

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
1 实验目的.....	1
2 实验环境.....	1
3 实验内容.....	2
4 实验步骤及结果分析.....	4
5 实验思考.....	19
心得体会与建议	20
1 心得体会.....	20
2 建议.....	20

实验三 VPN 实验

1 实验目的

虚拟专用网络（VPN）用于创建计算机通信的专用的通信域，或为专用网络到不安全的网络（如 Internet）的安全扩展。VPN 是一种被广泛使用的安全技术。在 IPSec 或 TLS/SSL（传输层安全性/安全套接字层）上构建 VPN 是两种根本不同的方法。本实验中，我们重点关注基于 TLS/SSL 的 VPN。这种类型的 VPN 通常被称为 TLS/SSL VPN。

本实验的学习目标是让学生掌握 VPN 的网络和安全技术。为实现这一目标，要求学生实现简单的 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

2.1 实验的网络拓扑

如图 2-1 为实验的网络拓扑，外网主机 HostU(HostU2...)上运行 VPN 客户端，其 IP 地址为 10.0.2.7。VPN 网关上运行 VPN 服务端，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 192.168.60.101。

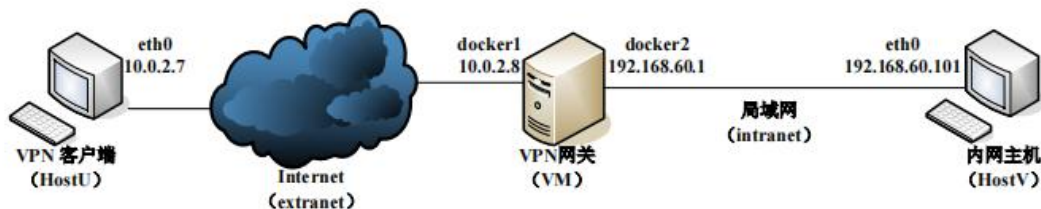


图 2-1 实验的网络拓扑

2.2 实验的操作系统

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。

2.3 网络配置

(1) 在 VM 上创建网络

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
```

(2) 创建容器 HostU 和 HostV 并删除路由

在 VM 上新开一个终端，创建并运行容器 HostU

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

在 VM 上新开一个终端，创建并运行容器 HostV

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

在容器 HostU 和 HostV 内分别删除掉默认路由

```
# route del default
```

2.4 软件组件

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。

3 实验内容

本次实验是为 Linux 操作系统实现一个简单的 VPN。我们将其称为 miniVPN。

3.1 任务 1：配置环境

我们将在计算机（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）。

3.2 任务 2：使用 TUN/TAP 创建一个主机到主机的隧道

TLS/SSL VPN 中使用了 TUN/TAP 技术。TUN 和 TAP 是虚拟网络内核驱动程序；它们可以实现完全由软件支持的网络设备。TAP 模拟以太网设备，处理的是以太网帧等二层数据包；TUN 模拟网络层设备，处理的是 IP 等三层数据包。我们可以用 TAP/TUN 创建虚拟网络接口。

用户空间程序通常访问 TUN/TAP 虚拟网络接口。操作系统通过 TUN/TAP 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/TAP 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。

vpnclient 和 vpnserver 程序是 VPN 隧道的两端，它们使用 TCP 或 UDP 协议通过图 2-2 中描述的套接字相互通信。为简单起见，在我们的示例代码中，我们选择使用 UDP。客户端和服务端之间的虚线描述了 VPN 隧道的

路径。VPN 客户端和服务端程序通过 TUN 接口连接到主机系统，做以下两件事：

- (1) 从主机系统获取 IP 数据包，因此数据包可以通过隧道发送；
- (2) 从隧道获取 IP 数据包，然后将其转发到主机系统，主机系统将数据包转发到其最终目的地。以下过程介绍了如何使用 `vpncclient` 和 `vpnservice` 程序创建 VPN 隧道。

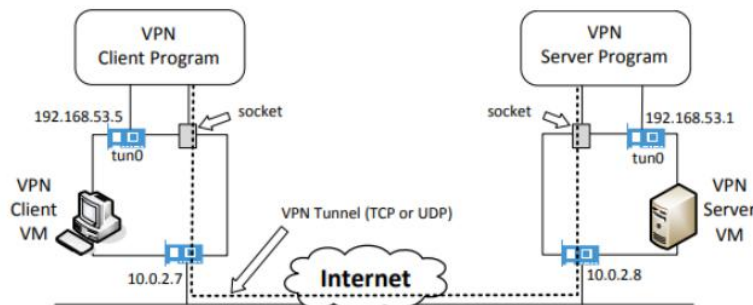


图 2-2 VPN 的客户端和服务端

3.3 任务 3：加密隧道

此时，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。

TLS 通常建立在 TCP 之上。任务 2 中的示例 VPN 客户端和服务端程序使用 UDP，因此我们首先需要使用 TCP 通道替换示例代码中的 UDP 通道，然后在隧道的两端之间建立 TLS 会话。TLS 客户端和服务端程序示例 (`tlsclient` 和 `tlsserver`)。tgz 压缩文件中包含的 README 文件中提供了有关如何编译和运行代码的说明。在演示中，你需要使用 Wireshark 捕获 VPN 隧道内的流量，并显示流量确实已加密。

3.4 任务 4：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。另一方面，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。在这个任务中，我们实现服务器认证。客户端身份认证在下一个任务中。

认证服务器的典型方法是使用公钥证书。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。

3.5 任务 5：认证 VPN 客户端

访问专用网络内的计算机是一项权限，仅授予授权用户，而不是授予所有人。因此，只允许授权用户与 VPN 服务器建立 VPN 隧道。在此任务中，授

权用户是在 VPN 服务器上拥有有效帐户的用户。因此，我们将使用标准密码身份验证来验证用户身份。基本上，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。

3.6 任务 6：支持多个客户端

在真实应用中，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。

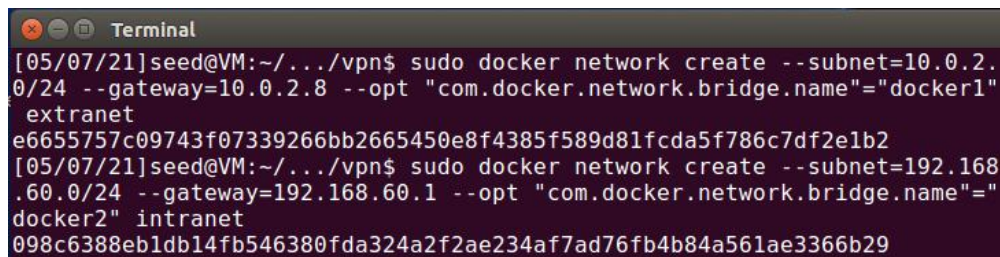
在典型的实现中，VPN 服务器进程（父进程）将为每个隧道创建一个子进程（参见图 4）。当数据包来自隧道时，其相应的子进程将获取数据包，并将其转发到 TUN 接口。无论是否支持多个客户端，此方向都相同，另一个方向变得具有挑战性。当数据包到达 TUN 接口（来自专用网络）时，父进程将获取数据包，现在需要确定该数据包应该到达哪个隧道。你需要考虑如何实施这种决策逻辑。

一旦做出决定并选择了隧道，父进程就需要将数据包发送到附加了所选隧道的子进程。这需要 IPC（进程间通信）。典型的方法是使用管道。

4 实验步骤及结果分析

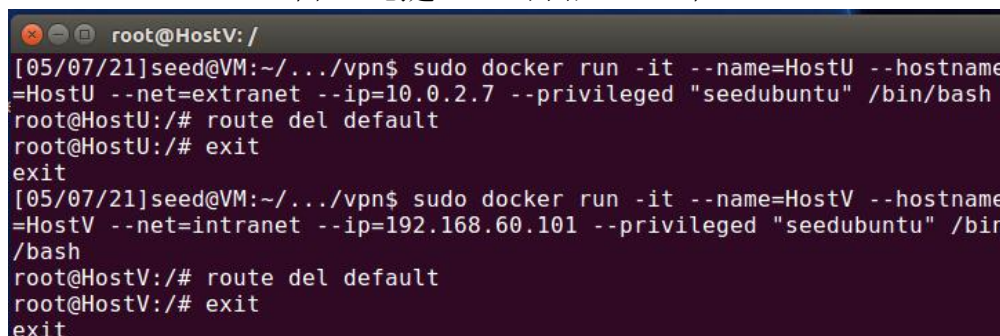
4.1 任务 1：配置环境

按照上述 2.3 中描述的指令先创建 docker 网络 extranet 和 intranet，分别表示外网和内网。其中，10.0.2.0/24 网段是外网，192.168.60.0/24 网段是内网。然后在 VM 创建容器 HostU 和 HostV 分别代表外网的主机和内网中带访问的主机。同时为了后面 VPN 的设置，将两个容器的默认路由进行删除。如图 4-1 和图 4-2 所示。当前的 VM 作为 VPN 网关。



```
Terminal
[05/07/21]seed@VM:~/.../vpn$ sudo docker network create --subnet=10.0.2.0/24 --gateway=10.0.2.8 --opt "com.docker.network.bridge.name"="docker1" extranet
e6655757c09743f07339266bb2665450e8f4385f589d81fcda5f786c7df2e1b2
[05/07/21]seed@VM:~/.../vpn$ sudo docker network create --subnet=192.168.60.0/24 --gateway=192.168.60.1 --opt "com.docker.network.bridge.name"="docker2" intranet
098c6388eb1db14fb546380fda324a2f2ae234af7ad76fb4b84a561ae3366b29
```

图 4-1 创建 docker 网络 extranet 和 intranet



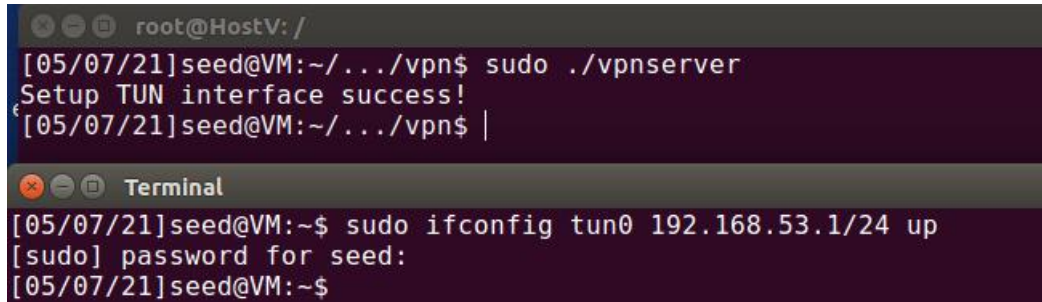
```
root@HostV: /
[05/07/21]seed@VM:~/.../vpn$ sudo docker run -it --name=HostU --hostname=HostU --net=extranet --ip=10.0.2.7 --privileged "seedubuntu" /bin/bash
root@HostU:/# route del default
root@HostU:/# exit
exit
[05/07/21]seed@VM:~/.../vpn$ sudo docker run -it --name=HostV --hostname=HostV --net=intranet --ip=192.168.60.101 --privileged "seedubuntu" /bin/bash
root@HostV:/# route del default
root@HostV:/# exit
exit
```

图 4-2 创建容器 HostU 和 HostV

4.2 任务 2：使用 TUN/TAP 创建一个主机到主机的隧道

运行示例给出的 vpn.tgz 压缩包中的 VPN 服务端和客户端程序。

首先在 VM 上启动 VPN 服务端程序 `vpnservice`，同时需要在另一个终端配置 `tun0` 设备的虚拟 IP 为 `192.168.53.1/24` 并激活该接口。如图 4-3 所示。

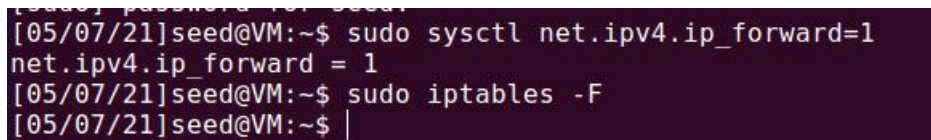


```
root@HostV: /
[05/07/21]seed@VM:~/.../vpn$ sudo ./vpnservice
Setup TUN interface success!
[05/07/21]seed@VM:~/.../vpn$ |

Terminal
[05/07/21]seed@VM:~$ sudo ifconfig tun0 192.168.53.1/24 up
[sudo] password for seed:
[05/07/21]seed@VM:~$
```

图 4-3 运行 VPN 服务端并激活 `tun0` 端口

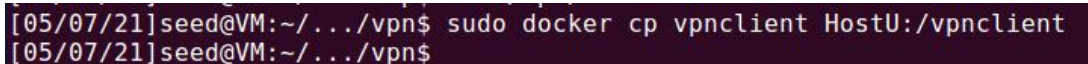
VPN 服务端需要在内网和隧道之间转发报文，因此需要作为网关。需要使用 `sysctl` 命令启用 IP 转发，使其行为类似于网关。同时由于 VM 上的 `iptables` 规则可能阻断转发报文，还需要清除 `iptables` 规则。如图 4-4 所示。



```
[05/07/21]seed@VM:~$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
[05/07/21]seed@VM:~$ sudo iptables -F
[05/07/21]seed@VM:~$ |
```

图 4-4 设置 VPN 服务端启用 IP 转发及清除 `iptables` 规则

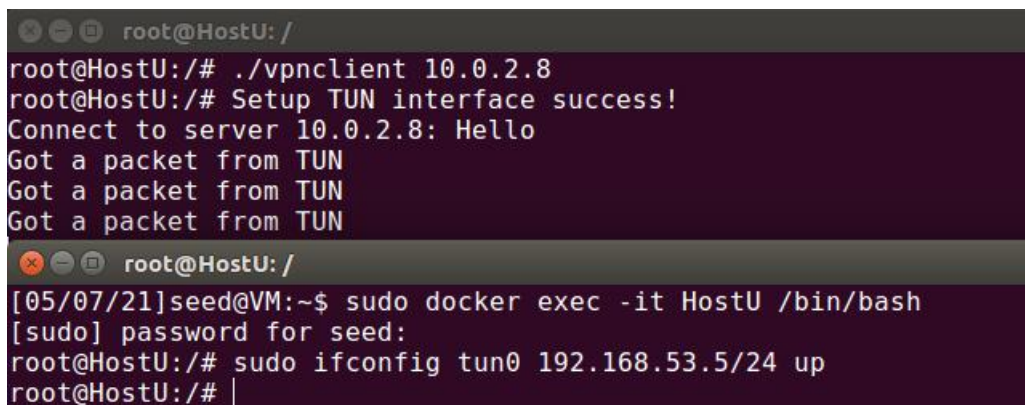
将编译好的 VPN 客户端程序拷贝到外网主机 HostU 中，如图 4-5 所示。



```
[05/07/21]seed@VM:~/.../vpn$ sudo docker cp vpnclient HostU:/vpnclient
[05/07/21]seed@VM:~/.../vpn$
```

图 4-5 拷贝 VPN 客户端到 HostU

在 HostU 中运行 VPN 客户端，会显示“Setup TUN interface success!”的字样表示成功建立了 TUN 设备，此时需要重新开启一个终端，将 IP 地址 `192.168.53.5/24` 分配给客户端的 `tun0` 接口。此时就会和 VM 上的 VPN 服务端成功建立起 VPN 隧道，也可以看到两者的控制台上都有输出，表示收到或发送数据包，如图 4-6 和 4-7 所示。



```
root@HostU: /
root@HostU:/# ./vpnclient 10.0.2.8
root@HostU:/# Setup TUN interface success!
Connect to server 10.0.2.8: Hello
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN

root@HostU: /
[05/07/21]seed@VM:~$ sudo docker exec -it HostU /bin/bash
[sudo] password for seed:
root@HostU:/# sudo ifconfig tun0 192.168.53.5/24 up
root@HostU:/# |
```

图 4-6 配置 VPN 客户端并运行

```
root@HostV: /
[05/07/21]seed@VM:~/.../vpn$ sudo ./vpnsver
Setup TUN interface success!
[05/07/21]seed@VM:~/.../vpn$ Got a packet from TUN
Got a packet from the tunnel
Got a packet from TUN
Got a packet from the tunnel
Got a packet from TUN
Got a packet from the tunnel
Got a packet from TUN
Got a packet from the tunnel
```

图 4-7 VPN 成功建立后 VPN 服务端

此时虽然建立了 VPN 隧道，但实际上 HostU 和 HostV 仍不能互通，需要为两者添加路由。对于 HostU，添加一个路由，将所有进入专用网络（192.168.60.0/24）的数据包定向到 tun0 接口，从该接口可以通过 VPN 隧道转发数据包，如图 4-8 所示。同时，还需要在 HostV 上添加路由，将所有发送给虚拟网卡设备所在网络（192.168.53.0/24）的数据包都转发给 VPN 网关 192.168.60.1/24，如图 4-9 所示。

```
root@HostU:/# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Ifac
e
default 10.0.2.8 0.0.0.0 UG 0 0 0 eth0
10.0.2.0 * 255.255.255.0 U 0 0 0 eth0
192.168.53.0 * 255.255.255.0 U 0 0 0 tun0
root@HostU:/# route del default
root@HostU:/# route add -net 192.168.60.0/24 tun0
root@HostU:/# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Ifac
e
10.0.2.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
192.168.53.0 0.0.0.0 255.255.255.0 U 0 0 0 tun0
192.168.60.0 0.0.0.0 255.255.255.0 U 0 0 0 tun0
```

图 4-8 在 HostU 上添加路由

```
root@HostV:/# route add -net 192.168.53.0/24 gw 192.168.60.1
root@HostV:/# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Ifac
e
0.0.0.0 192.168.60.1 0.0.0.0 UG 0 0 0 eth0
192.168.53.0 192.168.60.1 255.255.255.0 UG 0 0 0 eth0
192.168.60.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
```

图 4-9 在 HostV 上添加路由

添加完路由之后，这时候在 HostU 上 ping 内网主机 HostV，如图 4-10 所示，此时 HostU 和 HostV 是连通的，并且在 Wireshark 中截获到了相应的 ICMP 报文，如图 4-11 所示。

```
root@HostU:/# ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
64 bytes from 192.168.60.101: icmp_seq=1 ttl=63 time=0.130 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=0.145 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=0.171 ms
^C
--- 192.168.60.101 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2037ms
rtt min/avg/max/mdev = 0.130/0.148/0.171/0.022 ms
root@HostU:/#
```

图 4-10 HostU ping HostV

→	1	192.168.53.5	192.168.60.101	ICMP	84 Echo (ping) request	id=0x003a, seq=1/256,
←	2	192.168.60.101	192.168.53.5	ICMP	84 Echo (ping) reply	id=0x003a, seq=1/256,
	3	192.168.53.5	192.168.60.101	ICMP	84 Echo (ping) request	id=0x003a, seq=2/512,
	4	192.168.60.101	192.168.53.5	ICMP	84 Echo (ping) reply	id=0x003a, seq=2/512,
	5	192.168.53.5	192.168.60.101	ICMP	84 Echo (ping) request	id=0x003a, seq=3/768,
	6	192.168.60.101	192.168.53.5	ICMP	84 Echo (ping) reply	id=0x003a, seq=3/768,
	7	192.168.53.5	192.168.60.101	ICMP	84 Echo (ping) request	id=0x003a, seq=4/1024,
	8	192.168.60.101	192.168.53.5	ICMP	84 Echo (ping) reply	id=0x003a, seq=4/1024,

图 4-11 HostU ping HostV 的 Wireshark 截图

如图 4-12 所示，在 HostV 上开启 telnet 服务，此时在 HostU 上也可以成功使用 telnet 访问 HostV，如图 4-13 所示是 Wireshark 截获的 telnet 报文。

```

root@HostV: /
root@HostV:/# service openbsd-inetd start
* Starting internet superserver inetd
root@HostV:/# |

root@HostU: /
root@HostU:/# telnet 192.168.60.101
Trying 192.168.60.101...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
HostV login:

```

图 4-12 HostU 使用 telnet 访问 HostV

3	192.168.53.5	192.168.60.101	TCP	52 52746 → 23 [ACK]	Seq=3113710151
4	192.168.53.5	192.168.60.101	TELNET	76 Telnet Data ...	
5	192.168.60.101	192.168.53.5	TCP	52 23 → 52746 [ACK]	Seq=3669348182
6	192.168.60.101	192.168.53.5	TELNET	64 Telnet Data ...	
7	192.168.53.5	192.168.60.101	TCP	52 52746 → 23 [ACK]	Seq=3113710175
8	192.168.60.101	192.168.53.5	TELNET	67 Telnet Data ...	
9	192.168.53.5	192.168.60.101	TELNET	55 Telnet Data ...	
10	192.168.60.101	192.168.53.5	TCP	52 23 → 52746 [ACK]	Seq=3669348209
11	192.168.53.5	192.168.60.101	TELNET	61 Telnet Data ...	
12	192.168.60.101	192.168.53.5	TELNET	70 Telnet Data ...	
13	192.168.53.5	192.168.60.101	TELNET	86 Telnet Data ...	
14	192.168.60.101	192.168.53.5	TELNET	55 Telnet Data ...	
15	192.168.53.5	192.168.60.101	TELNET	55 Telnet Data ...	
16	192.168.60.101	192.168.53.5	TELNET	55 Telnet Data ...	
17	192.168.53.5	192.168.60.101	TELNET	55 Telnet Data ...	
18	192.168.60.101	192.168.53.5	TELNET	85 Telnet Data ...	
19	192.168.53.5	192.168.60.101	TCP	52 52746 → 23 [ACK]	Seq=3113710227

图 4-13 HostU 与 HostV 的 telnet 数据包

4.3 任务 3：加密隧道

```

Terminal
[05/11/21]seed@VM:~/Desktop$ cd tls
[05/11/21]seed@VM:~/./tls$ ./tlsserver
listen_sock = 3
sock = 4
SSL_accept return 1
SSL connection established!
Received: GET / HTTP/1.1
Host: 10.0.2.8

root@HostU: /tls
root@HostU: /tls# ./tlsclient 10.0.2.8 4433
Enter PEM pass phrase:
SSL connection is successful
SSL connection using AES256-GCM-SHA384
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html><html><head><title>Hello World</title></head><style>body
{background-color: black}h1 {font-size:3cm; text-align: center; color: w
hite;text-shadow: 0 0 3mm yellow}</style></head><body><h1>Hello, world!<
/h1></body></html>

root@HostU: /tls#

```

图 4-14 运行带加密的 VPN 服务端和客户端

运行示例给出的 tls.tgz 压缩包中的 VPN 服务端和客户端程序。在 VM 上运行 VPN 服务端程序，在 HostU 上运行 VPN 客户端程序，如图 4-14 所示，可以看到服务端接收到了客户端发送的“GET”数据，客户端也收到了服务端对“GET”的响应数据。这个过程中使用 Wireshark 进行数据截包（实现需要对 Wireshark 的 TLS 加密端口进行配置，如图 4-15 所示），可以看到在客户端和服务端之间的通信报文，且数据使用了 TLS 进行了加密，如图 4-16 和 4-17 所示。

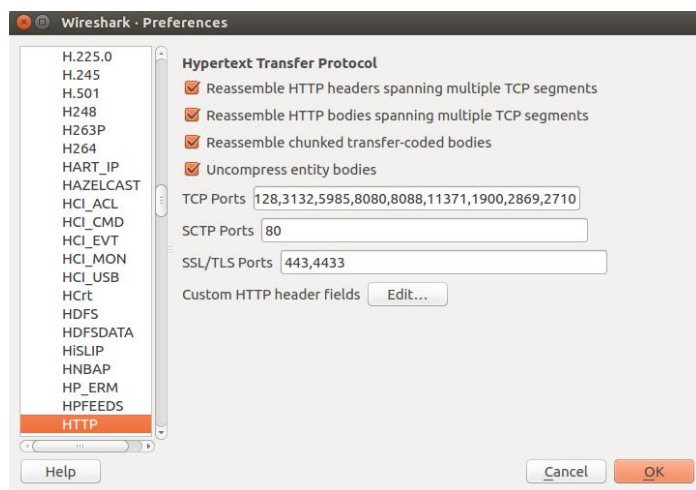


图 4-15 在 Wireshark 中添加 TLS 端口

No.	Source	Destination	Protocol	Length	Info
1	10.0.2.7	10.0.2.8	TCP	74	34292 → 4433 [SYN] Seq=2135176727 Win=29200 Len=0
2	10.0.2.8	10.0.2.7	TCP	74	4433 → 34292 [SYN, ACK] Seq=687608187 Ack=2135176727 Win=0 Len=0
3	10.0.2.7	10.0.2.8	TCP	66	34292 → 4433 [ACK] Seq=2135176728 Ack=687608188 Len=0
4	10.0.2.7	10.0.2.8	TLSv1...	371	Client Hello
5	10.0.2.8	10.0.2.7	TCP	66	4433 → 34292 [ACK] Seq=687608188 Ack=2135177033 Len=0
6	10.0.2.8	10.0.2.7	TLSv1...	1482	Server Hello, Certificate, Server Hello Done
7	10.0.2.7	10.0.2.8	TCP	66	34292 → 4433 [ACK] Seq=2135177033 Ack=687609604 Len=0
8	10.0.2.7	10.0.2.8	TLSv1...	256	Client Key Exchange, Change Cipher Spec, Encrypt...
9	10.0.2.8	10.0.2.7	TCP	66	4433 → 34292 [ACK] Seq=687609604 Ack=2135177223 Len=0
10	10.0.2.8	10.0.2.7	TLSv1...	292	New Session Ticket, Change Cipher Spec, Encrypt...
11	10.0.2.7	10.0.2.8	TLSv1...	126	Application Data
12	10.0.2.8	10.0.2.7	TLSv1...	373	Application Data
13	10.0.2.8	10.0.2.7	TLSv1...	97	Encrypted Alert
14	10.0.2.7	10.0.2.8	TCP	66	34292 → 4433 [ACK] Seq=2135177283 Ack=687610169 Len=0
15	10.0.2.7	10.0.2.8	TCP	66	34292 → 4433 [FIN, ACK] Seq=2135177283 Ack=687610169 Len=0
16	10.0.2.8	10.0.2.7	TCP	66	4433 → 34292 [ACK] Seq=687610169 Ack=2135177284 Len=0

图 4-16 Wireshark 截获到的 VPN 加密报文

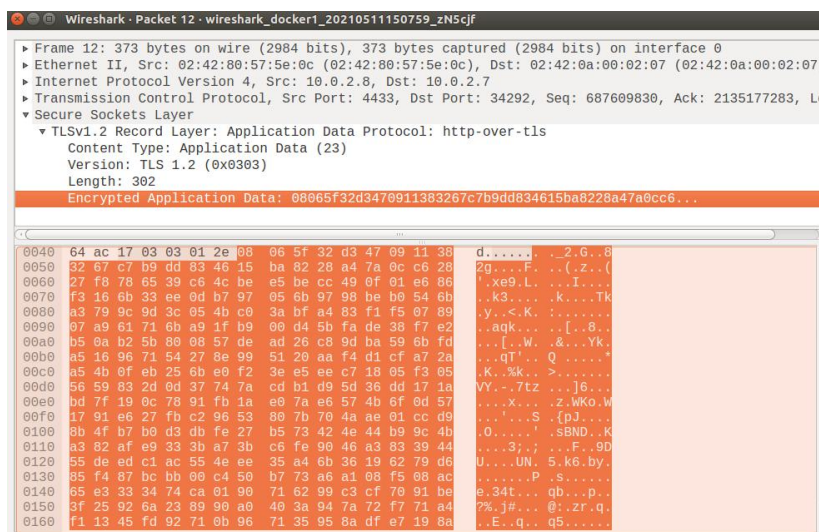


图 4-17 Wireshark 截获到的 VPN 加密报文

4.4 任务 4: 认证 VPN 服务器

在建立 VPN 之前, VPN 客户端对 VPN 服务器进行身份认证。此处通过 OpenSSL 进行证书的颁发和认证。

(1) 生成证书

- ① 首先创建一个目录 `cert_server` 用来存放服务器证书的相关文件。然后从 `/usr/lib/ssl/openssl.cnf` 获取配置文件的副本, 复制到 `cert_server` 目录中。接下来创建 `cert_server` 的子目录及相关文件。
 - 1) `cert_server/demoCA`: 用来存放 CA 相关的文件
 - 2) `cert_server/demoCA/newcerts`: 新证书的默认存放位置
 - 3) `cert_server/demoCA/index.txt`: 数据库索引文件
 - 4) `cert_server/demoCA/serial`: 当前序列号文件。输入 1000 表示起始的序列号。
- ② 创建证书颁发机构(CA)。使用如下命令为 CA 生成自签名证书, 作为根证书。其中, 证书的密码为 162169。运行命令后会在目录 `cert_server` 中生成证书文件 `ca.crt` 和根 CA 的私钥文件 `ca.key`。

```
$ openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
```

```
[05/23/21]seed@VM:~/../cert_server$ openssl req -new -x509 -keyout ca.k
ey -out ca.crt -config openssl.cnf
Generating a 2048 bit RSA private key
.....+++
writing new private key to 'ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a
DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:Hubei
Locality Name (eg, city) []:Wuhan
Organization Name (eg, company) [Internet Widgits Pty Ltd]:HUST
Organizational Unit Name (eg, section) []:CSE
Common Name (e.g. server FQDN or YOUR name) []:huanghaoyanvpn.com
Email Address []:ca@hhy.com
```

图 4-18 生成 CA 证书

Name	Size	Type	Modified
demoCA	3 items	Folder	09:52
ca.crt	1.4 kB	Unknown	09:53
ca.key	1.8 kB	Presentation	09:53
openssl.cnf	10.8 kB	Link to Text	Jan 31 2017

图 4-18 生成的 CA 证书及其私钥的目录

- ③ 生成服务端的私钥文件。在服务端执行以下命令获得 RSA 密钥对。其中, 密码为 162169。运行命令后在当前目录下会生成服务端的私钥文件 `server.key`。

```
$ openssl genrsa -des3 -out server.key 1024
```

```
Terminal
[05/22/21]seed@VM:~/../cert_server$ openssl genrsa -des3 -out server.ke
y 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
.....+++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```


图 4-19 生成 RSA 密钥对

- ④ 生成服务端的证书文件. 有服务端的私钥文件就可以生成证书签名请求(CSR), 发送给 CA 后, CA 将为该私钥生成证书. 使用如下命令. 运行上述命令后, 则会在当前路径下生成服务端 CSR 文件 server.csr.

```
$ openssl req -new -key server.key -out server.csr -config openssl.cnf
```

```
[05/23/21]seed@VM:~/.../cert_server$ openssl req -new -key server.key -o
ut server.csr -config openssl.cnf
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a
DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:Hubei
Locality Name (eg, city) []:Wuhan
Organization Name (eg, company) [Internet Widgits Pty Ltd]:HUST
Organizational Unit Name (eg, section) []:CSE
Common Name (e.g. server FQDN or YOUR name) []:hhyServer.com
Email Address []:server@hhy.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:server
An optional company name []:
```

图 4-20 生成服务端 CSR 文件

- ⑤ 生成客户端证书. 由 CSR 文件发送给受信任的 CA 进行签名生成证书. 使用如下命令. 运行上述命令后, 会在当前目录下生成服务端的证书 server.crt.

```
$ openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf
```

```
[05/23/21]seed@VM:~/.../cert_server$ openssl ca -in server.csr -out serv
er.crt -cert ca.crt -keyfile ca.key -config openssl.cnf
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
    Serial Number: 4096 (0x1000)
    Validity
        Not Before: May 23 02:03:37 2021 GMT
        Not After : May 23 02:03:37 2022 GMT
    Subject:
        countryName           = CN
        stateOrProvinceName   = Hubei
        organizationName      = HUST
        organizationalUnitName = CSE
        commonName            = hhyServer.com
        emailAddress          = server@hhy.com
    X509v3 extensions:
        X509v3 Basic Constraints:
            CA:FALSE
        Netscape Comment:
            OpenSSL Generated Certificate
        X509v3 Subject Key Identifier:
            0D:7C:AB:29:4E:F6:13:DC:39:C5:F3:CE:8C:67:F6:54:33:25:A6
:AF
        X509v3 Authority Key Identifier:
            keyid:0B:5E:67:23:6A:C4:76:FF:30:A3:20:15:E9:A2:50:1C:14
:6E:BE:C1

Certificate is to be certified until May 23 02:03:37 2022 GMT (365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]:y
Write out database with 1 new entries
Data Base Updated
```

图 4-21 生成服务端证书

- ⑥ 生成证书链接。将 CA 证书 ca.crt 复制到 ca_client 目录下。使用下列命令生成链接。

```
$ openssl x509 -in ca.crt -noout -subject_hash  
$ ln -sf ca.crt <hash>
```

执行结果图 4-22 所示。生成哈希值的符号链接之后客户端才能正常验证服务端的证书从而建立 TLS 连接，如果不建立链接则会在证书验证时出现“self signed certificate in certificate chain.”的自签名证书错误。

```
[05/26/21]seed@VM:~/.../ca_client$ openssl x509 -in ca.crt -noout -subject_hash  
ea9db75e  
[05/26/21]seed@VM:~/.../ca_client$ ln -sf ca.crt ea9db75e.0  
[05/26/21]seed@VM:~/.../ca_client$ ls -l  
total 4  
-rw-rw-r-- 1 seed seed 1395 May 23 10:09 ca.crt  
lrwxrwxrwx 1 seed seed 6 May 26 15:16 ea9db75e.0 -> ca.crt  
[05/26/21]seed@VM:~/.../ca_client$
```

图 4-22 生成证书哈希值符号链接

(2) 证书验证

证书验证通过 OpenSSL 提供的相关函数在 VPN 客户端代码中实现证书验证。具体而言，在初始化 TLS 客户端函数 TLSClient 中，使用 SSL_set_verify 函数设置证书的验证方式。其中第二个参数设置了证书验证方式，此处为 SSL_VERIFY_PEER，表示进行证书的校验；第三函数设置的是整数验证后的回调函数 verifyCallback，在该函数中使用 X509_get_subject_name 函数获取了当前整数的 SubjectName 并进行输出，同时对整数验证结果 preverify_ok 进行判断，若为 1 表示验证成功，为 0 表示验证失败，此外通过设置该回调函数的返回值来决定 TLS 连接是否继续，返回 1 表示继续 TLS 连接，返回 0 表示结束 TLS 连接。整数部分验证代码具体如下：

```
//证书验证  
int verifyCallback(int preverify_ok, X509_STORE_CTX *x509_ctx) {  
    char buf[300];  
    X509 *cert = X509_STORE_CTX_get_current_cert(x509_ctx);  
    X509_NAME_oneline(X509_get_subject_name(cert), buf, 300);  
    printf("certificate subject= %s\n", buf);  
    if (preverify_ok == 0) {  
        int err = X509_STORE_CTX_get_error(x509_ctx);  
        printf("Verification failed: %s.\n",  
              X509_verify_cert_error_string(err));  
        return 0;    //返回 0 结束 TLS 握手连接  
    }  
    printf("Verification passed.\n");  
    return 1;    //返回 1 继续 TLS 连接  
}  
//初始化 TLS 客户端  
SSL *setupTLSClient(const char *hostname) {  
    // ...  
    //设置证书验证方式  
    SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, verifyCallback);
```



```
// ...
}
```

```
VPN Client Closed: sockfd = 4
sockfd = 5
Connection from IP:10.0.2.7 port:41666
SSL_accept return 1
SSL connection established!
SSL connection using AES256-GCM-SHA384

root@HostU:/miniVPN# ./vpnclient hhyServer.com
TCP connect succeed! hostname IP:10.0.2.8 port:4433
certificate subject= /C=CN/ST=Hubei/L=Wuhan/O=HUST/OU=CSE/CN=huanghaoyan
vpn.com/emailAddress=ca@hhy.com
Verification passed.
certificate subject= /C=CN/ST=Hubei/O=HUST/OU=CSE/CN=hhyServer.com/email
Address=server@hhy.com
Verification passed.
SSL connection is successful
SSL connection using AES256-GCM-SHA384
Please input username:
```

图 4-23 服务端证书验证成功的情况

SSL_CTX_set_verify 函数只是设置了证书验证方式，实际的证书验证是在 TLS 建立握手连接时服务端会将自己的证书发给客户端，由客户端进行验证，即在客户端 SSL_connect 函数的调用过程中发生证书验证。如图 4-23 所示，为服务端使用正确证书时的验证情况；如图 4-24 则为服务端使用了假证书，客户端证书验证不通过的情况。

```
[05/26/21]seed@VM:~/.../miniVPN$ sudo ./vpnservice
Enter PEM pass phrase:
Private key match the certificate public key
listenSock = 3
sockfd = 4
Connection from IP:10.0.2.7 port:39106
SSL_accept failed!

root@HostU:/miniVPN# ./vpnclient hhyServer.com
TCP connect succeed! hostname IP:10.0.2.8 port:4433
certificate subject= /C=CN/ST=Hubei/L=Wuhan/O=Hust/OU=cse/CN=wmz
Verification failed: certificate has expired.
SSL_connect failed!
```

图 4-24 服务端证书验证失败的情况

4.5 任务 5：认证 VPN 客户端

客户端认证是服务端要求客户端提供用户名和密码，服务器将检查其 shadow 文件，若找到匹配的记录，则认证成功并建立起 VPN 隧道，反之则认证失败将断开连接。

这一部分的实习分客户端和服务端。在客户端中，只需要发送用户名和密码即可。在服务端，向客户端发送输入用户名密码的提示语后，读取客户端发送过来的用户名和密码，并使用 login 函数进行密码验证。在 login 函数中，会使用 getsnam 函数从 shadow 文件中获取给定用户的账号信息，若没有则表示服务端没有该账号返回失败；然后使用 crypt 函数对明文进行加密并与密文进行比较，若不同则验证失败，相同则验证成功。如下是客户端和服务端对客户

//客户端的 VPN 客户端验证

```
int verifyClient(SSL *ssl) {
    char username[20];
    char passwd[20];
```

```

char recvBuf[BUFF_SIZE];
int len = SSL_read(ssl,recvBuf,BUFF_SIZE);

//输入用户名
printf("%s\n",recvBuf);
scanf("%s", username);
getchar();
SSL_write(ssl,username,strlen(username)+1);
//输入密码
SSL_read(ssl,recvBuf,BUFF_SIZE);
printf("%s\n",recvBuf);
getPasswd(passwd, 20);
SSL_write(ssl,passwd,strlen(passwd)+1);
//获取验证结果
SSL_read(ssl,recvBuf,BUFF_SIZE);
if(strcmp(recvBuf, "Client verify succeed") != 0) {
    printf("Client verify failed!\n")
    return -1;
}
printf("Client verify succeed\n")
return 1;
}
}

//登录函数
int login(char *user, char *passwd) {
    //shadow 文件的结构体
    struct spwd *pw = getspnam(user);    //从 shadow 文件中获取给定用户的帐户信息
    if (pw == NULL) return -1;

    printf("Login name: %s\n", user);    //用户登录名
    // printf("Passwd: %s\n", passwd);    // 加密口令

    char *epasswd = crypt(passwd, pw->sp_pwdp);    //对 passwd 进行加密
    if (strcmp(epasswd, pw->sp_pwdp)) {
        return -1;
    }
    return 1;
}

//服务端的 VPN 客户端验证
int verifyClient(SSL *ssl) {
    //获取用户名和密码
    char iptNameMsg[]="Please input username: ";
    SSL_write(ssl, iptNameMsg, strlen(iptNameMsg)+1);
    char username[BUFF_SIZE];
    int len = SSL_read(ssl, username, BUFF_SIZE);

```

```

char iptPasswdMsg[]="Please input password: ";
SSL_write(ssl, iptPasswdMsg, strlen(iptPasswdMsg)+1);
char passwd[BUFF_SIZE];
len = SSL_read(ssl, passwd, BUFF_SIZE);

int r = login(username, passwd);
if(r != 1){
    char no[] = "Client verify failed";
    printf("%s\n",no);
    SSL_write(ssl, no, strlen(no)+1);
    return -1;
}
char yes[] = "Client verify succeed";
printf("%s\n",yes);
SSL_write(ssl, yes, strlen(yes)+1);
return 1;
}

```

如 4-25 图所示，当客户端输入正确用户名和密码时，服务端和客户端都会显示验证成功。

```

[05/25/21]seed@VM:~/.../miniVPN$ sudo docker cp ../miniVPN/ HostU:/
[05/25/21]seed@VM:~/.../miniVPN$ sudo ./vpnserv 4789
Enter PEM pass phrase:
Private key match the certificate public key
listenSock = 3
sockfd = 4
Connection from IP:10.0.2.7 port:14471
SSL_accept return 1
SSL connection established!
SSL connection using AES256-GCM-SHA384
Login name: seed
Client verify succeed

root@HostU: /miniVPN
SSL connection is successful
SSL connection using AES256-GCM-SHA384
Please input username:
seed
Please input password:
****
Client verify succeed
virtualIP: 192.168.53.128/24
Create a tun device :tun0

```

图 4-25 客户端验证

4.6 任务 6：支持多个客户端

对于服务端支持多客户端，主要需要解决的问题是当有数据包由服务端发送给客户端时，如何选择发送给指定的客户端。

此处个人选取的解决方案是对于每个客户端，服务端都会创建一个与之对应的虚拟网卡设备 TUN 的方法，使得服务端发送给客户端数据时会向客户端对应的 TUN 设备发送，然后通过套接字发送到客户端。

具体来讲，在服务端，使用了多线程来处理每个来自客户端的连接。在 accept 函数成功与客户端建立套接字连接后，使用 pthread_create 函数为该连接创建一个线程进行处理。在该线程中，首先进行 TLS 的握手连接以及客户端认

证，接下来就是服务器端创建虚拟网卡设备 TUN。使用 `open("/dev/net/tun", O_RDWR)` 来创建一个 TUN 设备，然后根据系统创建的 TUN 编号为服务端 TUN 接口设置一个虚拟 IP，并同时计算得到一个给客户端的 TUN 接口的虚拟 IP。接下来，使用 `system` 函数设置 3 条命令：使用 `ipconfig` 命令将服务端 TUN 接口的 IP 绑定到相应的 TUN 设备；使用 `route add` 命令添加一条路由，将服务端收到的由客户端 TUN 接口虚拟 IP 的数据包交由相应的服务端 TUN 设备处理；最后使用 `sysctl` 命令设置当前主机为报文转发模式。紧接着，将上述分配给客户端 TUN 接口的虚拟 IP 由套接字发送给客户端。最后使用 `select` 函数监听客户端对应的套接字和 TUN 设备进行数据的收发。服务端代码中和多客户端相关的部分如下：

```
//线程处理函数
void *threadFunc(void *arg) {
    int sockfd = (int)arg;
    /*-----TLS handshake -----*/
    // ...
    /*-----Verify client -----*/
    // ...
    /*-----Create TUN device -----*/
    //创建虚拟设备 TUN
    int virtualIP;
    int tunfd = createTunDevice(ssl, &virtualIP);
    if(tunfd == -1) return NULL;
    /*-----Send Virtual IP -----*/
    sendVirtualIP(ssl,virtualIP);
    /*-----Send/Receive data -----*/
    //select 监听套接字和虚拟设备
    selectTunnel(ssl, sockfd, tunfd);
    // ...
}

//创建虚拟网卡设备
//虚拟网卡绑定一个设备以及一个 TCP 套接字
int createTunDevice(SSL* ssl, int* virtualIP) {
    struct ifreq ifr;
    memset(&ifr, 0, sizeof(ifr));

    ifr.ifr_flags = IFF_TUN | IFF_NO_PI;
    //IFF_TUN:表示创建一个 TUN 设备
    //IFF_NO_PI:表示不包含包头信息

    //创建虚拟网卡设备
    //此处由系统自己找合适的名称，可用名称是一个共享的资源，需要加锁
    pthread_mutex_lock(&mutex);
    int tunfd = open("/dev/net/tun", O_RDWR);
    pthread_mutex_unlock(&mutex);
```

```

if (tunfd == -1) {
    printf("Open TUN failed! (%d: %s)\n", errno, strerror(errno));
    return -1;
}

//注册设备工作模式
int ret = ioctl(tunfd, TUNSETIFF, &ifr);
if (ret == -1) {
    printf("Setup TUN interface by ioctl failed! (%d: %s)\n", errno, strerror(errno));
    return -1;
}
printf("Create a tun device :%s\n", ifr.ifr_name); //tunXXX

//虚拟设备编号
int tunId = atoi(ifr.ifr_name+3);
if(tunId == 127) {
    printf("Exceed the maximum number of clients!\n");
    return -1;
}

//根据网卡名称配置服务端 TUN 设备的虚拟 IP(IP=192.168.53.1+tunId)
char cmd[60];
sprintf(cmd,"sudo ifconfig tun%d 192.168.53.%d/24 up",tunId,tunId+1);
// printf("%s\n",cmd);
//根据虚拟设备设定分配给客户端 TUN 的虚拟 IP 地址, 并设置路由走当前创建的虚拟设备
system(cmd);
sprintf(cmd,"route add -host 192.168.53.%d tun%d",tunId+127,tunId);
// printf("%s\n",cmd);
system(cmd);
system("sudo sysctl net.ipv4.ip_forward=1");

*virtualIP = tunId + 127; //分配给客户端 TUN 接口的虚拟 IP
return tunfd;
}
//发送给客户端其虚拟 IP
void sendVirtualIP(SSL* ssl, int virtualIP) {
    char buf[10];
    sprintf(buf,"%d",virtualIP);
    printf("send virtual IP: 192.168.53.%s/24\n",buf);
    SSL_write(ssl,buf,strlen(buf)+1);
}

```

在客户端, 并不需要做出太多改动, 不过此处客户端的 TUN 接口的虚拟 IP 不是之前固定的 IP 了, 而是需要使用服务端发送给它的虚拟 IP。因此, 客

户端在创建 TUN 设备前，先从套接字中读取服务端发给它的虚拟 IP 地址，将该 IP 地址与自己创建的 TUN 设备绑定在一起。客户端代码中和多客户端相关的部分如下：

```
int main(int argc, char *argv[]) {
    // ...
    /*-----TLS initialization -----*/
    SSL *ssl = setupTLSClient(hostname);
    /*-----Create a TCP connection -----*/
    // ...
    /*-----TLS handshake -----*/
    // ...
    /*-----Verify client -----*/
    // ...
    /*-----Receive Virtual IP -----*/
    int virtualIP = recvVirtualIP(ssl);

    /*-----Create TUN device -----*/
    int tunfd = createTunDevice(virtualIP);
    /*-----Send/Receive data -----*/
    selectTunnel(ssl,sockfd,tunfd);
    // ...
}
//获取服务端分配的虚拟 IP
int recvVirtualIP(SSL* ssl) {
    char buf[10];
    SSL_read(ssl,buf,BUFF_SIZE);
    int virtualIP = atoi(buf);
    printf("virtualIP: 192.168.53.%d/24\n",virtualIP);
    return virtualIP;
}
//创建虚拟网卡设备
//虚拟网卡绑定一个设备以及一个 TCP 套接字
int createTunDevice(int virtualIP) {
    int tunfd;
    struct ifreq ifr;
    memset(&ifr, 0, sizeof(ifr));

    ifr.ifr_flags = IFF_TUN | IFF_NO_PI;
    //IFF_TUN:表示创建一个 TUN 设备
    //IFF_NO_PI:表示不包含包头信息

    //打开 TUN 设备
    tunfd = open("/dev/net/tun", O_RDWR);
    if (tunfd == -1) {
```

```

        printf("Open /dev/net/tun failed! (%d: %s)\n", errno, strerror(errno));
        return -1;
    }
    //注册设备工作模式
    int ret = ioctl(tunfd, TUNSETIFF, &ifr);
    if (ret == -1) {
        printf("Setup TUN interface by ioctl failed! (%d: %s)\n", errno, strerror(errno));
        return -1;
    }
    printf("Create a tun device :%s\n", ifr.ifr_name);
    //虚拟设备编号
    int tunId = atoi(ifr.ifr_name+3);

    char cmd[60];
    //将虚拟 IP 绑定到 TUN 设备上
    sprintf(cmd,"sudo ifconfig tun%d 192.168.53.%d/24 up",tunId, virtualIP);
    system(cmd);
    //将发送给 192.168.60.0/24 的数据包交由 TUN 设备处理
    sprintf(cmd,"sudo route add -net 192.168.60.0/24 dev tun%d",tunId);
    system(cmd);
    return tunfd;
}

```

总体而言，服务端对多客户端的处理是通过为每个客户端创建对应的 TUN 设备来实现的。如图 4-26 所示，外网主机 HostU 和 HostU2 同时运行客户端程序，并为两个客户端分配了不同的虚拟 IP。此时 HostU 和 HostU2 分别使用 ping 和 telnet 与内网 HostV 通信，可以正常运行，如图 4-27 所示。

```

root@HostU: /miniVPN
Please input username:
seed
Please input password:
****
Client verify succeed
virtualIP: 192.168.53.128/24
Create a tun device :tun0
[ ->tunnel ] Got a 48-byte packet from TUN and will write in socket
[ <-tunnel ] Got a 48-byte packet from socket and will write in TUN
[ ->tunnel ] Got a 48-byte packet from TUN and will write in socket

root@HostU2: /miniVPN
Please input password:
****
Client verify succeed
virtualIP: 192.168.53.129/24
Create a tun device :tun0
[ ->tunnel ] Got a 48-byte packet from TUN and will write in socket
[ <-tunnel ] Got a 48-byte packet from socket and will write in TUN
[ <-tunnel ] Got a 48-byte packet from socket and will write in TUN

```

图 4-26 在 HostU 和 HostU 运行 VPN 客户端

```
root@HostU: /
root@HostU:/# telnet 192.168.60.101
Trying 192.168.60.101...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
HostV login: |

root@HostU2: /
root@HostU2:/# ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data:
64 bytes from 192.168.60.101: icmp_seq=1 ttl=63 time=0.387 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=0.225 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=0.333 ms
64 bytes from 192.168.60.101: icmp_seq=4 ttl=63 time=0.266 ms
```

图 4-27 HostU 和 HostU2 同时与 HostV 通信

5 实验思考

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。正确重连后, telnet 连接会发生什么? 会被断开还是继续?

解答:

在断开 HostU 和 VM 的 VPN 隧道后, 在 telnet 窗口输入内容时无反应, 字符也不会显示。经过分析, 这是由于 VPN 关闭后, 内外网之间的套接字也就关闭了, 外网的数据已经无法发送到内网了, 自然内网 HostV 也没有办法对外网主机 HostU 响应, telnet 也就没有回显字符。如图 5-1 所示。而重新建立隧道后仍旧窗口没有反应。

```
Got a packet from the tunnel [05/08/21]seed@VM:~/.../vpn$ sudo killall vpnserver
killall vpnclient [sudo] password for seed:
root@HostU:/# | [05/08/21]seed@VM:~/.../vpn$ |
index.html.1
[05/08/21]seed@HostV:~$ pwd
/home/seed
[05/08/21]seed@HostV:~$ |
```

图 5-1 断开 VPN 隧道后的 telnet

心得体会与建议

1 心得体会

本次 VPN 实验让我对 VPN 的实现原理有了更深的认识与理解，了解了 VPN 隧道是如何实现外网与内网主机进行通信的，巩固了课上所学的知识。同时通过进行编程，使用 OpenSSL 进行隧道加密和服务端证书认证，让我对 SSL 数据加密及其连接握手和证书的生成与认证的原理有了进一步的认识，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，运用了 Linux 的多线程，同时在对描述符监听时使用了 select 函数进行多路复用，这个过程中也学习了 Linux 编程一些技能，总体上来说收获颇丰。

2 建议

实验中会出现各种各样的问题，希望老师能够在讲解实验时就进行提醒，避免在一些不必要的地方浪费太多时间。