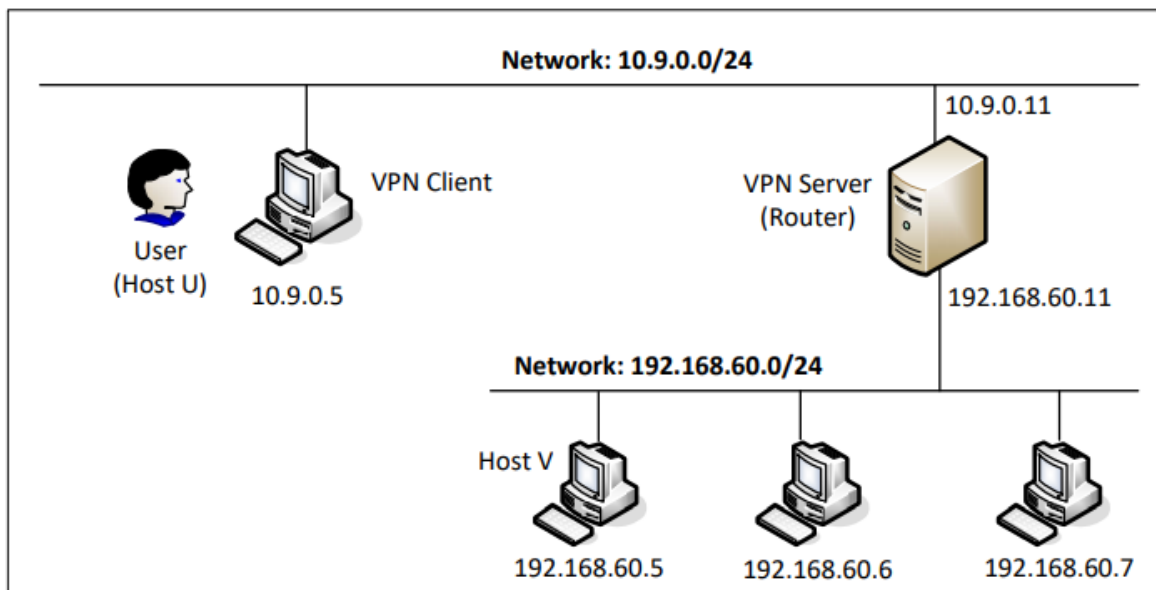


Lab7: VPN Lab

57118228 孙志刚



Task 1: Network Setup

验证主机 U 可以与 VPN Server 通信，Host U ping VPN server 时，VPN server 可从 eth 0 利用 tcpdump 命令抓取数据包。

```
seed@VM: ~/.../Labsetup
[08/04/21]seed@VM:~/.../Labsetup$ docksh 07
root@07625e3e55f5:/# 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=0.099 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.064 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.126 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.073 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.085 ms
64 bytes from 10.9.0.11: icmp_seq=6 ttl=64 time=0.073 ms

seed@VM: ~/.../Labsetup
[08/04/21]seed@VM:~/.../Labsetup$ docksh 60
root@60a8b498c53c:/# 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
22:11:05.964533 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 20, seq 3, length 64
22:11:05.964566 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 20, seq 3, length 64
22:11:06.993201 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 20, seq 4, length 64
22:11:06.993223 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 20, seq 4, length 64
22:11:08.009701 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 20, seq 5, length 64
22:11:08.009726 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 20, seq 5, length 64
22:11:09.002548 ARP, Request who-has 10.9.0.5 tell 10.9.0.11, length 28
22:11:09.002669 ARP, Request who-has 10.9.0.11 tell 10.9.0.5, length 28
22:11:09.002682 ARP, Reply 10.9.0.11 is-at 02:42:0a:09:00:0b, length 28
22:11:09.002684 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
22:11:09.054866 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 20, seq 6, length 64
22:11:09.054889 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 20, seq 6, length 64
22:11:10.117028 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 20, seq 7, length 64
22:11:10.117075 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 20, seq 7, length 64
22:11:11.145621 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 20, seq 8, length 64
```

验证主机 V 192.168.60.5 可以与 VPN Server 通信，同时在VPN服务器上利用tcpdump命令抓取数据包，得到结果如下：

```
seed@VM: ~/.../Labsetup
[08/04/21]seed@VM:~/.../Labsetup$ docksh 60
root@60a8b498c53c:/# 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=0.160 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=0.065 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=0.055 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=64 time=0.073 ms

seed@VM: ~/.../Labsetup
root@60a8b498c53c:/# 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
22:15:54.664586 IP 192.168.60.11 > 192.168.60.5: ICMP echo request, id 21, seq 4, length 64
22:15:54.664626 IP 192.168.60.5 > 192.168.60.11: ICMP echo reply, id 21, seq 4, length 64
22:15:55.688559 IP 192.168.60.11 > 192.168.60.5: ICMP echo request, id 21, seq 5, length 64
22:15:55.688604 IP 192.168.60.5 > 192.168.60.11: ICMP echo reply, id 21, seq 5, length 64
22:15:56.713318 IP 192.168.60.11 > 192.168.60.5: ICMP echo request, id 21, seq 6, length 64
22:15:56.713353 IP 192.168.60.5 > 192.168.60.11: ICMP echo reply, id 21, seq 6, length 64
22:15:56.744480 ARP, Request who-has 192.168.60.11 tell 192.168.60.5, length 28
22:15:56.744497 ARP, Reply 192.168.60.11 is-at 02:42:c0:a8:3c:0b, length 28
22:15:57.736667 IP 192.168.60.11 > 192.168.60.5: ICMP echo request, id 21, seq 7, length 64
22:15:57.736714 IP 192.168.60.5 > 192.168.60.11: ICMP echo reply, id 21, seq 7, length 64
22:15:58.776039 IP 192.168.60.11 > 192.168.60.5: ICMP echo request, id 21, seq 8, length 64
22:15:58.776089 IP 192.168.60.5 > 192.168.60.11: ICMP echo reply, id 21, seq 8, length 64
22:15:59.784590 IP 192.168.60.11 > 192.168.60.5: ICMP echo request, id 21, seq 9, length 64
22:15:59.784653 IP 192.168.60.5 > 192.168.60.11: ICMP echo reply, id 21, seq 9, length 64
22:16:00.808529 IP 192.168.60.11 > 192.168.60.5: ICMP echo request, id 21, seq 10, length 64
22:16:00.808576 IP 192.168.60.5 > 192.168.60.11: ICMP echo reply, id 21, seq 10, length 64
```

验证主机 U 不可与主机 V 通信。

```
seed@VM: ~/.../Labsetup
root@07625e3e55f5:/# 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 ---
7 packets transmitted, 0 received, 100% packet loss, time 6136ms

root@07625e3e55f5:/#
```

Task 2: Create and Configure TUN Interface

Task 2.a: Name of the Interface

在代码此处将 tun 修改成自己名字简拼 szg 。

```
...
ifr = struct.pack('16sH', b'szg%d', IFF_TUN | IFF_NO_PI)
...
```

在主机 U(10.9.0.5) 上运行 `chmod a+x tun.py` 和 `tun.py` 可以观察到修改接口成功。

```
seed@VM: ~/.../Labsetup
[08/04/21]seed@VM:~/.../Labsetup$ docksh 07
root@07625e3e55f5:/# cd volumes
root@07625e3e55f5:/volumes# chmod a+x tun.py
root@07625e3e55f5:/volumes# tun.py
Interface Name: szg0
```

然后在主机 U(10.9.0.5) 上运行 ip address 查看所有接口，可发现我们修改的 tun 接口，命名为 szg0。

```
seed@VM: ~/.../Labsetup
[08/04/21]seed@VM:~/.../Labsetup$ docksh 07
root@07625e3e55f5:/# 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
2: szg0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
102: eth0@if103: <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@07625e3e55f5:/#
```

Task 2.b: Set up the TUN Interface

在 tun.py 文件中添加以下两行代码，给端口自动分配地址：

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

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。

```
root@07625e3e55f5:/# 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: szg0: <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 szg0
        valid_lft forever preferred_lft forever
102: eth0@if103: <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@07625e3e55f5:/#
```

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

```
#!/usr/bin/env python3
import fcntl
import struct
```


在主机 U 上 ping 主机 V，无法连接，这是因为相应报文的目的 IP 不在 TUN 接口的网段内。

```
seed@VM: ~/.../volumes
[08/04/21]seed@VM:~/.../volumes$ docksh 07
root@07625e3e55f5:/# cd volumes
root@07625e3e55f5:/volumes# tun.py
Interface Name: szg0

```

```
seed@VM: ~/.../volumes
root@07625e3e55f5:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```

Task 2.d: Write to the TUN Interface

修改while循环的内容，当host U ping 192.168.53.11网段的地址时，可以收到答复

```
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))
```

```
root@07625e3e55f5:/# ping 192.168.53.11
PING 192.168.53.11 (192.168.53.11) 56(84) bytes of data.
64 bytes from 192.168.53.11: icmp_seq=4 ttl=64 time=2.39 ms
64 bytes from 192.168.53.11: icmp_seq=5 ttl=64 time=2.43 ms
64 bytes from 192.168.53.11: icmp_seq=6 ttl=64 time=4.31 ms
64 bytes from 192.168.53.11: icmp_seq=7 ttl=64 time=2.63 ms
64 bytes from 192.168.53.11: icmp_seq=8 ttl=64 time=3.28 ms
64 bytes from 192.168.53.11: icmp_seq=9 ttl=64 time=2.30 ms
```

```
seed@VM: ~/.../volumes
root@07625e3e55f5:/volumes# tun.py
Interface Name: szg0
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
```

倘若向接口随意写入字符串

```
...  
os.write(tun,bytes("virtualbox1234"))
```

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。

```
root@07625e3e55f5:/volumes# tun.py  
Interface Name: szg0  
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw  
Traceback (most recent call last):  
  File "./tun.py", line 41, in <module>  
    os.write(tun,bytes("virtualbox1234"))  
TypeError: string argument without an encoding  
root@07625e3e55f5:/volumes#
```

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

tun_client.py的代码如下。

```
#!/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'szg%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")  
  
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))  
os.system("ip link set dev {} up".format(ifname))  
  
print("Interface Name: {}".format(ifname))  
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  
  
while True:  
    packet = os.read(tun, 2048)  
    if packet:
```



```
sock.sendto(packet, ("10.9.0.11", 9090))
```

tun_server.py代码如下

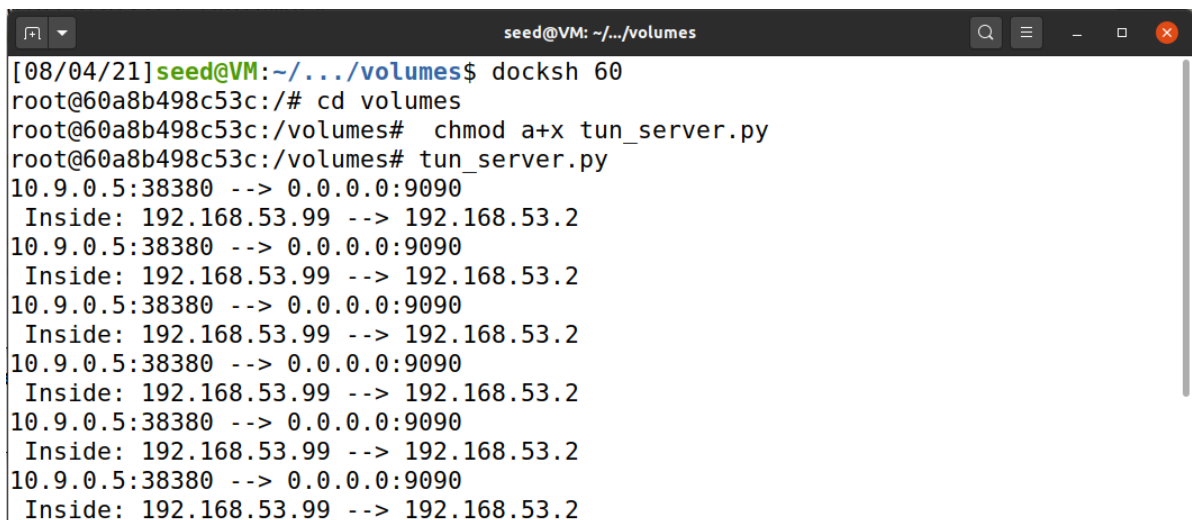
```
#!/usr/bin/env python3
from scapy.all import *

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))
```

在VPN server上运行tun_server.py, 在Host U上运行tun_client.py, 然后在U上ping192.168.53.0/24 网段的IP, 可以看到VPN server上收到了相应的报文。

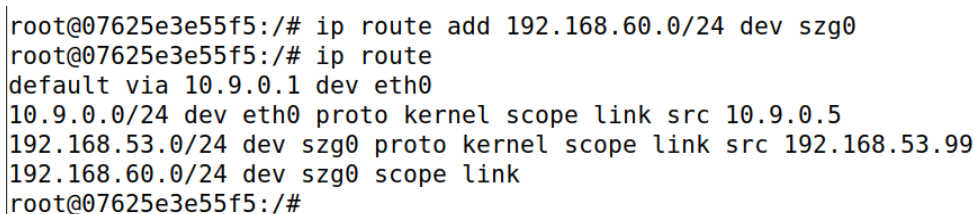


```
seed@VM: ~/../volumes
[08/04/21]seed@VM:~/../volumes$ docksh 60
root@60a8b498c53c:/# cd volumes
root@60a8b498c53c:/volumes# chmod a+x tun_server.py
root@60a8b498c53c:/volumes# tun_server.py
10.9.0.5:38380 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:38380 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:38380 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:38380 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:38380 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:38380 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:38380 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.53.2
```

192.168.53.0/24的路由经过接口 szg0 , 成功通过隧道发送 udp 报文

在主机U上ping主机V, 可知无法连接,因为 192.168.60.0/24 的路由不经过接口 szg0 。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 szg0
```



```
root@07625e3e55f5:/# ip route add 192.168.60.0/24 dev szg0
root@07625e3e55f5:/# ip route
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.5
192.168.53.0/24 dev szg0 proto kernel scope link src 192.168.53.99
192.168.60.0/24 dev szg0 scope link
root@07625e3e55f5:/#
```

重复上述实验过程，在VPN服务器上利用root权限运行tun_server.py程序，得到

```
seed@VM: ~/.../volumes
root@60a8b498c53c:/volumes# tun_server.py
10.9.0.5:40340 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:40340 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:40340 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:40340 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:40340 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:40340 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:40340 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:40340 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
```

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'szg%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")

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

print("Interface Name: {}".format(ifname))
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

IP_A = "0.0.0.0"
PORT = 9090
```



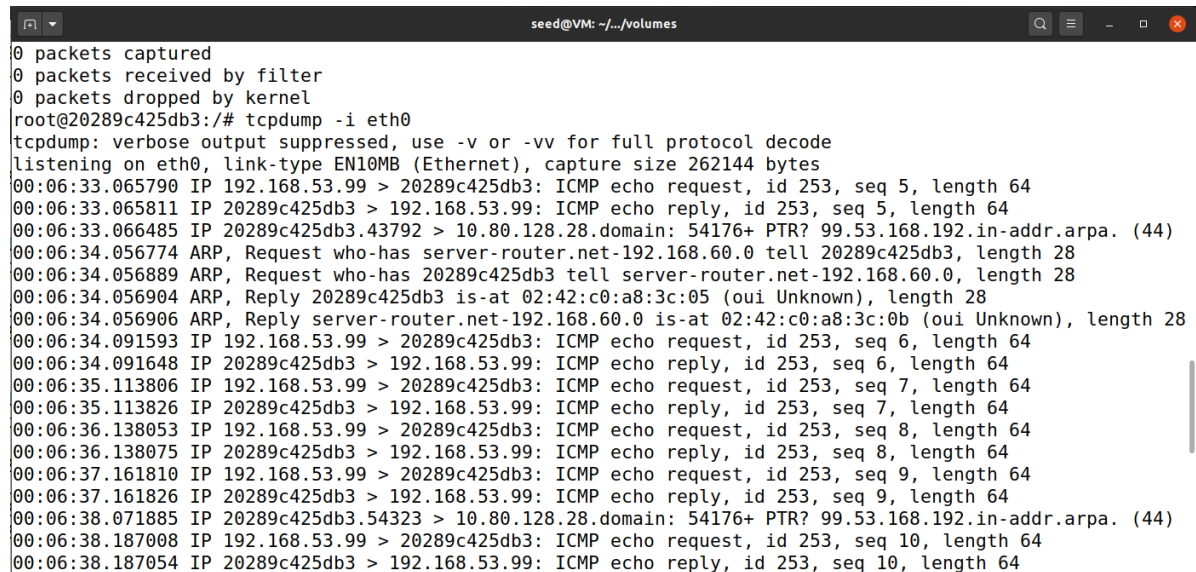
```

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 Task3 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: ~/../volumes
0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@20289c425db3:/# 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
00:06:33.065790 IP 192.168.53.99 > 20289c425db3: ICMP echo request, id 253, seq 5, length 64
00:06:33.065811 IP 20289c425db3 > 192.168.53.99: ICMP echo reply, id 253, seq 5, length 64
00:06:33.066485 IP 20289c425db3.43792 > 10.80.128.28.domain: 54176+ PTR? 99.53.168.192.in-addr.arpa. (44)
00:06:34.056774 ARP, Request who-has server-router.net-192.168.60.0 tell 20289c425db3, length 28
00:06:34.056889 ARP, Request who-has 20289c425db3 tell server-router.net-192.168.60.0, length 28
00:06:34.056904 ARP, Reply 20289c425db3 is-at 02:42:c0:a8:3c:05 (oui Unknown), length 28
00:06:34.056906 ARP, Reply server-router.net-192.168.60.0 is-at 02:42:c0:a8:3c:0b (oui Unknown), length 28
00:06:34.091593 IP 192.168.53.99 > 20289c425db3: ICMP echo request, id 253, seq 6, length 64
00:06:34.091648 IP 20289c425db3 > 192.168.53.99: ICMP echo reply, id 253, seq 6, length 64
00:06:35.113806 IP 192.168.53.99 > 20289c425db3: ICMP echo request, id 253, seq 7, length 64
00:06:35.113826 IP 20289c425db3 > 192.168.53.99: ICMP echo reply, id 253, seq 7, length 64
00:06:36.138053 IP 192.168.53.99 > 20289c425db3: ICMP echo request, id 253, seq 8, length 64
00:06:36.138075 IP 20289c425db3 > 192.168.53.99: ICMP echo reply, id 253, seq 8, length 64
00:06:37.161810 IP 192.168.53.99 > 20289c425db3: ICMP echo request, id 253, seq 9, length 64
00:06:37.161826 IP 20289c425db3 > 192.168.53.99: ICMP echo reply, id 253, seq 9, length 64
00:06:38.071885 IP 20289c425db3.54323 > 10.80.128.28.domain: 54176+ PTR? 99.53.168.192.in-addr.arpa. (44)
00:06:38.187008 IP 192.168.53.99 > 20289c425db3: ICMP echo request, id 253, seq 10, length 64
00:06:38.187054 IP 20289c425db3 > 192.168.53.99: ICMP echo reply, id 253, seq 10, length 64

```

Task 5: Handing Traffic in Both Directions

修改后的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'szg%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)

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, ("10.9.0.11", 9090))

```

修改后的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'szg%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, 可以看到顺利成功了。

```
seed@VM: ~/.../Labsetup
root@07625e3e55f5:/# 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=8 ttl=63 time=7.71 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=4.62 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=5.88 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=2.76 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=3.68 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=4.71 ms
64 bytes from 192.168.60.5: icmp_seq=14 ttl=63 time=4.90 ms
64 bytes from 192.168.60.5: icmp_seq=15 ttl=63 time=2.50 ms
64 bytes from 192.168.60.5: icmp_seq=16 ttl=63 time=2.76 ms
64 bytes from 192.168.60.5: icmp_seq=17 ttl=63 time=3.66 ms

root@07625e3e55f5:/volumes# tun_client.py
Interface Name: szg0
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request len=84
Return : 192.168.60.5 --> 192.168.53.99 echo-reply len=84
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request len=84
Return : 192.168.60.5 --> 192.168.53.99 echo-reply len=84
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request len=84
Return : 192.168.60.5 --> 192.168.53.99 echo-reply len=84
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request len=84
Return : 192.168.60.5 --> 192.168.53.99 echo-reply len=84
From socket :192.168.60.5 --> 192.168.53.99
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request len=84
Return : 192.168.60.5 --> 192.168.53.99 echo-reply len=84
From socket :192.168.60.5 --> 192.168.53.99
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request len=84
Return : 192.168.60.5 --> 192.168.53.99 echo-reply len=84
From socket :192.168.60.5 --> 192.168.53.99
```

通过Wireshrk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。

No.	Time	Source	Destination	Protocol	Length	Info
36	2021-08-04 20:2...	192.168.60.5	192.168.53.99	ICMP	100	Echo (ping) reply id=0x010d, seq=7/1792, ttl=64
37	2021-08-04 20:2...	10.9.0.11	10.9.0.5	UDP	128	9090 -> 52676 Len=84
38	2021-08-04 20:2...	10.9.0.11	10.9.0.5	UDP	128	9090 -> 52676 Len=84
39	2021-08-04 20:2...	10.9.0.5	10.9.0.11	UDP	128	52676 -> 9090 Len=84
40	2021-08-04 20:2...	10.9.0.5	10.9.0.11	UDP	128	52676 -> 9090 Len=84
41	2021-08-04 20:2...	192.168.53.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x010d, seq=8/2048, ttl=63 (no response yet)
42	2021-08-04 20:2...	192.168.53.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x010d, seq=8/2048, ttl=63 (reply in progress)
43	2021-08-04 20:2...	192.168.60.5	192.168.53.99	ICMP	100	Echo (ping) reply id=0x010d, seq=8/2048, ttl=64 (request id=0x010d, seq=7/1792, ttl=64)
44	2021-08-04 20:2...	192.168.60.5	192.168.53.99	ICMP	100	Echo (ping) reply id=0x010d, seq=8/2048, ttl=64
45	2021-08-04 20:2...	10.9.0.11	10.9.0.5	UDP	128	9090 -> 52676 Len=84
46	2021-08-04 20:2...	10.9.0.11	10.9.0.5	UDP	128	9090 -> 52676 Len=84
47	2021-08-04 20:2...	10.9.0.5	10.9.0.11	UDP	128	52676 -> 9090 Len=84
48	2021-08-04 20:2...	10.9.0.5	10.9.0.11	UDP	128	52676 -> 9090 Len=84
49	2021-08-04 20:2...	192.168.53.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x010d, seq=9/2304, ttl=63 (no response yet)

在Host U上 telnet Host V，同样成功了。

```
seed@VM: ~/.../Labsetup
root@07625e3e55f5:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
20289c425db3 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.
Last login: Thu Aug  5 00:38:34 UTC 2021 on pts/2
seed@20289c425db3:~$
```

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

No.	Time	Source	Destination	Protocol	Length	Info
249	2021-08-04 20:3...	10.9.0.11	10.9.0.5	UDP	147	9090 → 39624 Len=103
250	2021-08-04 20:3...	10.9.0.11	10.9.0.5	UDP	147	9090 → 39624 Len=103
251	2021-08-04 20:3...	10.9.0.5	10.9.0.11	UDP	96	39624 → 9090 Len=52
252	2021-08-04 20:3...	10.9.0.5	10.9.0.11	UDP	96	39624 → 9090 Len=52
253	2021-08-04 20:3...	192.168.53.99	192.168.60.5	TCP	68	53452 → 23 [ACK] Seq=1105188571 Ack=2079538139 Win=64128 Len=0
254	2021-08-04 20:3...	192.168.53.99	192.168.60.5	TCP	68	[TCP Dup ACK 253#1] 53452 → 23 [ACK] Seq=1105188571 Ack=2079538139 Win=64128 Len=0
255	2021-08-04 20:3...	192.168.60.5	192.168.53.99	TELNET	89	Telnet Data ...
256	2021-08-04 20:3...	192.168.60.5	192.168.53.99	TCP	89	[TCP Retransmission] 23 → 53452 [PSH, ACK] Seq=2079538139 Ack=1105188571 Win=64128 Len=0
257	2021-08-04 20:3...	10.9.0.11	10.9.0.5	UDP	117	9090 → 39624 Len=73
258	2021-08-04 20:3...	10.9.0.11	10.9.0.5	UDP	117	9090 → 39624 Len=73
259	2021-08-04 20:3...	10.9.0.5	10.9.0.11	UDP	96	39624 → 9090 Len=52
260	2021-08-04 20:3...	10.9.0.5	10.9.0.11	UDP	96	39624 → 9090 Len=52
261	2021-08-04 20:3...	192.168.53.99	192.168.60.5	TCP	68	53452 → 23 [ACK] Seq=1105188571 Ack=2079538160 Win=64128 Len=0
262	2021-08-04 20:3...	192.168.53.99	192.168.60.5	TCP	68	[TCP Dup ACK 261#1] 53452 → 23 [ACK] Seq=1105188571 Ack=2079538160 Win=64128 Len=0

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@20289c425db3:~$ sssss
-bash: sssss: command not found
seed@20289c425db3:~$ whoami
seed
seed@20289c425db3:~$
```

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@20289c425db3:~$ sssss
-bash: sssss: command not found
seed@20289c425db3:~$ whoami
seed
seed@20289c425db3:~$ dfsf sfgsrf
```

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。