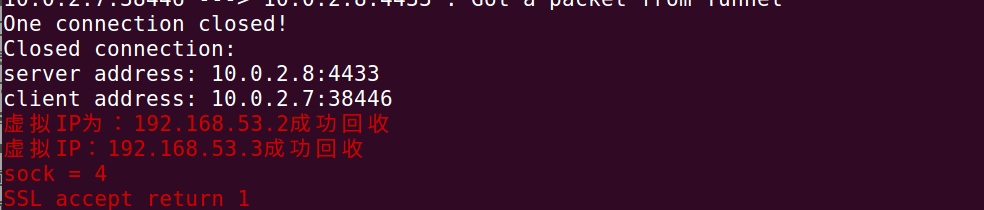


图4.33 虚拟IP回收

当这个IP回收后，再次建立新的连接时就可以再次将这些可用的IP分配给该组连接。

多客户端连接服务器，HostU和HostW两个客户端同时与服务器进行连接，同时ping内网主机192.168.60.101，可以看到两个不同的连接的流量。

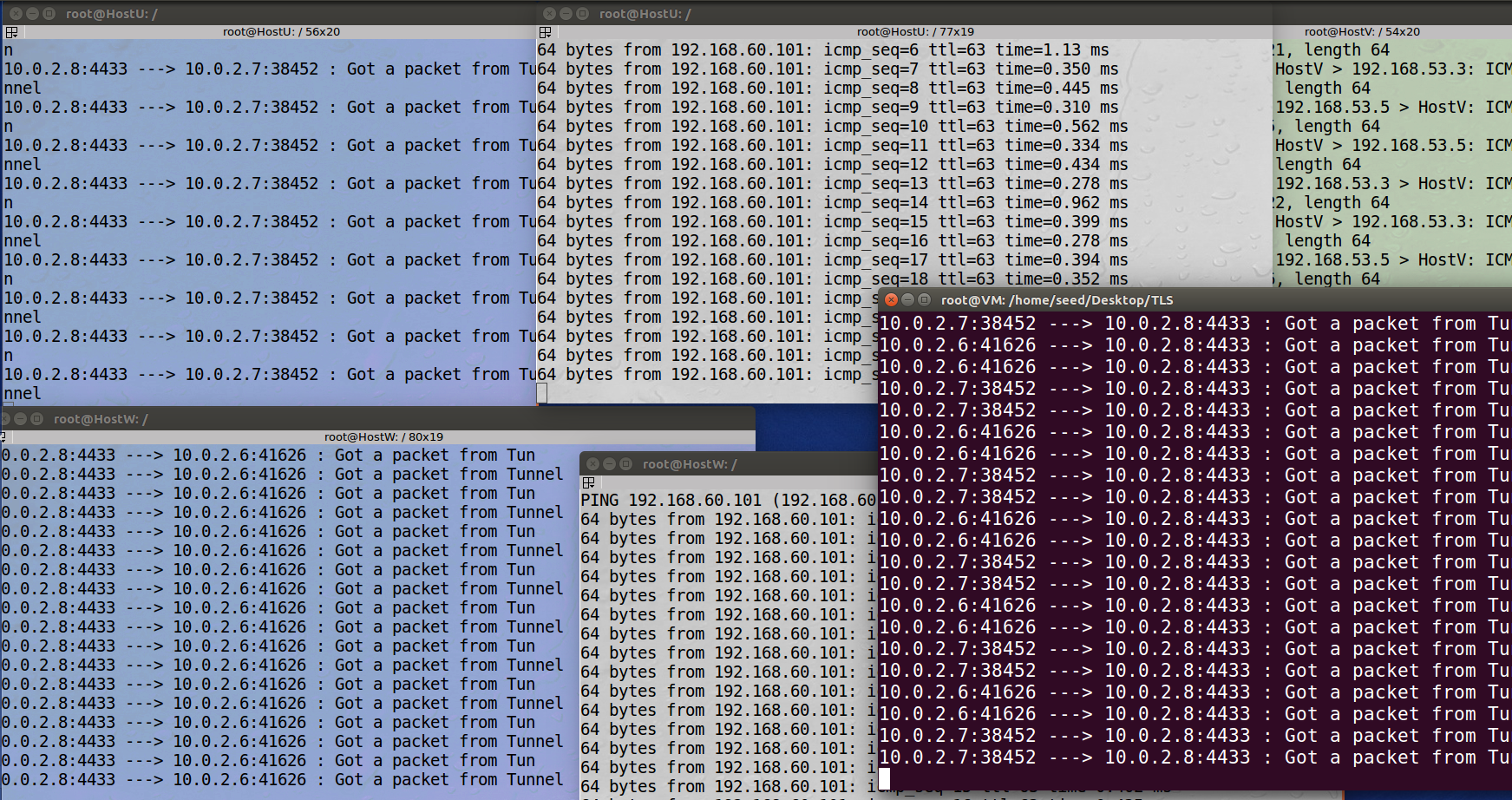


图4.34 多客户端ping

随后断开HostW的连接，查看到对应的虚拟IP组:192.168.53.4/31被回收，由于在程序中使用了waitpid()接受服务端子进程的退出状态故不会使得生成的子进程为僵尸进程。如下图4.33所示，结束了一个客户端进程，对应连接的服务端子进程也随之结束，此时服务端仅有一个主进程等待新的连接建立和一个已建立连接的子进程服务：

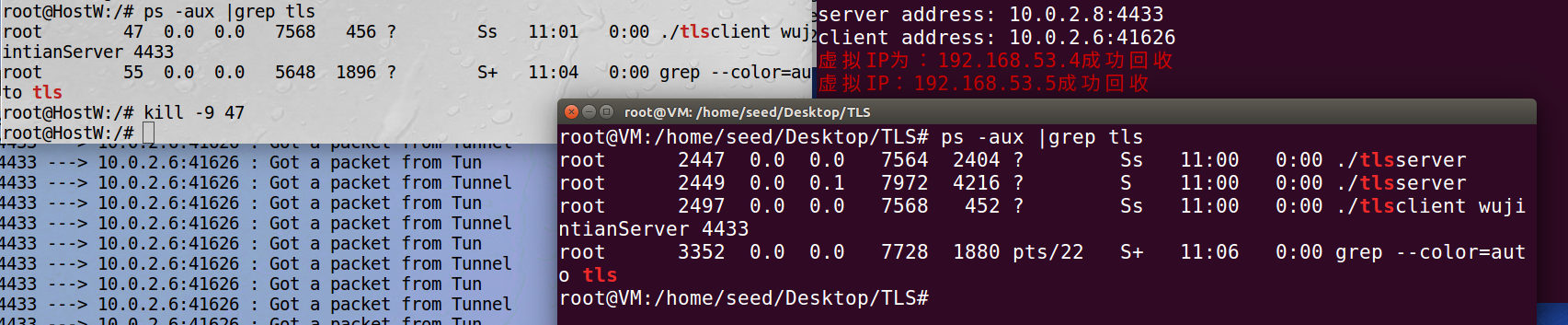


图4.35 退出客户端IP回收

当主机HostW再次建立连接的时候，重新分配的IP就是最小的可用IP：

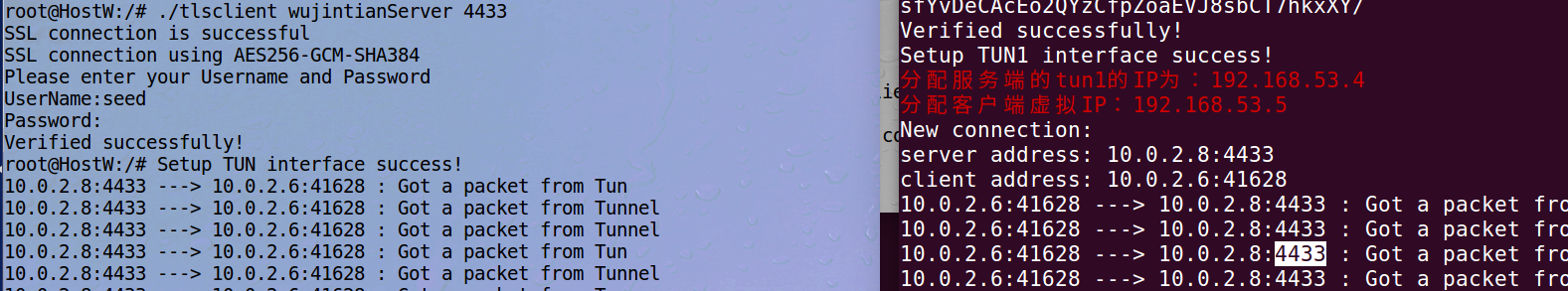


图4.36 重新建立连接分配的虚拟IP

由上图可知，当有用户退出，该用户使用的虚拟IP被回收了，另一个用户连接时可以获得该分配的IP。

#### 4.6 报文处理流程

以下讲解一次客户端与服务器端交互的报文处理，以下为网络拓扑图：

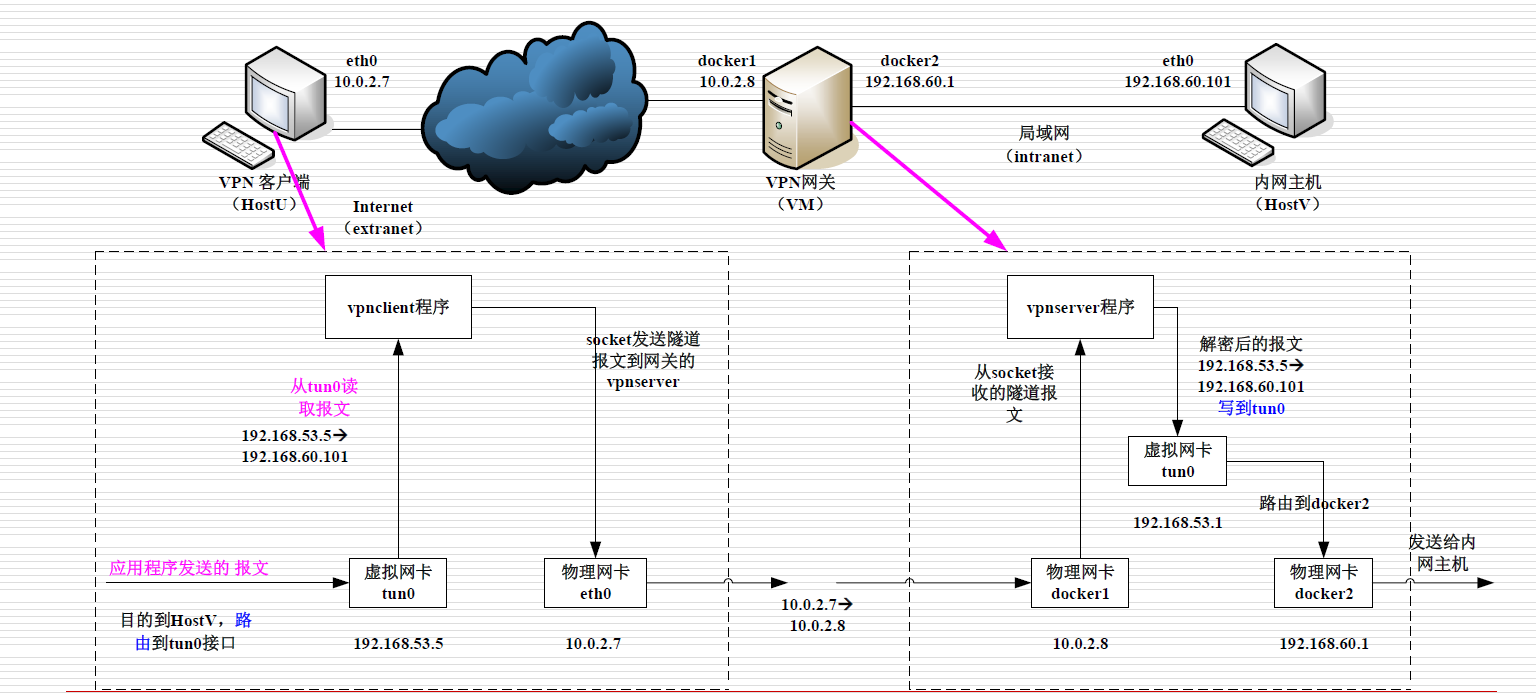


图4.37 网络拓扑

1. 外网主机发出数据
2. 数据需要经过虚拟网卡tun0发送数据，从tun0发送出去之前，内核使用虚拟网卡的虚拟IP作为源IP地址来构造程序数据的IP数据包。此后虚拟网卡tun0将源IP为:192.168.53.5，目标地址为192.168.60.101的IP数据包根据路由表中的路由将数据包转发给VPN客户端程序。
3. IP数据包通过虚拟接口tun0来到了我们的VPN客户端程序，之后数据包会被加密再发送给内核，这次将客户端的网关和服务端的网关作为通信隧道，即此时封装的IP数据包源IP:10.0.2.7，目的IP：10.0.2.8.
4. 此时加密数据就在Internet中传递转发，即可保障数据在传输过程的机密性。
5. 数据包到达服务器端的网关后完成了一步解封装，到达服务器程序进行解密，然后再通过服务器端的虚拟网卡将解密后的程序数据包传递给内核。这时的数据包源IP：192.168.53.5，目的IP：192.168.60.101.
6. 内核查看目标地址，并根据它进行路由转发到对应的内网主机。最终数据包达到192.168.60.101主机。

内网主机的回复过程也类似，只是IP不同而已，流程和封装过程是一样的。

在保持telnet 连接存活的同时，断开VPN 隧道。

## **5 实验思考**

**测试：**

当HostU主机使用telnet连接上HostV后，再断开VPN隧道，HostU主机的telnet终端界面卡死了，无法输入任何的指令。

重新连接VPN 隧道时，需要将vpnserver 和vpnclient 都退出后再重复操作，因为tun可以认为是配置在程序中的软件，断开隧道后tun虚拟网卡接口也断开了，需要重新配置激活后再进行连接。当正确重连后，可以继续通信了，之前卡住的HostU主机的终端界面可以输入指令并成功执行。

**思考与分析：**

在隧道未断开的时候，当客户端与服务器互相验证通过后，在客户机使用telnet指令和内网主机建立连接。HostU和内网主机已经建立了telnet连接，这时候断开VPN隧道，HostU和内网主机之间的连接仍然存在着，只是没有了传输数据的路径（就是隧道）。当VPN隧道重新连接的时候，HostU和内网主机之间的通信要求1.两者之间建立者连接 2.有着数据传输路径，这两条都满足了，自然可以继续通信了。

而且发现，重新建立连接后的docker2中的telnet会话的ACK是和断开前最后一个报文的Seq对应的，这也说明了其实隧道断开但是会话并没有断开。重新建立隧道后就可以继续通信。

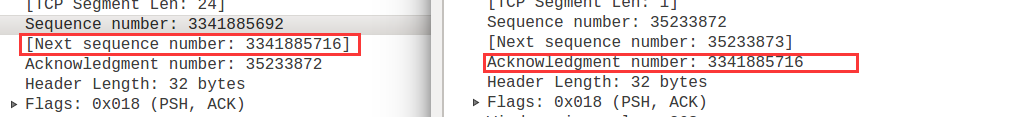


图5.1 会话ACK

# **心得体会与建议**

## **1 心得体会**

本次实验确实收获了挺多的，在刚开始的时候对照着老师所给的vpn程序和tls程序一个函数一个函数的对照着看函数的参数以及函数的功能。在学习和查找资料的过程中，又复习了一下上个学期使用过的socket套接字，同时对于TCP建立连接时实现也有了更多的理解。基于TCP的socket编程，在服务器端程序的流程：

1. 创建套接字socket
2. 再使用bind函数将套接字绑定到本地地址和端口
3. 使用listen函数监听客户端的连接请求，将请求加入到队列中
4. 使用accept函数从监听队列中取出一个请求进行连接并返回一个会话套接字
5. 使用该会话套接字进行通信使用send和recv函数，在程序中使用的是SSL的SSL\_write和SSL\_read函数
6. 关闭会话套接字

我觉得实验的难点在于对一些函数的底层实现和原理不太了解而产生的错误，以及多客户端的实现。

在实现双方身份认证的时候，将daemon函数放在了SSL\_CTX\_use\_PrivateKey\_file函数前，这将产生终端直接卡死的错误。使用了daemon函数将程序转到后台，而运行到SSL\_CTX\_use\_PrivateKey\_file函数是需要输入文件密码，但是此时控制权从程序转交回给bash，输入的密码被识别为指令，同时终端卡死。

在实现多客户端功能时，一开始使用的是管道进行父子进程之间的通信，主进程将分配的虚拟IP通过管道发送给对应的子进程，子进程在selectPipe函数中获取自己对应管道写来的信息。但是在实现的过程中出现了问题，对于多客户端进行ping指令时，丢包率不为0，怀疑是管道的管理出现问题，但是查看了很多遍的代码没有看出问题在哪，于是改用了服务器创建多虚拟网卡来对应客户机的方法。

在使用多虚拟网卡的过程中，一开始是将虚拟IP的最后八位的数值当作tun的编号，但是运行时会提示No such device的错误。这表明在程序中创建的tunX并没有成功，但是使用ifconfig -a查看到所有创建的tun都是顺序编号的，这个编号其实在使用tunfd = open("/dev/net/tun", O\_RDWR)的时候系统自动进行了分配。所以建立一个服务端虚拟IP到tun编号的线性映射，这样直接可以通过分配的虚拟IP来激活对应的虚拟网卡。

总的来说，这次的实验收获挺大的，不仅对SSL VPN的原理和实现有了更加深刻的了解，也重温了一下socket编程和多进程。

也非常感谢各位老师和助教的耐心解答！

## **2 建议**

感觉大部分的代码都已经给了出来，实际完整编写的功能就是多客户端了，感觉可以不需要那么多的提示代码。