

LAB7: VPN Lab

学号: 57118227 姓名: 孙浩 日期: 2021年7月26日

LAB7: VPN Lab

Container names & IDs

Task 1: Network Setup

Task 2: Create and Configure TUN Interface

Task 2.a: Name of the Interface

Task 2.b: Set up the TUN Interface

Task 2.c: Read from the TUN Interface

Task 2.d: Write to the TUN Interface

Task 3: Send the IP Packet to VPN Server Through a Tunnel

Task 4: Set Up the VPN Server

Task 5: Handling Traffic in Both Directions

Task 6: Tunnel-Breaking Experiment

Container names & IDs

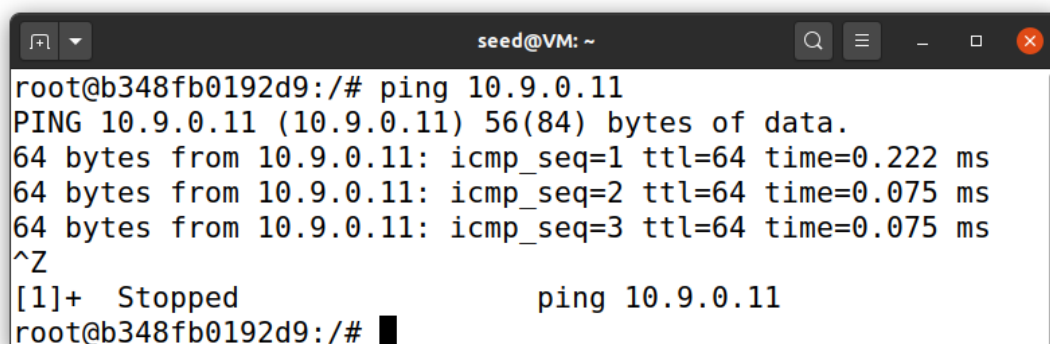
本实验报告在展示结果时用到了较多的命令行终端的窗口截图，为了区分是在哪个Docker容器上运行的结果，在这里对这些容器的名字和ID进行一个说明。

name	ID
Host U-10.9.0.5	b348fb0192d9
Host V-192.168.60.5	c05182b10b67
server-router	3cb0dbc80085
192.168.60.6	88cb2edf43e2

Task 1: Network Setup

Setup步骤同前6次实验相同，接下来进行测试。

- Host *U* can communicate with VPN Server.



```
seed@VM: ~
root@b348fb0192d9:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.222 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.075 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.075 ms
^Z
[1]+  Stopped                  ping 10.9.0.11
root@b348fb0192d9:/#
```

- VPN Server can communicate with Host V

```
seed@VM: ~
root@3cb0dbc80085:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=64 time=0.167 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=0.103 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=0.088 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=64 time=0.117 ms
^Z
[1]+  Stopped                  ping 192.168.60.5
root@3cb0dbc80085:/#
```

- Host U should not be able to communicate with Host V.

```
seed@VM: ~
root@b348fb0192d9:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
30 packets transmitted, 0 received, 100% packet loss, time 29702ms
root@b348fb0192d9:/#
```

- Run *tcpdump* on the router, and sniff the traffic on each of the network. Show that you can capture packets

10.9.0.0子网:

```
seed@VM: ~
root@3cb0dbc80085:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144
bytes
11:49:48.851872 ARP, Request who-has 10.9.0.6 tell 10.9.0.5, length
28
11:49:49.877249 ARP, Request who-has 10.9.0.6 tell 10.9.0.5, length
28
11:49:50.886206 ARP, Request who-has 10.9.0.6 tell 10.9.0.5, length
28
```

192.168.60.0子网:

```
seed@VM: ~  
root@3cb0dbc80085:/# tcpdump -i eth1 -n  
tcpdump: verbose output suppressed, use -v or -vv for full protocol  
decode  
listening on eth1, link-type EN10MB (Ethernet), capture size 262144  
bytes  
11:51:11.654059 ARP, Request who-has 192.168.60.6 tell 192.168.60.5  
, length 28  
11:51:29.129823 ARP, Request who-has 192.168.60.7 tell 192.168.60.5  
, length 28  
11:51:30.150917 ARP, Request who-has 192.168.60.7 tell 192.168.60.5  
, length 28
```

Task 2: Create and Configure TUN Interface

Task 2.a: Name of the Interface

在Host U上运行修改后的tun.py，然后查看网络接口，可以看到一个叫sun0的接口。

```
seed@VM: ~  
root@b348fb0192d9:/# ip address  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
3: sun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500  
    link/none  
8: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default  
    link/ether 02:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0  
    inet 10.9.0.5/24 brd 10.9.0.255 scope global eth0  
        valid_lft forever preferred_lft forever  
root@b348fb0192d9:/#
```

Task 2.b: Set up the TUN Interface

在tun.py末尾增加这样两行代码。

```
os.system("ip addr add 192.168.53.99/24 dev {}".format(iframe))  
os.system("ip link set dev {} up".format(iframe))
```

再次运行，可以看见这个接口已经具备了IP地址，并且不处于DOWN状态了。

```
seed@VM: ~  
root@b348fb0192d9:/# ip address  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
4: sun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 500  
    link/none  
    inet 192.168.53.99/24 scope global sun0  
        valid_lft forever preferred_lft forever  
8: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default  
    link/ether 02:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0  
    inet 10.9.0.5/24 brd 10.9.0.255 scope global eth0  
        valid_lft forever preferred_lft forever  
root@b348fb0192d9:/#
```

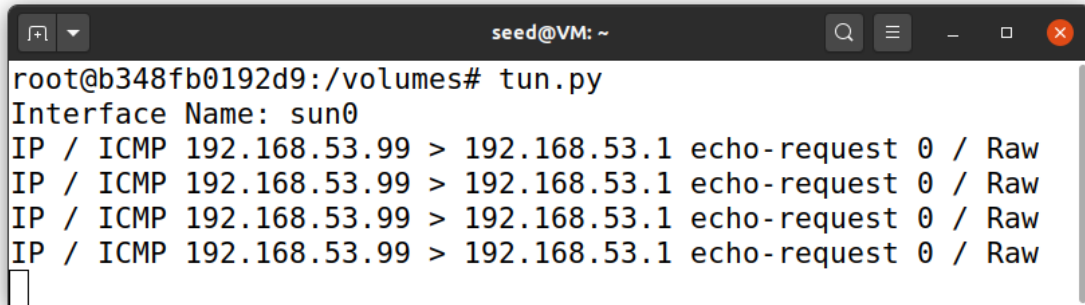
Task 2.c: Read from the TUN Interface

对tun.py原代码最后的while循环部分进行一个修改。

```
while True:
    packet = os.read(tun, 2048)
    if packet:
        ip = IP(packet)
        print(ip.summary())
```

- On Host U, ping a host in the 192.168.53.0/24 network

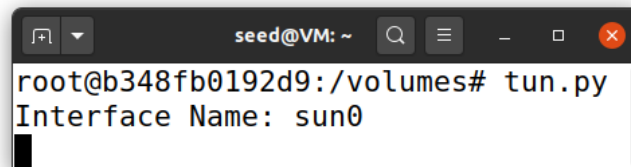
可以听到响应的报文。



```
seed@VM: ~
root@b348fb0192d9:/volumes# tun.py
Interface Name: sun0
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
```

- On Host U, ping a host in the internal network 192.168.60.0/24

不可以听到相应的报文。



```
seed@VM: ~
root@b348fb0192d9:/volumes# tun.py
Interface Name: sun0
```

这是因为相应报文的目的IP不在TUN接口的网段内。

Task 2.d: Write to the TUN Interface

- After getting a packet from the TUN interface, if this packet is an ICMP echo request packet, construct a corresponding echo reply packet and write it to the TUN interface. Please provide evidence to show that the code works as expected.

更改代码的while True部分如下：

```
while True:
    packet = os.read(tun, 2048)
    if packet:
        pkt = IP(packet)
        print(pkt.summary())

        if ICMP in pkt:
            newip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
            newip.ttl = 64
            newicmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
            if pkt.haslayer(Raw):
                data = pkt[Raw].load
                newpkt = newip/newicmp/data
            else:
                newpkt = newip/newicmp
            os.write(tun, bytes(newpkt))
```

可以看到此时PING192.168.53.0/24的IP都可以收到回复。

```
seed@VM: ~  
root@b348fb0192d9:/# ping 192.168.53.77  
PING 192.168.53.77 (192.168.53.77) 56(84) bytes of data.  
64 bytes from 192.168.53.77: icmp_seq=1 ttl=64 time=2.25 ms  
64 bytes from 192.168.53.77: icmp_seq=2 ttl=64 time=2.01 ms  
64 bytes from 192.168.53.77: icmp_seq=3 ttl=64 time=2.89 ms  
64 bytes from 192.168.53.77: icmp_seq=4 ttl=64 time=1.87 ms  
^C  
--- 192.168.53.77 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3025ms  
rtt min/avg/max/mdev = 1.872/2.256/2.889/0.389 ms  
root@b348fb0192d9:/# ping 192.168.53.66  
PING 192.168.53.66 (192.168.53.66) 56(84) bytes of data.  
64 bytes from 192.168.53.66: icmp_seq=1 ttl=64 time=1.71 ms  
64 bytes from 192.168.53.66: icmp_seq=2 ttl=64 time=3.37 ms  
64 bytes from 192.168.53.66: icmp_seq=3 ttl=64 time=2.25 ms  
64 bytes from 192.168.53.66: icmp_seq=4 ttl=64 time=2.06 ms  
^C  
--- 192.168.53.66 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3247ms  
rtt min/avg/max/mdev = 1.713/2.348/3.369/0.619 ms  
root@b348fb0192d9:/#
```

- Instead of writing an IP packet to the interface, write some arbitrary data to the interface, and report your observation

更改while True部分如下。

```
while True:  
    packet = os.read(tun,2048)  
    if packet:  
        pkt = IP(packet)  
        print(pkt.summary())  
        os.write(tun,bytes("Here is SUNHAO"))
```

可以看到因为没有IP包的相应构造方式，会显示无法解码，造成错误而发生中断。

```
seed@VM: ~  
root@b348fb0192d9:/volumes# tun.py  
Interface Name: sun0  
IP / ICMP 192.168.53.99 > 192.168.53.66 echo-request 0 / Raw  
Traceback (most recent call last):  
  File "./tun.py", line 31, in <module>  
    os.write(tun,bytes("Here is SUNHAO"))  
TypeError: string argument without an encoding  
root@b348fb0192d9:/volumes#
```

Task 3: Send the IP Packet to VPN Server Through a Tunnel

tun_server.py的代码如下。

```
#!/usr/bin/env python3  
from scapy.all import *  
  
IP_A = '0.0.0.0'
```

```

PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))

while True:
    data, (ip, port) = sock.recvfrom(2048)
    print("{}: {} --> {}: {}".format(ip, port, IP_A, PORT))
    pkt = IP(data)
    print("    Inside: {}--> {}".format(pkt.src, pkt.dst))

```

tun_client.py的代码的while True部分如下。

```

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

SERVER_IP = '10.9.0.11'
SERVER_PORT = 9090

while True:
    packet = os.read(tun, 2048)
    if packet:
        sock.sendto(packet, (SERVER_IP, SERVER_PORT))

```

在VPN server上运行tun_server.py，在Host U上运行tun_client.py，然后在U上ping 192.168.53.0/24网段的IP，可以看到VPN server上收到了相应的报文。但是ping 192.168.60.0/24网段的IP则没有反应。

```

seed@VM: ~
root@3cb0dbc80085:/volumes# tun_server.py
10.9.0.5:39551 --> 0.0.0.0:9090
    Inside: 192.168.53.99-->192.168.53.77
10.9.0.5:39551 --> 0.0.0.0:9090
    Inside: 192.168.53.99-->192.168.53.77
10.9.0.5:39551 --> 0.0.0.0:9090
    Inside: 192.168.53.99-->192.168.53.77
10.9.0.5:39551 --> 0.0.0.0:9090
    Inside: 192.168.53.99-->192.168.53.77

```

为了能够使60网段的报文通过tunnel，还需要增加一条路由。

```
# ip route add 192.168.60.0/24 dev sun0
```

再次重复上述步骤，可以看见相应的报文在VPN server端接收。

```
seed@VM: ~  
root@3cb0dbc80085:/volumes# tun_server.py  
10.9.0.5:39551 --> 0.0.0.0:9090  
    Inside:192.168.53.99-->192.168.60.8  
10.9.0.5:39551 --> 0.0.0.0:9090  
    Inside:192.168.53.99-->192.168.60.8  
10.9.0.5:39551 --> 0.0.0.0:9090  
    Inside:192.168.53.99-->192.168.60.8  
10.9.0.5:39551 --> 0.0.0.0:9090  
    Inside:192.168.53.99-->192.168.60.8
```

Task 4: Set Up the VPN Server

修改tun_server.py为如下。

```
#!/usr/bin/env python3  
  
import fcntl  
import struct  
import os  
import time  
from scapy.all import *  
  
TUNSETIFF = 0x400454ca  
IFF_TUN   = 0x0001  
IFF_TAP   = 0x0002  
IFF_NO_PI = 0x1000  
  
# Create the tun interface  
tun = os.open("/dev/net/tun", os.O_RDWR)  
ifr = struct.pack('16sH', b'sun%d', IFF_TUN | IFF_NO_PI)  
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)  
  
# Get the interface name  
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")  
print("Interface Name: {}".format(ifname))  
  
os.system("ip addr add 192.168.53.11/24 dev {}".format(ifname))  
os.system("ip link set dev {} up".format(ifname))  
  
IP_A = '0.0.0.0'  
PORT = 9090  
  
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  
sock.bind((IP_A, PORT))  
  
while True:  
    data, (ip, port) = sock.recvfrom(2048)  
    print("{}: {} --> {}: {}".format(ip, port, IP_A, PORT))  
    pkt = IP(data)  
    print("    Inside: {} --> {}".format(pkt.src, pkt.dst))  
    os.write(tun, data)
```


接下来重复上述步骤，这一次我们在Host U上ping Host V，同时在Host V上进行tcpdump，可以看到Host V收到了相应的ICMP request，并发出了reply。

```
seed@VM: ~  
root@c05182b10b67:~# tcpdump -i eth0  
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode  
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes  
13:07:41.902396 IP 192.168.53.99 > c05182b10b67: ICMP echo request, id 177, seq 1, length 64  
13:07:41.902415 IP c05182b10b67 > 192.168.53.99: ICMP echo reply, id 177, seq 1, length 64  
13:07:41.903017 IP c05182b10b67.37965 > 10.80.128.28.domain: 22405+ PTR? 99.53.168.192.in-addr.arpa. (44)  
13:07:42.940790 IP 192.168.53.99 > c05182b10b67: ICMP echo request, id 177, seq 2, length 64  
13:07:42.940809 IP c05182b10b67 > 192.168.53.99: ICMP echo reply, id 177, seq 2, length 64  
13:07:43.968292 IP 192.168.53.99 > c05182b10b67: ICMP echo request, id 177, seq 3, length 64  
13:07:43.968332 IP c05182b10b67 > 192.168.53.99: ICMP echo reply, id 177, seq 3, length 64  
13:07:44.970013 IP 192.168.53.99 > c05182b10b67: ICMP echo request, id 177, seq 4, length 64  
13:07:44.970053 IP c05182b10b67 > 192.168.53.99: ICMP echo reply, id 177, seq 4, length 64  
13:07:46.005993 IP 192.168.53.99 > c05182b10b67: ICMP echo request, id 177, seq 5, length 64  
13:07:46.006018 IP c05182b10b67 > 192.168.53.99: ICMP echo reply, id 177, seq 5, length 64  
13:07:46.907580 IP c05182b10b67.43405 > 10.80.128.28.domain: 22405+ PTR? 99.53.168.192.in-addr.arpa. (44)
```

Task 5: Handling Traffic in Both Directions

最终版本的tun_client.py如下。

```
#!/usr/bin/env python3  
  
import fcntl  
import struct  
import os  
import time  
from scapy.all import *  
  
TUNSETIFF = 0x400454ca  
IFF_TUN   = 0x0001  
IFF_TAP   = 0x0002  
IFF_NO_PI = 0x1000  
  
# Create the tun interface  
tun = os.open("/dev/net/tun", os.O_RDWR)  
ifr = struct.pack('16sH', b'sun%d', IFF_TUN | IFF_NO_PI)  
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)  
  
# Get the interface name  
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")  
print("Interface Name: {}".format(ifname))  
  
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))  
os.system("ip link set dev {} up".format(ifname))  
os.system("ip route add 192.168.60.0/24 dev {}".format(ifname))  
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  
  
SERVER_IP = '10.9.0.11'  
SERVER_PORT = 9090  
fds = [sock, tun]  
while True:  
    ready, _, _ = select.select(fds, [], [])  
    for fd in ready:  
        if fd is sock:  
            data, (ip, port) = sock.recvfrom(2048)  
            pkt = IP(data)  
            print("From socket :{} --> {}".format(pkt.src, pkt.dst))  
            os.write(tun, data)
```



```

        if fd is tun:
            packet = os.read(tun, 2048)
            if packet:
                pkt = IP(packet)
                print(pkt.summary())
                sock.sendto(packet, (SERVER_IP, SERVER_PORT))

```

最终版的tun_server.py如下。

```

#!/usr/bin/env python3

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN    = 0x0001
IFF_TAP    = 0x0002
IFF_NO_PI  = 0x1000

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'sun%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)

# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.53.11/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

IP_A = '0.0.0.0'
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))

fds = [sock, tun]

while True:
    ready, _, _ = select.select(fds, [], [])
    for fd in ready:
        if fd is sock:
            data, (ip, port) = sock.recvfrom(2048)
            print("{}: {} --> {}: {}".format(ip, port, IP_A, PORT))
            pkt = IP(data)
            print("    Inside: {}--> {}".format(pkt.src, pkt.dst))
            os.write(tun, data)
        if fd is tun:
            packet = os.read(tun, 2048)
            pkt = IP(packet)
            print("Return : {} --> {}".format(pkt.src, pkt.dst))
            sock.sendto(packet, (ip, port))

```

在Host U上ping Host V, 可以看到这时已经顺利成功了。

```
seed@VM: ~
root@b348fb0192d9:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=5.57 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=10.7 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=6.12 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=10.6 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=9.23 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=5.98 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=5.19 ms
^C
--- 192.168.60.5 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6014ms
rtt min/avg/max/mdev = 5.191/7.629/10.737/2.271 ms
root@b348fb0192d9:/#
```

通过Wireshark可以看见更为清晰的VPN tunneling过程, 先是10.9.0.5发送给10.9.0.11, 然后VPN变为192.168.53.99发往192.168.60.5, 然后再原路径返回。

1	2021-07-26 09:4...	10.9.0.5	10.9.0.11	UDP	128 42618 → 9090 Len=84	
2	2021-07-26 09:4...	10.9.0.5	10.9.0.11	UDP	128 42618 → 9090 Len=84	
3	2021-07-26 09:4...	192.168.53.99	192.168.60.5	ICMP	100 Echo (ping) request	id=0x00bd, seq=1/256, ttl=63 (no respons...
4	2021-07-26 09:4...	192.168.53.99	192.168.60.5	ICMP	100 Echo (ping) request	id=0x00bd, seq=1/256, ttl=63 (no respons...
5	2021-07-26 09:4...	192.168.53.99	192.168.60.5	ICMP	100 Echo (ping) request	id=0x00bd, seq=1/256, ttl=63 (reply in 6)
6	2021-07-26 09:4...	192.168.60.5	192.168.53.99	ICMP	100 Echo (ping) reply	id=0x00bd, seq=1/256, ttl=64 (request in...
7	2021-07-26 09:4...	192.168.60.5	192.168.53.99	ICMP	100 Echo (ping) reply	id=0x00bd, seq=1/256, ttl=64
8	2021-07-26 09:4...	10.9.0.5	10.9.0.5	UDP	128 9090 → 42618 Len=84	
9	2021-07-26 09:4...	10.9.0.11	10.9.0.5	UDP	128 9090 → 42618 Len=84	

在Host U上telnet Host V, 同样成功了。

```
seed@VM: ~
root@b348fb0192d9:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
c05182b10b67 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@c05182b10b67:~$
```

Wireshark同样可以看见, 走的是和ping相同的路径, 只不过内部变为了TCP协议。

13	2021-07-26 09:4...	10.9.0.5	10.9.0.11	UDP	104 42618 → 9090 Len=60
14	2021-07-26 09:4...	10.9.0.5	10.9.0.11	UDP	104 42618 → 9090 Len=60
15	2021-07-26 09:4...	192.168.53.99	192.168.60.5	TCP	76 58048 → 23 [SYN] Seq=1298747501 Win=64240 Len=0 MSS=1460 SACK...
16	2021-07-26 09:4...	192.168.53.99	192.168.60.5	TCP	76 [TCP Out-Of-Order] 58048 → 23 [SYN] Seq=1298747501 Win=64240 ...
17	2021-07-26 09:4...	192.168.53.99	192.168.60.5	TCP	76 [TCP Out-Of-Order] 58048 → 23 [SYN] Seq=1298747501 Win=64240 ...
18	2021-07-26 09:4...	192.168.60.5	192.168.53.99	TCP	76 23 → 58048 [SYN, ACK] Seq=2099584450 Ack=1298747502 Win=65160 ...
19	2021-07-26 09:4...	192.168.60.5	192.168.53.99	TCP	76 [TCP Out-Of-Order] 23 → 58048 [SYN, ACK] Seq=2099584450 Ack=1...
20	2021-07-26 09:4...	10.9.0.11	10.9.0.5	UDP	104 9090 → 42618 Len=60
21	2021-07-26 09:4...	10.9.0.11	10.9.0.5	UDP	104 9090 → 42618 Len=60
22	2021-07-26 09:4...	10.9.0.5	10.9.0.11	UDP	96 42618 → 9090 Len=52
23	2021-07-26 09:4...	10.9.0.5	10.9.0.11	UDP	96 42618 → 9090 Len=52
24	2021-07-26 09:4...	192.168.53.99	192.168.60.5	TCP	68 58048 → 23 [ACK] Seq=1298747502 Ack=2099584451 Win=64256 Len=...
25	2021-07-26 09:4...	192.168.53.99	192.168.60.5	TCP	68 [TCP Dup ACK 24#1] 58048 → 23 [ACK] Seq=1298747502 Ack=209958...
26	2021-07-26 09:4...	10.9.0.5	10.9.0.11	UDP	120 42618 → 9090 Len=76
27	2021-07-26 09:4...	10.9.0.5	10.9.0.11	UDP	120 42618 → 9090 Len=76
28	2021-07-26 09:4...	192.168.53.99	192.168.60.5	TELNET	92 Telnet Data ...

Task 6: Tunnel-Breaking Experiment

在telnet连接过程中，中断tun_client.py程序，在Host V上键入内容将不会有任何显示。

```
seed@VM: ~
seed@c05182b10b67:~$ whoami
seed
seed@c05182b10b67:~$
```

而当我们重新运行tun_client.py程序，重新建立连接的时候，在中断过程中输入的内容会一次性显示出来。

```
seed@VM: ~
seed@c05182b10b67:~$ whoami
seed
seed@c05182b10b67:~$ wholsswssd
```

原因是因为我们的程序是从tun接口上来接收和发送报文的，当程序终止时，报文会存储在这些接口的buffer上，等待程序运行的时候再来处理。相当于一个生产者-消费者问题，中断程序相当于暂时停止了消费者的工作，只要缓存不溢出，就可以继续正常运行。