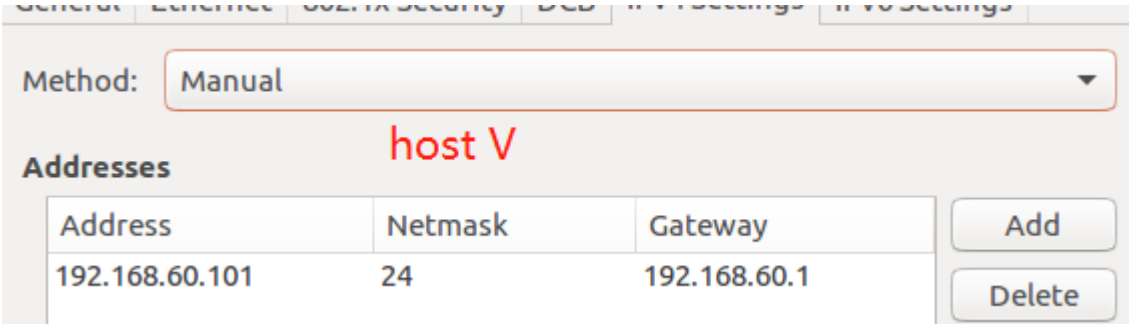
VPN and Bypassing Firewalls using VPN Lab Report

> Host U 10.0.2.4 Host V 192.168.60.101 Gateway enp0s3: 10.0.2.5 enp0s8:192.168.60.1 Host U： tun0 192.168.53.5 VPN Server tun0:192.168.53.1

Task1: VM Setup

因为在vpn server和 host V之间设置的是Internal Network网络，所以我们需要给VPN server的Internal Network网卡和host V 配置DHCP（动态主机设置协议），示例如下图：



Task2: Creating a VPN Tunnel using TUN/TAP

Step 1: Run VPN Server.

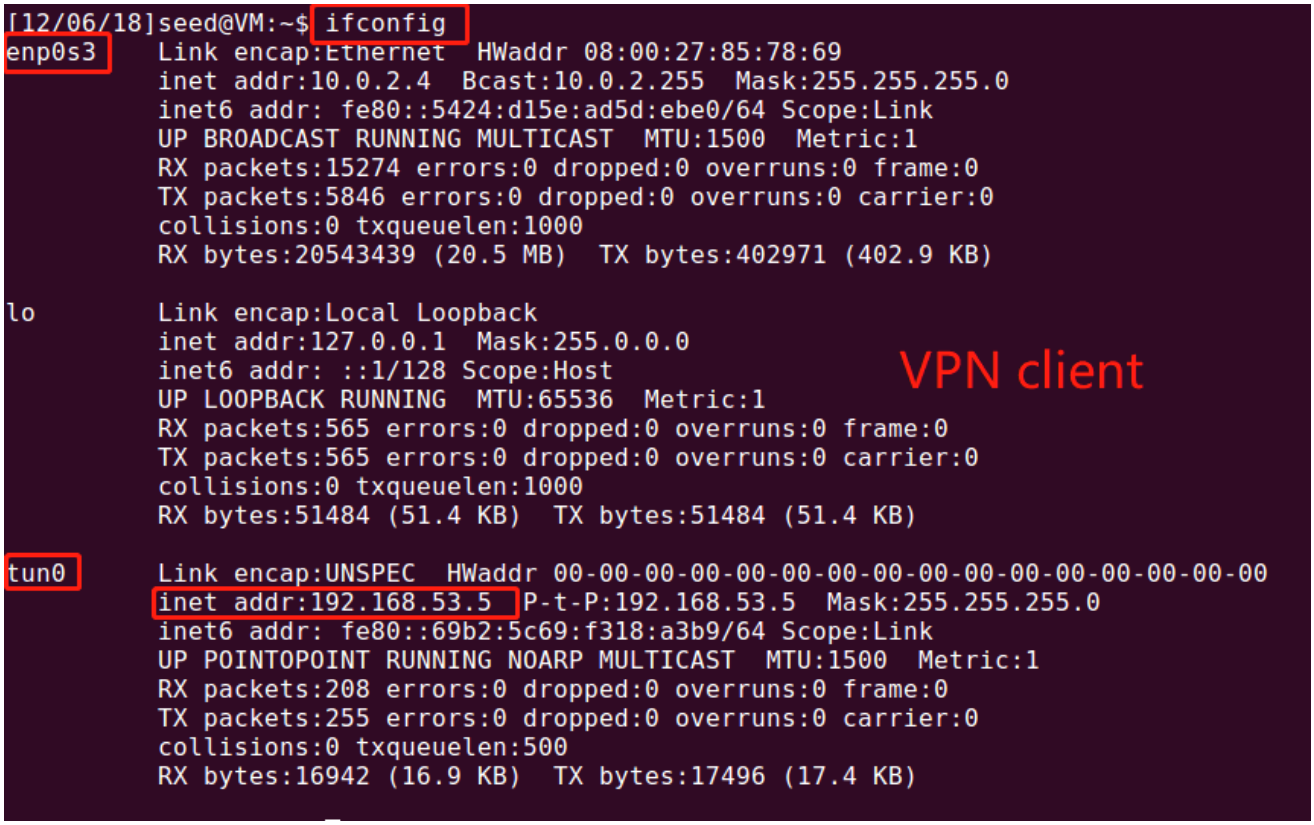
启动虚拟网卡并给它配置IP地址后，证据如图所示：

```
[12/06/18]seed@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:59:19:50
          inet addr:10.0.2.5  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::3d6:795d:8cd0:230d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1776 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1739 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:271670 (271.6 KB)  TX bytes:159551 (159.5 KB)

enp0s8    Link encap:Ethernet  HWaddr 08:00:27:61:5c:93
          inet addr:192.168.60.1  Bcast:192.168.60.255  Mask:255.255.255.0
          inet6 addr: fe80::d3:75d3:f4ae:39d8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:762 errors:0 dropped:0 overruns:0 frame:0
          TX packets:752 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:63089 (63.0 KB)  TX bytes:58151 (58.1 KB)

lo        Link encap:Local Loopback            VPN server
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:768 errors:0 dropped:0 overruns:0 frame:0
          TX packets:768 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:55572 (55.5 KB)  TX bytes:55572 (55.5 KB)

tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:192.168.53.1  P-t-P:192.168.53.1  Mask:255.255.255.0
          inet6 addr: fe80::a55f:b7b5:4575:8a0/64 Scope:Link
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:249 errors:0 dropped:0 overruns:0 frame:0
          TX packets:213 errors:0 dropped:0 overruns:0 carrier:0
```

Step 2: Run VPN Client.

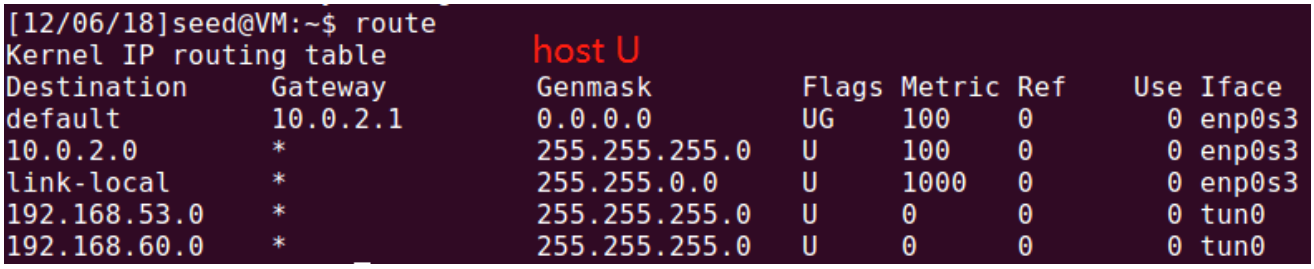
观察给的代码，发现将其连接IP设成127.0.0.1，与文档稍有差别，所以将其改为VPN server的外部连接地址10.0.2.5，运行程序，挂上IP后，显示结果：

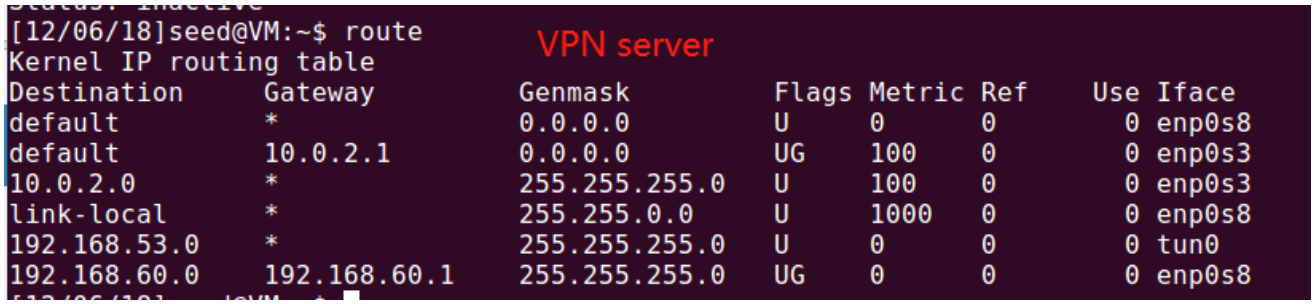
```
[12/06/18]seed@VM:~/.../vpn$ sudo ./vpnclient
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
```



Step 3: Set Up Routing on Client and Server VM

使用相关命令设置路由表项结果如下：





Step 4: Set Up Routing on Host V.

在host V 让其将从host U 来的报文转发到VPN服务器上，其实不需要这个命令也可以，因为外部网络的链接必定通过网关路由器。host V 命令：

```
sudo route add -net 192.168.53.0/24 gw 192.168.60.1 dev enp0s3
```

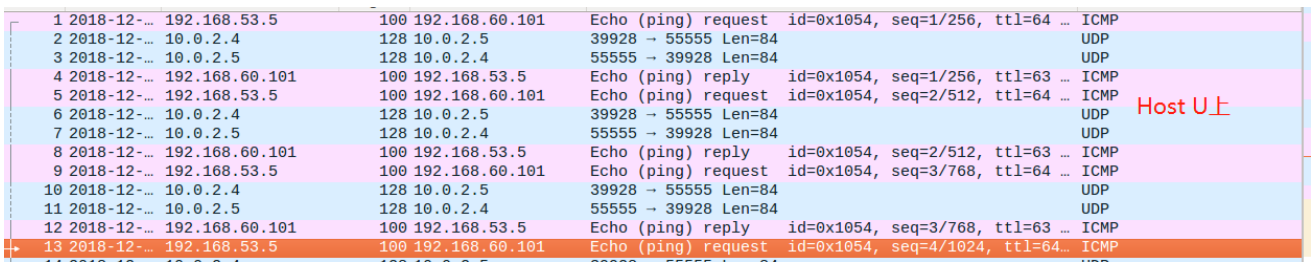
```
[12/06/18]seed@VM:~$ route
Kernel IP routing table                    host V
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         192.168.60.1    0.0.0.0         UG    100    0        0 enp0s3
link-local      *               255.255.0.0     U     1000   0        0 enp0s3
192.168.53.0    192.168.60.1    255.255.255.0   UG    0      0        0 enp0s3
192.168.60.0    *               255.255.255.0   U     100    0        0 enp0s3
```

Step 5: Test the VPN Tunnel

ping和telnet结果：

```
[12/06/18]seed@VM:~$ 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=1.12 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=2.68 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=1.19 ms
^C
--- 192.168.60.101 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2016ms
rtt min/avg/max/mdev = 1.126/1.667/2.684/0.720 ms
[12/06/18]seed@VM:~$ telnet 192.168.60.101
Trying 192.168.60.101...
Connected to 192.168.60.101.
Escape character is '^]'.                    host U 上
Ubuntu 16.04.5 LTS
VM login: seed
Password:
Last login: Wed Dec  5 08:03:05 EST 2018 from 192.168.60.1 on pts/17
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

49 packages can be updated.
14 updates are security updates.

12/06/18]seed@VM:~$ exit
ogout
onnection closed by foreign host.
```

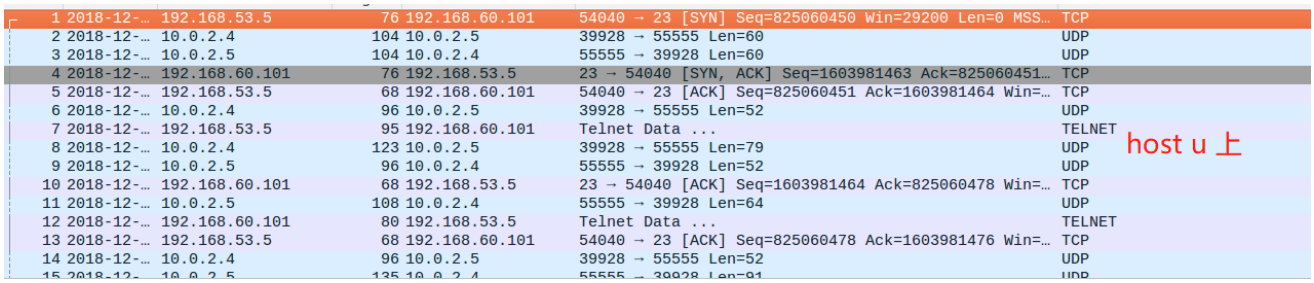
在主机U上抓取的报文：



1	2018-12-…	192.168.53.5	100	192.168.60.101	Echo (ping) request id=0x1054, seq=1/256, ttl=64 …	ICMP
2	2018-12-…	10.0.2.4	128	10.0.2.5	39928 → 55555 Len=84	UDP
3	2018-12-…	10.0.2.5	128	10.0.2.4	55555 → 39928 Len=84	UDP
4	2018-12-…	192.168.60.101	100	192.168.53.5	Echo (ping) reply id=0x1054, seq=1/256, ttl=63 …	ICMP
5	2018-12-…	192.168.53.5	100	192.168.60.101	Echo (ping) request id=0x1054, seq=2/512, ttl=64 …	ICMP
6	2018-12-…	10.0.2.4	128	10.0.2.5	39928 → 55555 Len=84	UDP
7	2018-12-…	10.0.2.5	128	10.0.2.4	55555 → 39928 Len=84	UDP
8	2018-12-…	192.168.60.101	100	192.168.53.5	Echo (ping) reply id=0x1054, seq=2/512, ttl=63 …	ICMP
9	2018-12-…	192.168.53.5	100	192.168.60.101	Echo (ping) request id=0x1054, seq=3/768, ttl=64 …	ICMP
10	2018-12-…	10.0.2.4	128	10.0.2.5	39928 → 55555 Len=84	UDP
11	2018-12-…	10.0.2.5	128	10.0.2.4	55555 → 39928 Len=84	UDP
12	2018-12-…	192.168.60.101	100	192.168.53.5	Echo (ping) reply id=0x1054, seq=3/768, ttl=63 …	ICMP
13	2018-12-…	192.168.53.5	100	192.168.60.101	Echo (ping) request id=0x1054, seq=4/1024, ttl=64…	ICMP

数据包1：由ping命令生成。 由于路由设置，ICMP数据包被路由到TUN接口（目的IP为192.168.60.0/24->分配给tun0的原因） 数据包2：隧道应用程序获取ICMP数据包，然后将其提供给其隧道 - >将其放入UDP数据包中，朝向VPN服务器（10.0.2.5）。 数据包No 3：来自VPN Server->的返回UDP数据包是来自192.168.60.101的封装ICMP回应数据包 数据包No 4：VPN Client上的隧道应用程序获取此UDP数据包，并取出封装的ICMP数据包，并通过tun0接口将其提供给内核。 数据包No 5到8：由另一个ICMP回应请求消息触发。

telnet解释通上：



1	2018-12-…	192.168.53.5	76	192.168.60.101	54040 → 23 [SYN] Seq=825060450 Win=29200 Len=0 MSS…	TCP
2	2018-12-…	10.0.2.4	104	10.0.2.5	39928 → 55555 Len=60	UDP
3	2018-12-…	10.0.2.5	104	10.0.2.4	55555 → 39928 Len=60	UDP
4	2018-12-…	192.168.60.101	76	192.168.53.5	23 → 54040 [SYN, ACK] Seq=1603981463 Ack=825060451…	TCP
5	2018-12-…	192.168.53.5	68	192.168.60.101	54040 → 23 [ACK] Seq=825060451 Ack=1603981464 Win=…	TCP
6	2018-12-…	10.0.2.4	96	10.0.2.5	39928 → 55555 Len=52	UDP
7	2018-12-…	192.168.53.5	95	192.168.60.101	Telnet Data ...	TELNET
8	2018-12-…	10.0.2.4	123	10.0.2.5	39928 → 55555 Len=79	UDP
9	2018-12-…	10.0.2.5	96	10.0.2.4	55555 → 39928 Len=52	UDP
10	2018-12-…	192.168.60.101	68	192.168.53.5	23 → 54040 [ACK] Seq=1603981464 Ack=825060478 Win=…	TCP
11	2018-12-…	10.0.2.5	108	10.0.2.4	55555 → 39928 Len=64	UDP
12	2018-12-…	192.168.60.101	80	192.168.53.5	Telnet Data ...	TELNET
13	2018-12-…	192.168.53.5	68	192.168.60.101	54040 → 23 [ACK] Seq=825060478 Ack=1603981476 Win=…	TCP
14	2018-12-…	10.0.2.4	96	10.0.2.5	39928 → 55555 Len=52	UDP

Step6: Tunnel-Breaking Test.

当使用命令：

```
sudo ifconfig tun0 192.168.53.1/24 down
```

将通道断了之后，发现telnet完全没反应了，观察流量包：

33	2018-12-…	192.168.53.5	69	192.168.60.101	Telnet Data ...	TELNET
34	2018-12-…	10.0.2.4	97	10.0.2.5	41884 → 55555 Len=53	UDP
35	2018-12-…	192.168.53.5	70	192.168.60.101	[TCP Retransmission] 44226 → 23 [PSH, ACK] Seq=148…	TCP
36	2018-12-…	10.0.2.4	98	10.0.2.5	41884 → 55555 Len=54	UDP
37	2018-12-…	192.168.53.5	70	192.168.60.101	[TCP Retransmission] 44226 → 23 [PSH, ACK] Seq=148…	TCP
38	2018-12-…	10.0.2.4	98	10.0.2.5	41884 → 55555 Len=54	UDP
39	2018-12-…	192.168.53.5	70	192.168.60.101	[TCP Retransmission] 44226 → 23 [PSH, ACK] Seq=148…	TCP
40	2018-12-…	10.0.2.4	98	10.0.2.5	41884 → 55555 Len=54	UDP
41	2018-12-…	192.168.53.5	70	192.168.60.101	[TCP Retransmission] 44226 → 23 [PSH, ACK] Seq=148…	TCP
42	2018-12-…	10.0.2.4	98	10.0.2.5	41884 → 55555 Len=54	UDP
43	2018-12-…	192.168.53.5	70	192.168.60.101	[TCP Retransmission] 44226 → 23 [PSH, ACK] Seq=148…	TCP
44	2018-12-…	10.0.2.4	98	10.0.2.5	41884 → 55555 Len=54	UDP

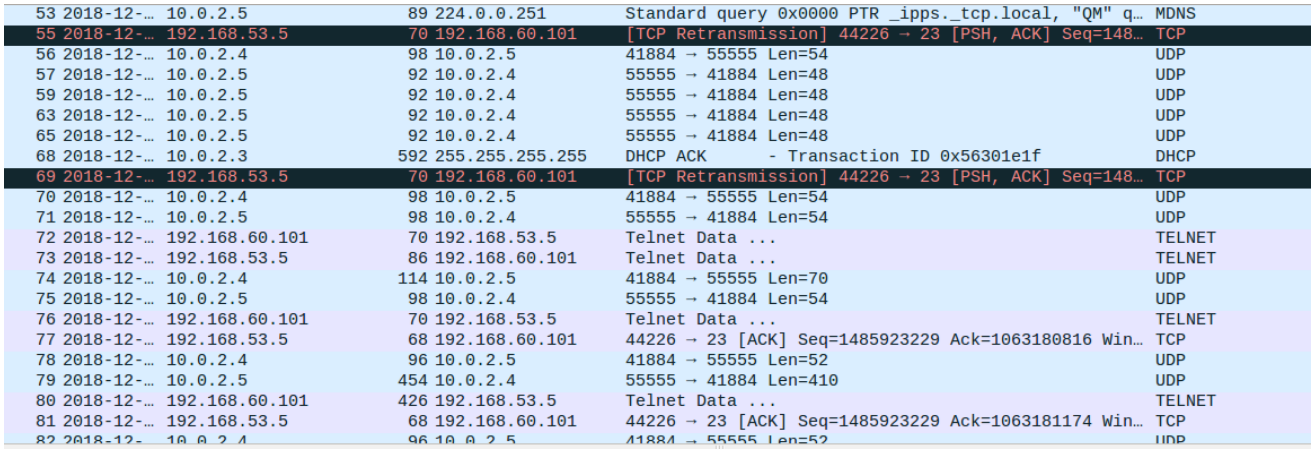
说明telnet仍在工作，但由于它通过损坏的VPN隧道发送的数据包不在任何地方，TCP将继续重新发送数据包。 VPN Server将丢弃UDP数据包并发回ICMP错误消息，告知VPN客户端端口不可访问。 这就是我们看到多个ICMP错误消息的原因。

当重新使用命令：

```
sudo ifconfig tun0 192.168.53.1/24 up
```

后，观察流量包以及显示屏，发现又可以正常化使用连接：

38	2018-12-…	10.0.2.4	98	10.0.2.5	41884 → 55555 Len=54	UDP
39	2018-12-…	192.168.53.5	70	192.168.60.101	[TCP Retransmission] 44226 → 23 [PSH, ACK] Seq=148…	TCP
40	2018-12-…	10.0.2.4	98	10.0.2.5	41884 → 55555 Len=54	UDP
41	2018-12-…	192.168.53.5	70	192.168.60.101	[TCP Retransmission] 44226 → 23 [PSH, ACK] Seq=148…	TCP
42	2018-12-…	10.0.2.4	98	10.0.2.5	41884 → 55555 Len=54	UDP
43	2018-12-…	192.168.53.5	70	192.168.60.101	[TCP Retransmission] 44226 → 23 [PSH, ACK] Seq=148…	TCP
44	2018-12-…	10.0.2.4	98	10.0.2.5	41884 → 55555 Len=54	UDP
46	2018-12-…	10.0.2.4	92	10.0.2.5	41884 → 55555 Len=48	UDP
47	2018-12-…	192.168.53.5	70	192.168.60.101	[TCP Retransmission] 44226 → 23 [PSH, ACK] Seq=148…	TCP
48	2018-12-…	10.0.2.4	98	10.0.2.5	41884 → 55555 Len=54	UDP
49	2018-12-…	192.168.53.5	70	192.168.60.101	[TCP Retransmission] 44226 → 23 [PSH, ACK] Seq=148…	TCP
50	2018-12-…	10.0.2.4	98	10.0.2.5	41884 → 55555 Len=54	UDP
53	2018-12-…	10.0.2.5	89	224.0.0.251	Standard query 0x0000 PTR _ipps._tcp.local, "QM" q…	MDNS
55	2018-12-…	192.168.53.5	70	192.168.60.101	[TCP Retransmission] 44226 → 23 [PSH, ACK] Seq=148…	TCP
56	2018-12-…	10.0.2.4	98	10.0.2.5	41884 → 55555 Len=54	UDP
57	2018-12-…	10.0.2.5	92	10.0.2.4	55555 → 41884 Len=48	UDP
59	2018-12-…	10.0.2.5	92	10.0.2.4	55555 → 41884 Len=48	UDP
63	2018-12-…	10.0.2.5	92	10.0.2.4	55555 → 41884 Len=48	UDP
65	2018-12-…	10.0.2.5	92	10.0.2.4	55555 → 41884 Len=48	UDP
68	2018-12-…	10.0.2.3	592	255.255.255.255	DHCP ACK - Transaction ID 0x56301e1f	DHCP
69	2018-12-…	192.168.53.5	70	192.168.60.101	[TCP Retransmission] 44226 → 23 [PSH, ACK] Seq=148…	TCP



53	2018-12-…	10.0.2.5	89	224.0.0.251	Standard query 0x0000 PTR _ipps._tcp.local, "QM" q…	MDNS
55	2018-12-…	192.168.53.5	70	192.168.60.101	[TCP Retransmission] 44226 → 23 [PSH, ACK] Seq=148…	TCP
56	2018-12-…	10.0.2.4	98	10.0.2.5	41884 → 55555 Len=54	UDP
57	2018-12-…	10.0.2.5	92	10.0.2.4	55555 → 41884 Len=48	UDP
59	2018-12-…	10.0.2.5	92	10.0.2.4	55555 → 41884 Len=48	UDP
63	2018-12-…	10.0.2.5	92	10.0.2.4	55555 → 41884 Len=48	UDP
65	2018-12-…	10.0.2.5	92	10.0.2.4	55555 → 41884 Len=48	UDP
68	2018-12-…	10.0.2.3	592	255.255.255.255	DHCP ACK - Transaction ID 0x56301e1f	DHCP
69	2018-12-…	192.168.53.5	70	192.168.60.101	[TCP Retransmission] 44226 → 23 [PSH, ACK] Seq=148…	TCP
70	2018-12-…	10.0.2.4	98	10.0.2.5	41884 → 55555 Len=54	UDP
71	2018-12-…	10.0.2.5	98	10.0.2.4	55555 → 41884 Len=54	UDP
72	2018-12-…	192.168.60.101	70	192.168.53.5	Telnet Data ...	TELNET
73	2018-12-…	192.168.53.5	86	192.168.60.101	Telnet Data ...	TELNET
74	2018-12-…	10.0.2.4	114	10.0.2.5	41884 → 55555 Len=70	UDP
75	2018-12-…	10.0.2.5	98	10.0.2.4	55555 → 41884 Len=54	UDP
76	2018-12-…	192.168.60.101	70	192.168.53.5	Telnet Data ...	TELNET
77	2018-12-…	192.168.53.5	68	192.168.60.101	44226 → 23 [ACK] Seq=1485923229 Ack=1063180816 Win…	TCP
78	2018-12-…	10.0.2.4	96	10.0.2.5	41884 → 55555 Len=52	UDP
79	2018-12-…	10.0.2.5	454	10.0.2.4	55555 → 41884 Len=410	UDP
80	2018-12-…	192.168.60.101	426	192.168.53.5	Telnet Data ...	TELNET
81	2018-12-…	192.168.53.5	68	192.168.60.101	44226 → 23 [ACK] Seq=1485923229 Ack=1063181174 Win…	TCP
82	2018-12-…	10.0.2.4	96	10.0.2.5	41884 → 55555 Len=52	UDP

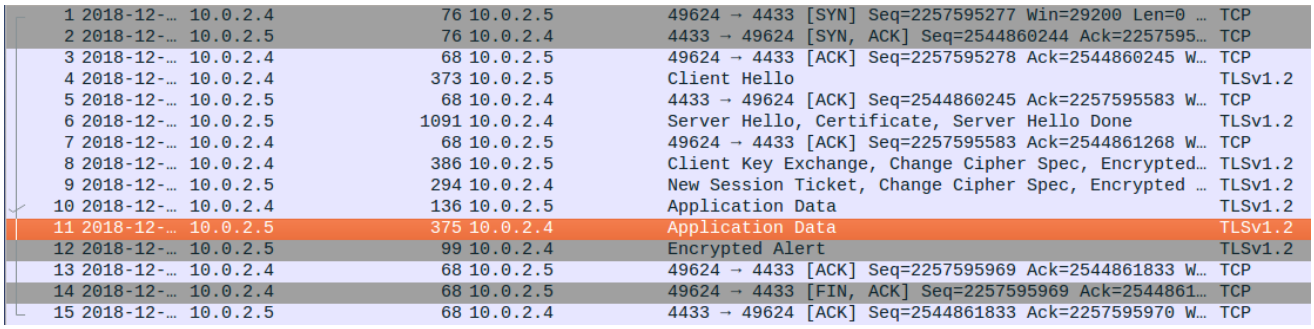
无论输入到telnet中的是什么都没有丢失，它们被缓冲，等待发送到telnet服务器。

Task3: Encrypting the Tunnel

首先，在host U上/etc/hosts 中 加入：

```
10.0.2.5        vpnlabserver.com
```

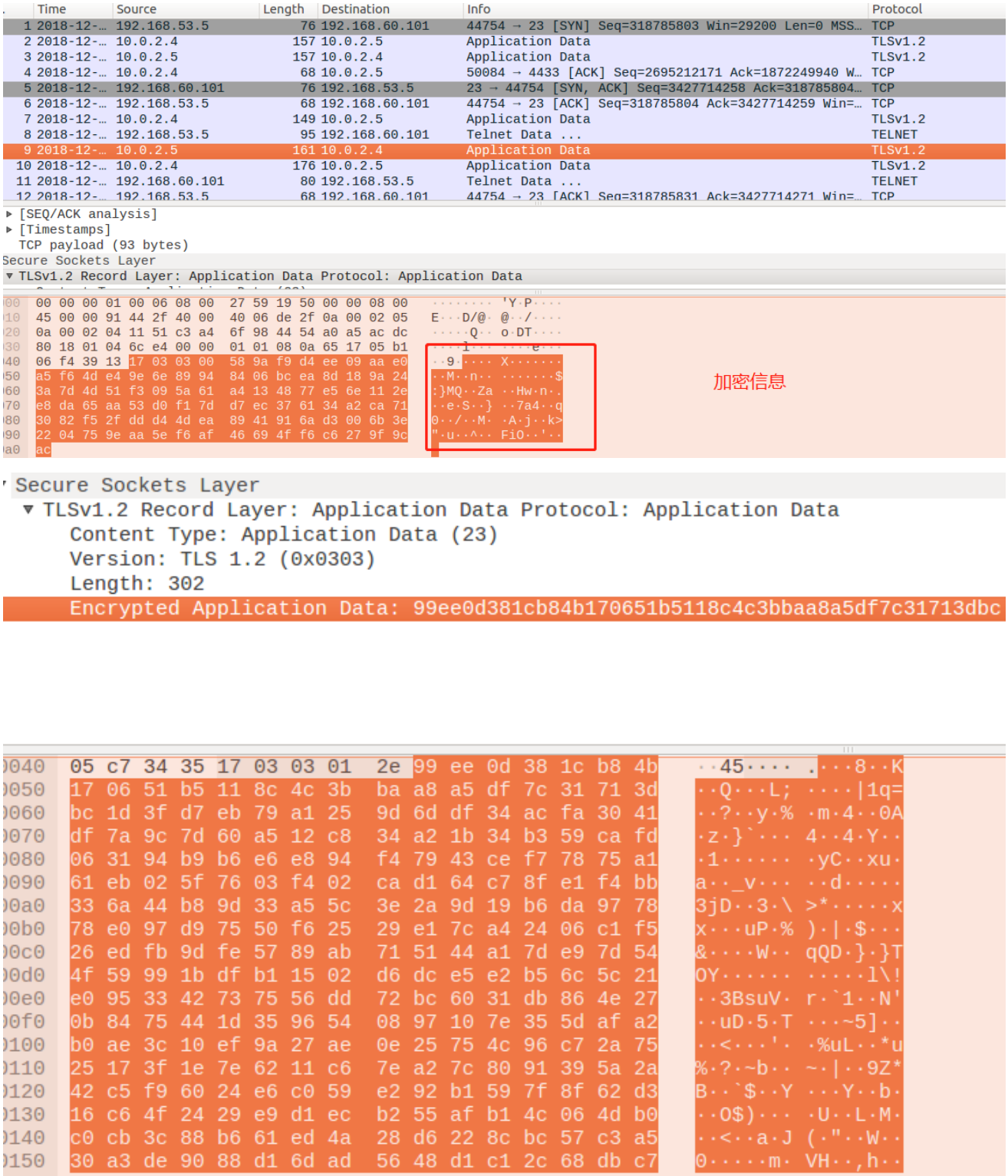
然后根据readme的指导，进行测试，获取的报文如下：



1	2018-12-…	10.0.2.4	76	10.0.2.5	49624 → 4433 [SYN] Seq=2257595277 Win=29200 Len=0 …	TCP
2	2018-12-…	10.0.2.5	76	10.0.2.4	4433 → 49624 [SYN, ACK] Seq=2544860244 Ack=2257595…	TCP
3	2018-12-…	10.0.2.4	68	10.0.2.5	49624 → 4433 [ACK] Seq=2257595278 Ack=2544860245 W…	TCP
4	2018-12-…	10.0.2.4	373	10.0.2.5	Client Hello	TLSv1.2
5	2018-12-…	10.0.2.5	68	10.0.2.4	4433 → 49624 [ACK] Seq=2544860245 Ack=2257595583 W…	TCP
6	2018-12-…	10.0.2.5	1091	10.0.2.4	Server Hello, Certificate, Server Hello Done	TLSv1.2
7	2018-12-…	10.0.2.4	68	10.0.2.5	49624 → 4433 [ACK] Seq=2257595583 Ack=2544861268 W…	TCP
8	2018-12-…	10.0.2.4	386	10.0.2.5	Client Key Exchange, Change Cipher Spec, Encrypted…	TLSv1.2
9	2018-12-…	10.0.2.5	294	10.0.2.4	New Session Ticket, Change Cipher Spec, Encrypted …	TLSv1.2
10	2018-12-…	10.0.2.4	136	10.0.2.5	Application Data	TLSv1.2
11	2018-12-…	10.0.2.5	375	10.0.2.4	Application Data	TLSv1.2
12	2018-12-…	10.0.2.5	99	10.0.2.4	Encrypted Alert	TLSv1.2
13	2018-12-…	10.0.2.4	68	10.0.2.5	49624 → 4433 [ACK] Seq=2257595969 Ack=2544861833 W…	TCP
14	2018-12-…	10.0.2.4	68	10.0.2.5	49624 → 4433 [FIN, ACK] Seq=2257595969 Ack=2544861…	TCP
15	2018-12-…	10.0.2.5	68	10.0.2.4	4433 → 49624 [ACK] Seq=2544861833 Ack=2257595970 W…	TCP

一个完成的TLS握手协议，client hello 以及 server hello，完成了各个参数，加密算法验证算法等的确认，以及改变加密方式格式，是一个完成的TLS传输过程，当握手过程完成，密钥协商完成后，使用对称加密方式传输数据，使其所有的信息都加密传输：

因为是TCP传输通道，所以比上面的UDP通道相比每次都有一个ACK确认。



Task4: Authenticating the VPN Server

> 说明，我的自签名证书Eric_hailong(cert.pem), 然后使用它给服务器签名Hailong.com (server.crt)，客户端的签名证书EricHailong.client.com(client.crt)，所有的密钥访问密码都是1234567890

首先，所有的验证都在TLS handshake阶段完成，主要函数都封装在SSL句柄里面，对于证书的认证，由函数SSL_CTX_set_verify处理，根据TLS协议，证书无效会立即断掉TLS握手过程，其他的认证都集中在SSL的句柄中，其中主机名称的认证，在函数X509_VERIFY_PARAM_set1_host中。验证服务器拥有它，在函数SSL_CTX_set_verify中，通过服务器的数字签名进行验证，而函数SSL_CTX_load_verify_locations只是设置验证证书有效性的本地根证书之一。

```
SSL* setupTLSClient(const char* hostname) {
    ...
    SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, verify_callback);
    if (SSL_CTX_load_verify_locations(ctx, NULL, CA_DIR) < 1) {
        printf("Error setting the verify locations. \n");
        exit(0);
    }
    ssl = SSL_new(ctx);
    X509_VERIFY_PARAM *vpm = SSL_get0_param(ssl);
    X509_VERIFY_PARAM_set1_host(vpm, hostname, 0);
    ...
}
```

我们可以通过回调函数，可以打印主机名称认证的过程，在这里SSL通过回调函数，将结果返回给我们，决定权在我们手中，如下图：即是认证没有通过，我们任然可以进行通信：

```
[12/07/18]seed@VM:~/.../tls$ ./tlsclient Hailong.com 4433
subject= /C=US/ST=NY/O=SYR/OU=SYR/CN=vpnlabserver.com
Verification failed: Hostname mismatch.
subject= /C=US/ST=NY/L=SYR/O=SYR/OU=SYR/CN=seedlabca.com
Verification passed.
subject= /C=US/ST=NY/O=SYR/OU=SYR/CN=vpnlabserver.com
Verification passed.
SSL connection is successful
SSL connection using AES256-GCM-SHA384
```

在这里，回调函数并没有因验证失败而exit()掉，只是将信息打印了出来：

```
int verify_callback(int preverify_ok, X509_STORE_CTX *x509_ctx){
...
    X509_NAME_oneline(X509_get_subject_name(cert), buf, 300);
    printf("subject= %s\n", buf);
    if (preverify_ok == 1) {
        printf("Verification passed.\n");
    }
    else {
        int err = X509_STORE_CTX_get_error(x509_ctx);
        printf("Verification failed: %s.\n",
            X509_verify_cert_error_string(err));
    }
}
```

成功的实例：

```
[12/07/18]seed@VM:~/.../code$ sudo ./vpn_tls_client Hailong.com
subject= /C=CN/ST=Shanghai/L=SH/O=FD/OU=NS/CN=Eric_hailong
Verification passed.
subject= /C=CN/ST=Shanghai/O=FD/OU=NS/CN=Hailong.com
Verification passed.
```

Task5: Authenticating the VPN Client

在SSL建立完成后，在客户端安全发送用户名和密码，然后服务端进行验证，代码：

```
    // 格式 sudo ./vpn_tls_client hailong:xxx@Hailong.com
    if (argc > 1&&strchr(argv[1],':')!=NULL&&strchr(argv[1],'@')!=NULL){
        i = (size_t)(strchr(argv[1],':')-argv[1]);
        j = (size_t)(strchr(argv[1],'@')-argv[1]);
        bzero(user, 12);
        bzero(passwd, 12);
        strncpy(user,argv[1],i);
        strncpy(passwd,argv[1]+i+1,j-i-1);
    }
    .....

    // -----login-----
    char buff[30];
    bzero(buff, 30);
    strncpy(buff, user, strlen(user));
    strncpy(buff+12, passwd, strlen(passwd));
    SSL_write(ssl, buff, 30);
    bzero(buff, 30);
    SSL_read(ssl, buff, sizeof(buff) - 1);
    printf("%s", buff);
```

客户端验证代码：

```
int login = Authentication(ssl);
    while (login&1) {
.....
```

结果截图：

服务器端：（为了测试，打印出了用户名和密码）

```
[12/12/18]seed@VM:~/.../code$ sudo ./vpn_tls_server
Enter PEM pass phrase:
SSL connection established!
seed
seed
dees
Login name: seed
Passwd : $6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/sfYvDeCAc
Eo2QYzCfpZoaEVJ8sbCT7hkxXY/
Got a packet from TUN
```

客户端：

```
[12/12/18]seed@VM:~/.../code$ sudo ./vpn_tls_client seed:dees@hailong.com
subject= /C=CN/ST=Shanghai/L=SH/O=FD/OU=NS/CN=Eric_hailong
Verification passed.
subject= /C=CN/ST=Shanghai/O=FD/OU=NS/CN=Hailong.com
Verification passed.
SSL connection is successful
SSL connection using AES256-GCM-SHA384
Login success!!!
```

Task6: Supporting Multiple Clients

此程序我使用的是文档所指导的使用fork函数为每一个vpn连接创建一个子进程的方法完成的。

思路：为每个client做个标识，当数据从tun来之后，根据唯一的标识使用pipe发送给子进程，然后就如一个连接的方法一样进行处理。

细节：首先，我在主函数里面使用一个数据结构来区分每个连接：

```
struct tun_route{
    pid_t pid;     //子进程pid，用来判断子进程是否已经结束，如果结束，需要释放此数据结构的内存
    unsigned int client_ip; //客户端使用的tun的ip地址，用来标识父进程发送给那个子进程的管道
    int fd[2];  //管道的fd
    struct tun_route* next;  //指向下一个链表，动态行为所必须的
};
```

主函数：

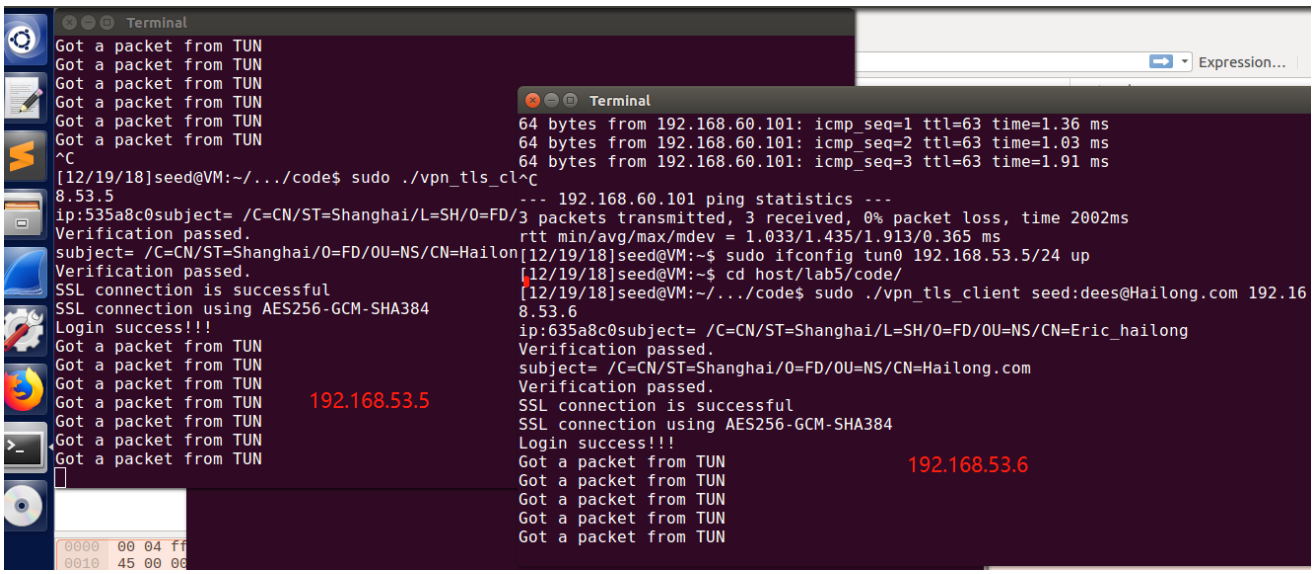
```
int main(int argc, char * argv[]) {
    SSL_METHOD *meth;
    SSL_CTX* ctx;
    SSL *ssl;
    int err;
    // Step 0: OpenSSL library initialization
    // This step is no longer needed as of version 1.1.0.
    SSL_library_init();
    SSL_load_error_strings();
    SSLeay_add_ssl_algorithms();
    // Step 1: SSL context initialization
    meth = (SSL_METHOD *)TLSv1_2_method();
    ctx = SSL_CTX_new(meth);
    SSL_CTX_set_verify(ctx, SSL_VERIFY_NONE, NULL);
    // Step 2: Set up the server certificate and private key
    SSL_CTX_use_certificate_file(ctx, "./cert_server/server.crt", SSL_FILETYPE_PEM);
    SSL_CTX_use_PrivateKey_file(ctx, "./cert_server/server.key", SSL_FILETYPE_PEM);
    // Step 3: Create a new SSL structure for a connection
    ssl = SSL_new(ctx);
    int tunfd, sockfd;
    tunfd = createTunDevice();
    int listen_sock = initTCPServer();
    struct sockaddr_in sa_client;
    size_t client_len;
    //在父进程中创建一个线程，用来一直检测tun是否有数据到达，进行tun数据的路由转发
    pthread_t t;
    if(pthread_create(&t, NULL, get_tunfd, (void *)(&tunfd))==-1){
```

```
        printf("pthread create dispath");
    }
    while (1) {
        sockfd = accept(listen_sock, (struct sockaddr*)&sa_client, &client_len);
        free_malloc();  //查询是否有子进程已经结束，如果结束，释放tun_route内存
        // 为此连接创建一个struct结构，来存储其信息
        struct tun_route* temp=tunfd_route;
        if(temp!=NULL){
            while(temp->next!=NULL){
                temp = temp->next;
            }
            temp->next = (struct tun_route*)malloc(sizeof(struct tun_route));
            temp = temp->next;
        }else{
            tunfd_route=temp= (struct tun_route*)malloc(sizeof(struct tun_route));
        }
        //建立管道
        pipe(temp->fd);
        temp->next = NULL;
        if ((temp->pid = fork())==0) { // The child process
            close(listen_sock);
            SSL_set_fd(ssl, sockfd);
            err = SSL_accept(ssl);
            if ( err != 1 ) {
                int err_SSL_get_error = SSL_get_error(ssl, err);
                int err_ERR_get_error = ERR_get_error();
                printf("[DEBUG] SSL_accept() : Failed with return %d\n", err );
                printf("[DEBUG]     SSL_get_error() returned :
%d\n",err_SSL_get_error);
                printf("[DEBUG]     Error string : %s\n",ERR_error_string(
err_SSL_get_error, NULL));
                printf("[DEBUG]     ERR_get_error() returned :
%d\n",err_ERR_get_error);
            }
            CHK_SSL(err);
            printf("SSL connection established!\n");

            // -----login-----
            int login = Authentication(ssl,temp->fd);

            // Enter the main loop
            while (login & 1) {
                fd_set readFDSet;
                FD_ZERO(&readFDSet);
                FD_SET(SSL_get_fd(ssl), &readFDSet);
                FD_SET(temp->fd[0], &readFDSet);
                select(FD_SETSIZE, &readFDSet, NULL, NULL, NULL);
                if (FD_ISSET(SSL_get_fd(ssl), &readFDSet))
                    if(socketSelected(tunfd, ssl)== 0){
                        break;  //如果通道断开，需要结束此子进程
                    }
                if (FD_ISSET(temp->fd[0], &readFDSet)) tunSelected(temp->fd[0], ssl);
            }
```

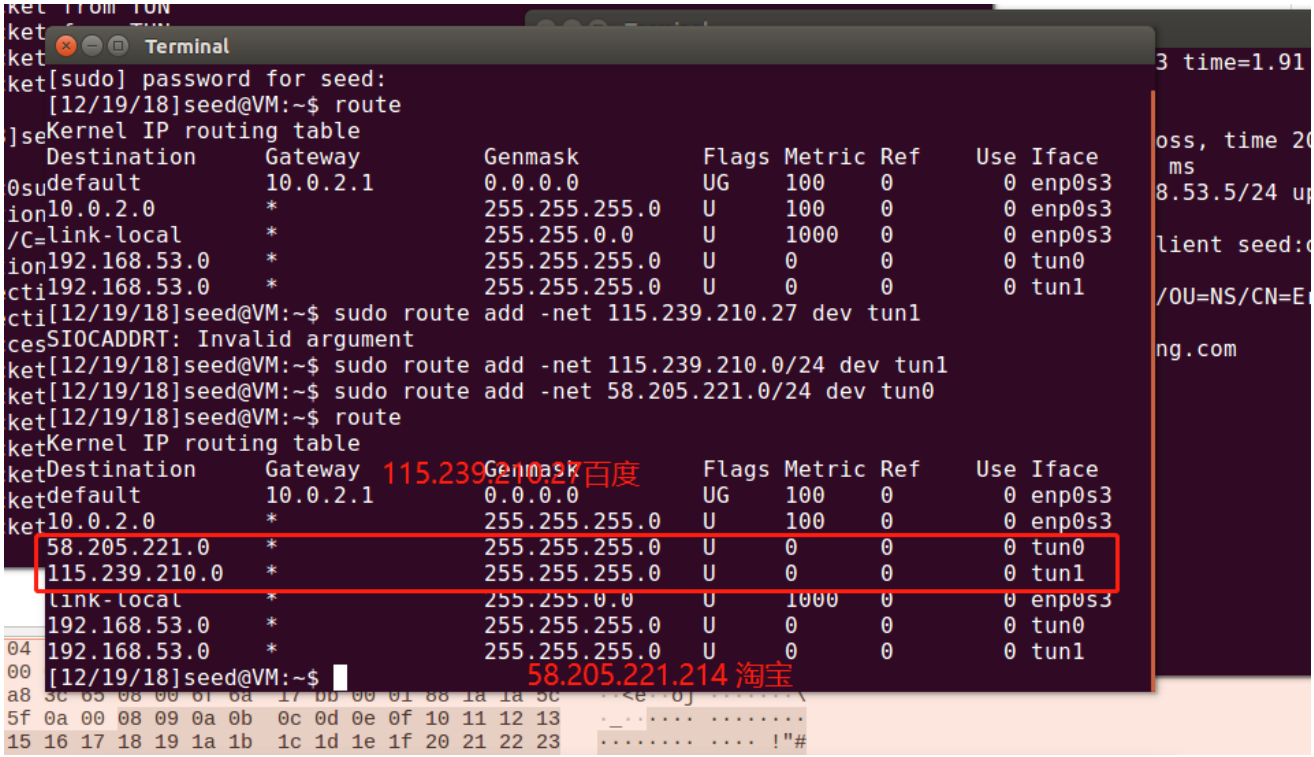
```
            SSL_shutdown(ssl);
            SSL_free(ssl);
            close(sockfd);
            return 0;
        }
        else {
            read(temp->fd[0],&(temp->client_ip),sizeof(int));
            close(temp->fd[0]);
            close(sockfd);
        }
    }
    return 0;
}
```

由于客户端的tunIP在TLS连接建立后才确认的，所以在 Authentication函数里面加了一个从子进程向父进程通知client_ip标识号的代码。因为区域ip是由服务器分配的，所以在客户端进行登录时，需要提供自己的tunIP,这一点其实可以通过服务器动态分配发送一个区域网的IP给客户端，在客户端代码里面可以使用system()函数自动配置自己的tunX网卡，不用手动配置。

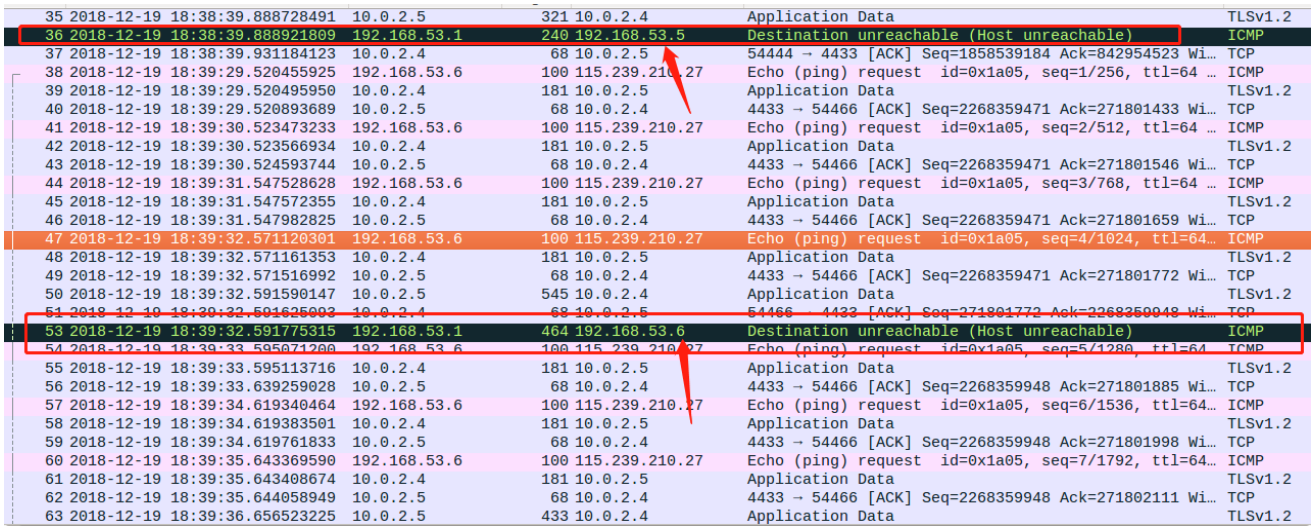
由于电脑的问题，所以我并没有使用4台虚拟机数据提供证明，为了证明程序的正确性，我在主机U上开启了两个VPN连接：



为了证明正确性，我在主机U上配置不同的route:



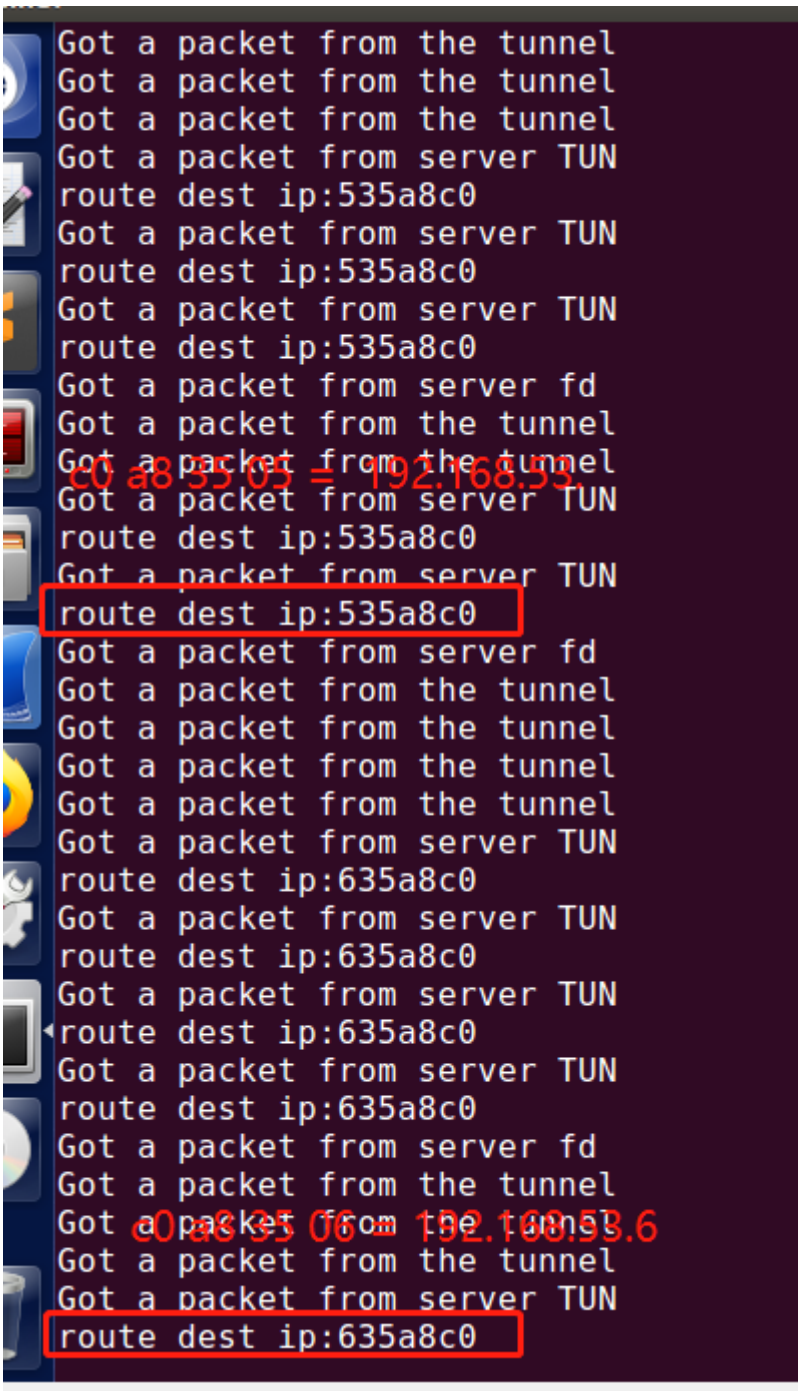
其中虚拟网卡tun1的IP是192.168.53.6.然后在主机U上使用ping 进行ping这两个IP。观察wireshark抓包：



35	2018-12-19 18:38:39.888728491	10.0.2.5	321	10.0.2.4	Application Data		TLSv1.2
36	2018-12-19 18:38:39.888921809	192.168.53.1	240	192.168.53.5	Destination unreachable (Host unreachable)		ICMP
37	2018-12-19 18:38:39.931184123	10.0.2.4	68	10.0.2.5	54444 → 4433 [ACK] Seq=1858539184 Ack=842954523 Wi…		TCP
38	2018-12-19 18:39:29.520455925	192.168.53.6	100	115.239.210.27	Echo (ping) request id=0x1a05, seq=1/256, ttl=64 …		ICMP
39	2018-12-19 18:39:29.520495950	10.0.2.4	181	10.0.2.5	Application Data		TLSv1.2
40	2018-12-19 18:39:29.520893689	10.0.2.5	68	10.0.2.4	4433 → 54466 [ACK] Seq=2268359471 Ack=271801433 Wi…		TCP
41	2018-12-19 18:39:30.523473233	192.168.53.6	100	115.239.210.27	Echo (ping) request id=0x1a05, seq=2/512, ttl=64 …		ICMP
42	2018-12-19 18:39:30.523566934	10.0.2.4	181	10.0.2.5	Application Data		TLSv1.2
43	2018-12-19 18:39:30.524593744	10.0.2.5	68	10.0.2.4	4433 → 54466 [ACK] Seq=2268359471 Ack=271801546 Wi…		TCP
44	2018-12-19 18:39:31.547528628	192.168.53.6	100	115.239.210.27	Echo (ping) request id=0x1a05, seq=3/768, ttl=64 …		ICMP
45	2018-12-19 18:39:31.547572355	10.0.2.4	181	10.0.2.5	Application Data		TLSv1.2
46	2018-12-19 18:39:31.547982825	10.0.2.5	68	10.0.2.4	4433 → 54466 [ACK] Seq=2268359471 Ack=271801659 Wi…		TCP
47	2018-12-19 18:39:32.571120301	192.168.53.6	100	115.239.210.27	Echo (ping) request id=0x1a05, seq=4/1024, ttl=64…		ICMP
48	2018-12-19 18:39:32.571161353	10.0.2.4	181	10.0.2.5	Application Data		TLSv1.2
49	2018-12-19 18:39:32.571516992	10.0.2.5	68	10.0.2.4	4433 → 54466 [ACK] Seq=2268359471 Ack=271801772 Wi…		TCP
50	2018-12-19 18:39:32.591590147	10.0.2.5	545	10.0.2.4	Application Data		TLSv1.2
51	2018-12-19 18:39:32.591625003	10.0.2.4	68	10.0.2.5	54466 → 4433 [ACK] Seq=271801772 Ack=2268359948 Wi…		TCP
53	2018-12-19 18:39:32.591775315	192.168.53.1	464	192.168.53.6	Destination unreachable (Host unreachable)		ICMP
54	2018-12-19 18:39:33.595071200	192.168.53.6	100	115.239.210.27	Echo (ping) request id=0x1a05, seq=5/1280, ttl=64		ICMP
55	2018-12-19 18:39:33.595113716	10.0.2.4	181	10.0.2.5	Application Data		TLSv1.2
56	2018-12-19 18:39:33.639259028	10.0.2.5	68	10.0.2.4	4433 → 54466 [ACK] Seq=2268359948 Ack=271801885 Wi…		TCP
57	2018-12-19 18:39:34.619340464	192.168.53.6	100	115.239.210.27	Echo (ping) request id=0x1a05, seq=6/1536, ttl=64…		ICMP
58	2018-12-19 18:39:34.619383501	10.0.2.4	181	10.0.2.5	Application Data		TLSv1.2
59	2018-12-19 18:39:34.619761833	10.0.2.5	68	10.0.2.4	4433 → 54466 [ACK] Seq=2268359948 Ack=271801998 Wi…		TCP
60	2018-12-19 18:39:35.643369590	192.168.53.6	100	115.239.210.27	Echo (ping) request id=0x1a05, seq=7/1792, ttl=64…		ICMP
61	2018-12-19 18:39:35.643408674	10.0.2.4	181	10.0.2.5	Application Data		TLSv1.2
62	2018-12-19 18:39:35.644058949	10.0.2.5	68	10.0.2.4	4433 → 54466 [ACK] Seq=2268359948 Ack=271802111 Wi…		TCP
63	2018-12-19 18:39:36.656523225	10.0.2.5	433	10.0.2.4	Application Data		TLSv1.2

发现，ping的结果通过通道正确传过来了，（由于我在服务端没有额外配置，所以是无法访问外网）。

观察server端的信息：

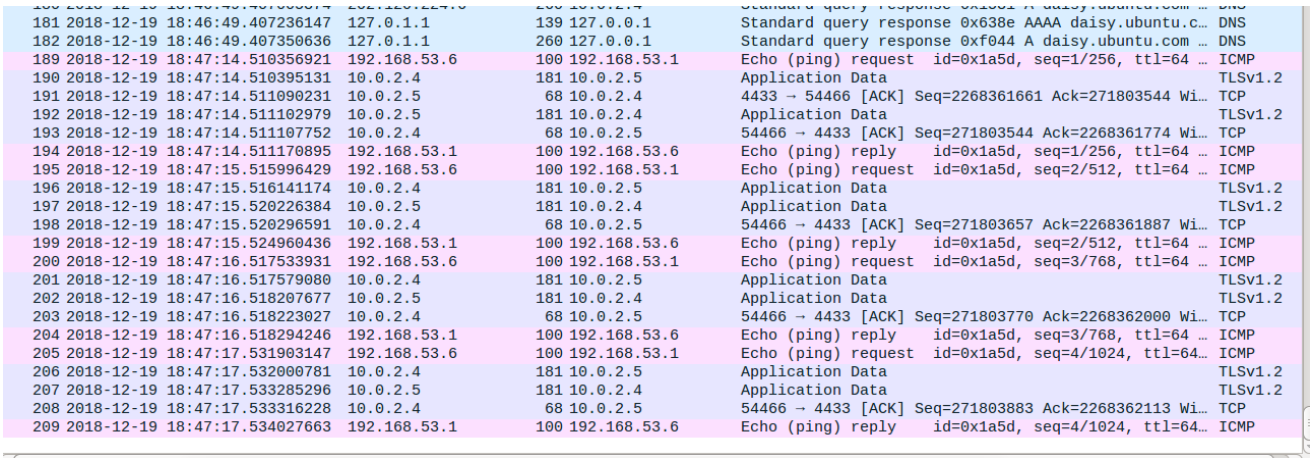


能够正确区分不同的routeIp.

接着断掉192.168.53.5的连接，观察运行状态，发现：

```
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
^C
[12/19/18]seed@VM:~/.../code$ ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
64 bytes from 192.168.53.1: icmp_seq=1 ttl=64 time=0.824 ms
64 bytes from 192.168.53.1: icmp_seq=2 ttl=64 time=9.01 ms
64 bytes from 192.168.53.1: icmp_seq=3 ttl=64 time=0.777 ms
64 bytes from 192.168.53.1: icmp_seq=4 ttl=64 time=2.15 ms
^C
--- 192.168.53.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3021ms
rtt min/avg/max/mdev = 0.777/3.191/9.010/3.404 ms
[12/19/18]seed@VM:~/.../code$
```

发现完全可以正确运行，



181	2018-12-19 18:46:49.407236147	127.0.1.1		139	127.0.0.1	Standard query response 0x638e AAAA daisy.ubuntu.c…	DNS
182	2018-12-19 18:46:49.407350636	127.0.1.1		260	127.0.0.1	Standard query response 0xf044 A daisy.ubuntu.com …	DNS
189	2018-12-19 18:47:14.510356921	192.168.53.6		100	192.168.53.1	Echo (ping) request id=0x1a5d, seq=1/256, ttl=64 …	ICMP
190	2018-12-19 18:47:14.510395131	10.0.2.4		181	10.0.2.5	Application Data	TLSv1.2
191	2018-12-19 18:47:14.511090231	10.0.2.5		68	10.0.2.4	4433 → 54466 [ACK] Seq=2268361661 Ack=271803544 Wi…	TCP
192	2018-12-19 18:47:14.511102979	10.0.2.5		181	10.0.2.4	Application Data	TLSv1.2
193	2018-12-19 18:47:14.511107752	10.0.2.4		68	10.0.2.5	54466 → 4433 [ACK] Seq=271803544 Ack=2268361774 Wi…	TCP
194	2018-12-19 18:47:14.511170895	192.168.53.1		100	192.168.53.6	Echo (ping) reply id=0x1a5d, seq=1/256, ttl=64 …	ICMP
195	2018-12-19 18:47:15.515996429	192.168.53.6		100	192.168.53.1	Echo (ping) request id=0x1a5d, seq=2/512, ttl=64 …	ICMP
196	2018-12-19 18:47:15.516141174	10.0.2.4		181	10.0.2.5	Application Data	TLSv1.2
197	2018-12-19 18:47:15.520226384	10.0.2.5		181	10.0.2.4	Application Data	TLSv1.2
198	2018-12-19 18:47:15.520296591	10.0.2.4		68	10.0.2.5	54466 → 4433 [ACK] Seq=271803657 Ack=2268361887 Wi…	TCP
199	2018-12-19 18:47:15.524960436	192.168.53.1		100	192.168.53.6	Echo (ping) reply id=0x1a5d, seq=2/512, ttl=64 …	ICMP
200	2018-12-19 18:47:16.517533931	192.168.53.6		100	192.168.53.1	Echo (ping) request id=0x1a5d, seq=3/768, ttl=64 …	ICMP
201	2018-12-19 18:47:16.517579080	10.0.2.4		181	10.0.2.5	Application Data	TLSv1.2
202	2018-12-19 18:47:16.518207677	10.0.2.5		181	10.0.2.4	Application Data	TLSv1.2
203	2018-12-19 18:47:16.518223027	10.0.2.4		68	10.0.2.5	54466 → 4433 [ACK] Seq=271803770 Ack=2268362000 Wi…	TCP
204	2018-12-19 18:47:16.518294246	192.168.53.1		100	192.168.53.6	Echo (ping) reply id=0x1a5d, seq=3/768, ttl=64 …	ICMP
205	2018-12-19 18:47:17.531903147	192.168.53.6		100	192.168.53.1	Echo (ping) request id=0x1a5d, seq=4/1024, ttl=64…	ICMP
206	2018-12-19 18:47:17.532000781	10.0.2.4		181	10.0.2.5	Application Data	TLSv1.2
207	2018-12-19 18:47:17.533285296	10.0.2.5		181	10.0.2.4	Application Data	TLSv1.2
208	2018-12-19 18:47:17.533316228	10.0.2.4		68	10.0.2.5	54466 → 4433 [ACK] Seq=271803883 Ack=2268362113 Wi…	TCP
209	2018-12-19 18:47:17.534027663	192.168.53.1		100	192.168.53.6	Echo (ping) reply id=0x1a5d, seq=4/1024, ttl=64…	ICMP

说明此程序在用户的连接与用户的离开，等路由机制完全正确。

说明

客户端

```
$make
$sudo ./vpn_tls_client seed:dees@Hailong.com 192.168.53.5
```

服务端

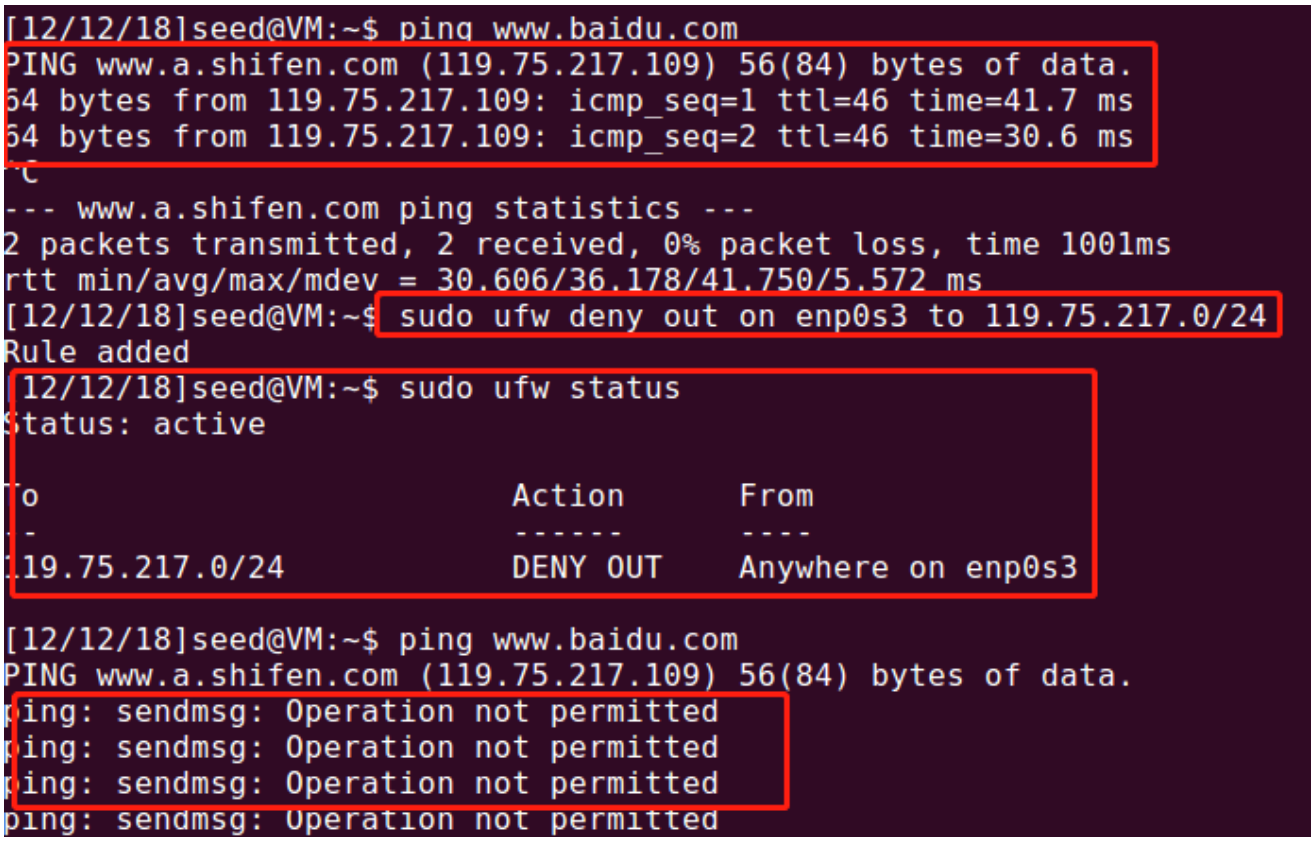
```
$make
$sudo ./vpn_tls_server
$需要输入证书私钥pem的读取密码：1234567890
$sudo ifconfig tun0 192.168.53.1/24 up
```

Firewall Evasion Lab: Bypassing Firewalls using VPN

Task1: VM Setup

前面实验已经完成

Task2: Setup Firewall



使用www.baidu.com测试，如前面实验所做，可以发现当启用防火墙后，将119.75.217.0/24子网拒绝后，可以发现，百度无法访问。

Task3: Bypassing Firewall using VPN

Step 1: Run VPN Server.

前面已经完成

Step 2: Run VPN Client.

前面已经完成

Step3: SetUp Routing on Client and Server VMs

命令：在客户端将所有的流量转发到VPN管道中，或者把被防火墙所阻挡得流量转发到VPN通道中。

```
sudo route add -net default dev tun0
```

在服务端的设置：

```
[12/19/18]seed@VM:~$ sudo route add -net default  dev enp0s3
[12/19/18]seed@VM:~$ sudo iptables -F
[12/19/18]seed@VM:~$  sudo iptables -t nat -F
[12/19/18]seed@VM:~$  sudo iptables -t nat -A POSTROUTING -j MASQUERADE -o enp0s3
[12/19/18]seed@VM:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.1        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        *               255.255.255.0   U     100    0        0 enp0s3
link-local      *               255.255.0.0     U     1000   0        0 enp0s8
192.168.53.0    *               255.255.255.0   U     0      0        0 tun0
192.168.60.0    *               255.255.255.0   U     100    0        0 enp0s8
```

需要注意，把默认第一个网卡enp0s8删掉。因为是内网网卡。

Step 4: Set Up NAT on Server VM.

When the final destination sends packets back to users, the packet will be sent to the VPN Server first (think about why and write down your answer in the report).

因为当客户端得报文到了服务器后，里面的报文源IP地址是服务器局域网中的一个IP地址，当服务器转发后，相当于服务器局域网中的主机发送的报文，所以，报文回复后，直接来的服务器的局域网中，又因为VPN服务器是一个网关，所以它决定了此IP地址转发方向。

telnet连接：

```
[12/19/18]seed@VM:~$ telnet 192.168.60.101
Trying 192.168.60.101...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 16.04.5 LTS
VM login: seed
Password:
Last login: Thu Dec  6 20:15:26 EST 2018 from 192.168.53.5 on pts/0
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

49 packages can be updated.
14 updates are security updates.
```

数据包：

355	2018-12-19 20:21:06.383553734	192.168.60.101	135	192.168.53.6	Telnet Data ...		TELNET
356	2018-12-19 20:21:06.383585191	192.168.53.6	68	192.168.60.101	58596 → 23 [ACK] Seq=1436031705 Ack=4044500681 Win…		TCP
357	2018-12-19 20:21:06.383644183	10.0.2.4	149	10.0.2.5	Application Data		TLSv1.2
358	2018-12-19 20:21:06.386689753	10.0.2.5	151	10.0.2.4	Application Data		TLSv1.2
359	2018-12-19 20:21:06.386798556	192.168.60.101	70	192.168.53.6	Telnet Data ...		TELNET
360	2018-12-19 20:21:06.386818542	192.168.53.6	68	192.168.60.101	58596 → 23 [ACK] Seq=1436031705 Ack=4044500683 Win…		TCP
361	2018-12-19 20:21:06.386851679	10.0.2.4	149	10.0.2.5	Application Data		TLSv1.2
362	2018-12-19 20:21:06.427685352	10.0.2.5	68	10.0.2.4	4433 → 54572 [ACK] Seq=3003254905 Ack=347101868 Wi…		TCP
363	2018-12-19 20:21:06.598792404	10.0.2.5	212	10.0.2.4	Application Data		TLSv1.2
364	2018-12-19 20:21:06.599146426	192.168.60.101	131	192.168.53.6	Telnet Data ...		TELNET
365	2018-12-19 20:21:06.599187751	192.168.53.6	68	192.168.60.101	58596 → 23 [ACK] Seq=1436031705 Ack=4044500746 Win…		TCP
366	2018-12-19 20:21:06.599212292	10.0.2.4	149	10.0.2.5	Application Data		TLSv1.2
367	2018-12-19 20:21:06.599596315	10.0.2.5	68	10.0.2.4	4433 → 54572 [ACK] Seq=3003255049 Ack=347101949 Wi…		TCP
368	2018-12-19 20:21:06.601342401	10.0.2.5	365	10.0.2.4	Application Data		TLSv1.2
369	2018-12-19 20:21:06.601427442	192.168.60.101	284	192.168.53.6	Telnet Data ...		TELNET
370	2018-12-19 20:21:06.601439571	192.168.53.6	68	192.168.60.101	58596 → 23 [ACK] Seq=1436031705 Ack=4044500962 Win…		TCP
371	2018-12-19 20:21:06.601464259	10.0.2.4	149	10.0.2.5	Application Data		TLSv1.2
372	2018-12-19 20:21:06.643520207	10.0.2.5	68	10.0.2.4	4433 → 54572 [ACK] Seq=3003255346 Ack=347102030 Wi…		TCP
373	2018-12-19 20:21:06.910754746	10.0.2.5	170	10.0.2.4	Application Data		TLSv1.2
374	2018-12-19 20:21:06.910895595	192.168.60.101	89	192.168.53.6	Telnet Data ...		TELNET
375	2018-12-19 20:21:06.910907751	192.168.53.6	68	192.168.60.101	58596 → 23 [ACK] Seq=1436031705 Ack=4044500983 Win…		TCP
376	2018-12-19 20:21:06.910932769	10.0.2.4	149	10.0.2.5	Application Data		TLSv1.2

网络访问：

```
[12/19/18]seed@VM:~$ ping 119.75.217.26
PING 119.75.217.26 (119.75.217.26) 56(84) bytes of data.
64 bytes from 119.75.217.26: icmp_seq=1 ttl=44 time=80.5 ms
64 bytes from 119.75.217.26: icmp_seq=2 ttl=44 time=67.7 ms
^C
--- 119.75.217.26 ping statistics ---
3 packets transmitted, 2 received, 33% packet loss, time 2004ms
```

数据包：

11…	2018-12-19 20:28:25.389234191	119.75.217.26	100	192.168.53.6	Echo (ping) reply id=0x22dd, seq=1/256, ttl=44 …		ICMP
11…	2018-12-19 20:28:26.316544731	192.168.53.6	100	119.75.217.26	Echo (ping) request id=0x22dd, seq=2/512, ttl=64 …		ICMP
11…	2018-12-19 20:28:26.316809902	10.0.2.4	181	10.0.2.5	Application Data		TLSv1.2
11…	2018-12-19 20:28:26.317192706	10.0.2.5	68	10.0.2.4	4433 → 54572 [ACK] Seq=3003265802 Ack=347116721 Wi…		TCP
11…	2018-12-19 20:28:26.380558036	10.0.2.5	181	10.0.2.4	Application Data		TLSv1.2
11…	2018-12-19 20:28:26.380589770	10.0.2.4	68	10.0.2.5	54572 → 4433 [ACK] Seq=347116721 Ack=3003265915 Wi…		TCP
11…	2018-12-19 20:28:26.380875454	119.75.217.26	100	192.168.53.6	Echo (ping) reply id=0x22dd, seq=2/512, ttl=44 …		ICMP
11…	2018-12-19 20:28:27.318002056	192.168.53.6	100	119.75.217.26	Echo (ping) request id=0x22dd, seq=3/768, ttl=64 …		ICMP
11…	2018-12-19 20:28:27.318110919	10.0.2.4	181	10.0.2.5	Application Data		TLSv1.2
11…	2018-12-19 20:28:27.318873477	10.0.2.5	68	10.0.2.4	4433 → 54572 [ACK] Seq=3003265915 Ack=347116834 Wi…		TCP
11…	2018-12-19 20:28:27.405368846	10.0.2.5	181	10.0.2.4	Application Data		TLSv1.2
11…	2018-12-19 20:28:27.405397618	10.0.2.4	68	10.0.2.5	54572 → 4433 [ACK] Seq=347116834 Ack=3003266028 Wi…		TCP
11…	2018-12-19 20:28:27.405491958	119.75.217.26	100	192.168.53.6	Echo (ping) reply id=0x22dd, seq=3/768, ttl=44 …		ICMP
11…	2018-12-19 20:28:28.319608731	192.168.53.6	100	119.75.217.26	Echo (ping) request id=0x22dd, seq=4/1024, ttl=64…		ICMP
11…	2018-12-19 20:28:28.319905993	10.0.2.4	181	10.0.2.5	Application Data		TLSv1.2
11…	2018-12-19 20:28:28.320315051	10.0.2.5	68	10.0.2.4	4433 → 54572 [ACK] Seq=3003266028 Ack=347116947 Wi…		TCP
11…	2018-12-19 20:28:28.358469473	10.0.2.5	181	10.0.2.4	Application Data		TLSv1.2
11…	2018-12-19 20:28:28.358497991	10.0.2.4	68	10.0.2.5	54572 → 4433 [ACK] Seq=347116947 Ack=3003266141 Wi…		TCP
11…	2018-12-19 20:28:28.358577505	119.75.217.26	100	192.168.53.6	Echo (ping) reply id=0x22dd, seq=4/1024, ttl=44…		ICMP
11…	2018-12-19 20:28:33.883566381	10.0.2.4	344	10.0.2.3	DHCP Request - Transaction ID 0x387f014c		DHCP