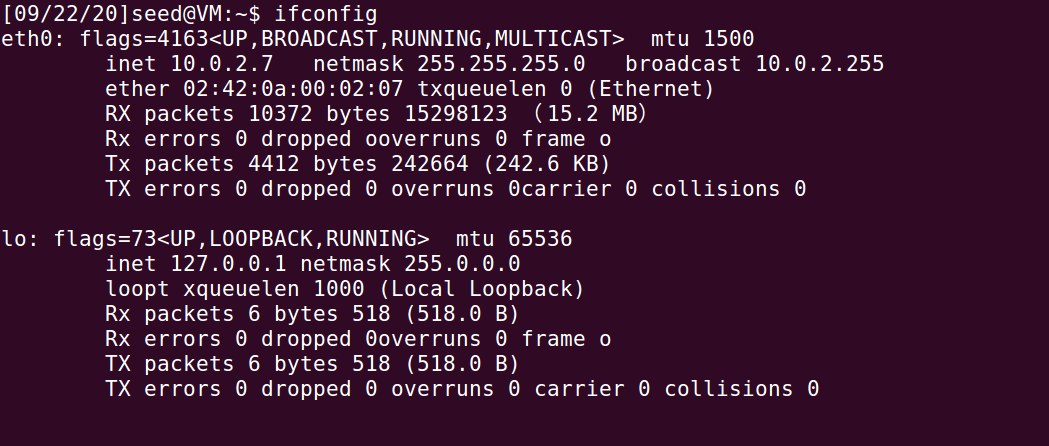
**实验七：VPN Tunneling Lab**

57117222 舒卓卓

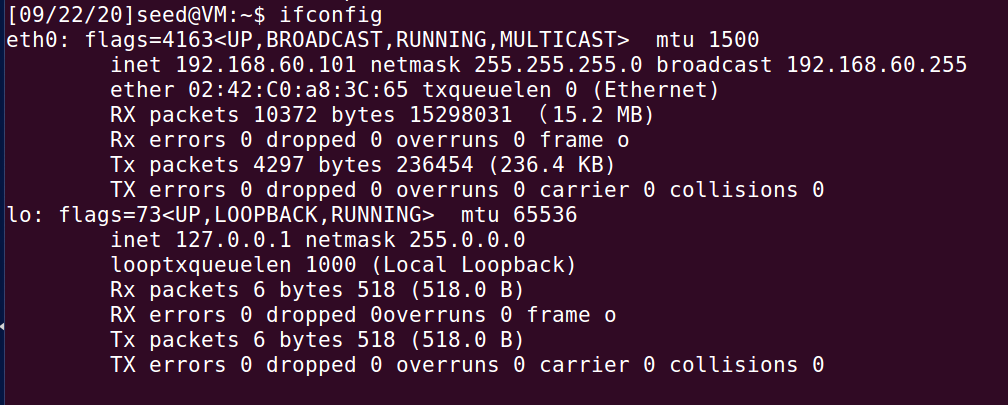
**任务一**：**Network Setup**

1. 新建两个网络，其ip地址和网络名分别为，**10.0.2.0/24 internet** **192.168.60.0/24 internet1**
2. 然后依次创建虚拟机A，B，以及VPN服务器，其中：

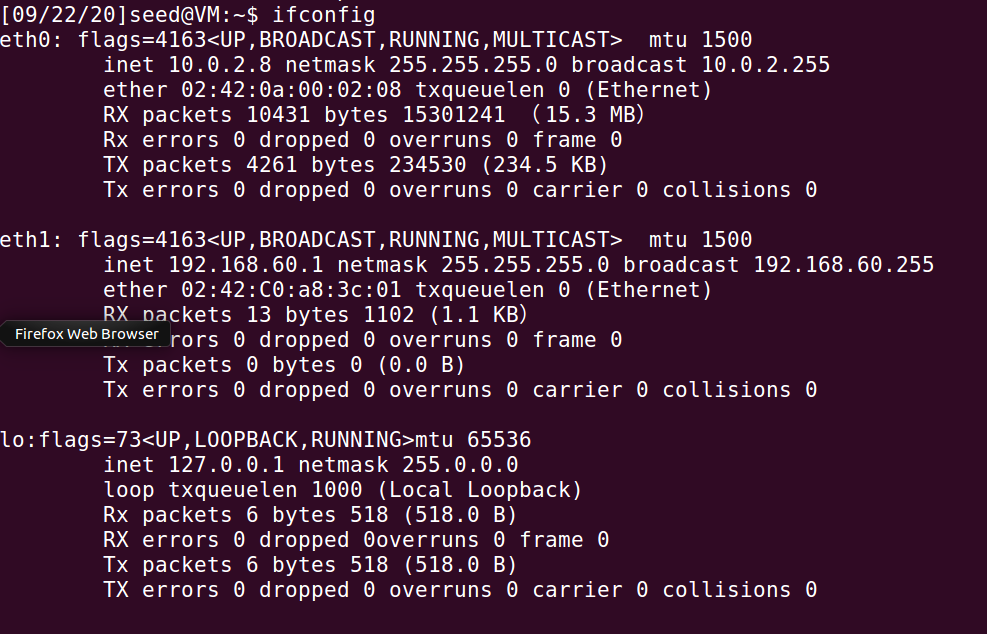
虚拟机A，连接internet，其ip地址为10.0.2.7



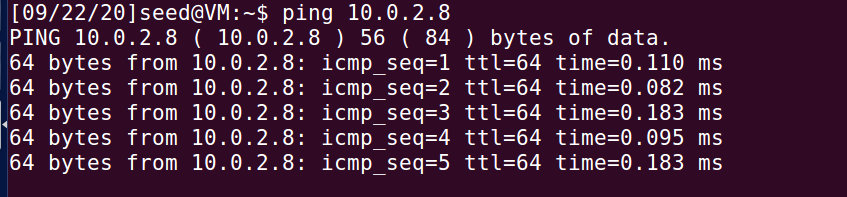
虚拟机B，连接internet1，其ip地址为192.168.60.101



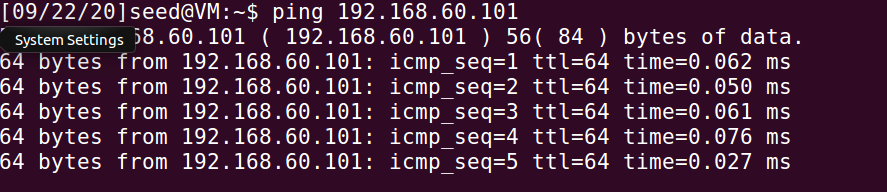
VPN服务器，连接internet，其ip地址为10.0.2.8，连接internet1，ip地址为192.168.60.1



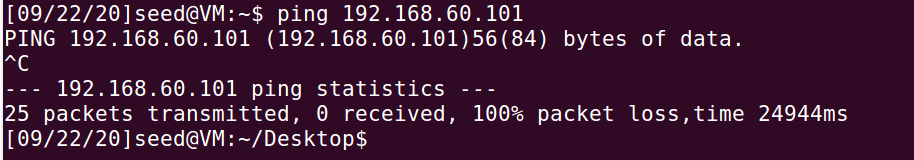
1. 然后我们使用主机A去尝试连接VPN服务器，得到如下结果，可以看到连接成功



1. 接着我们使用VPN服务器连接主机B，得到如下结果，连接成功



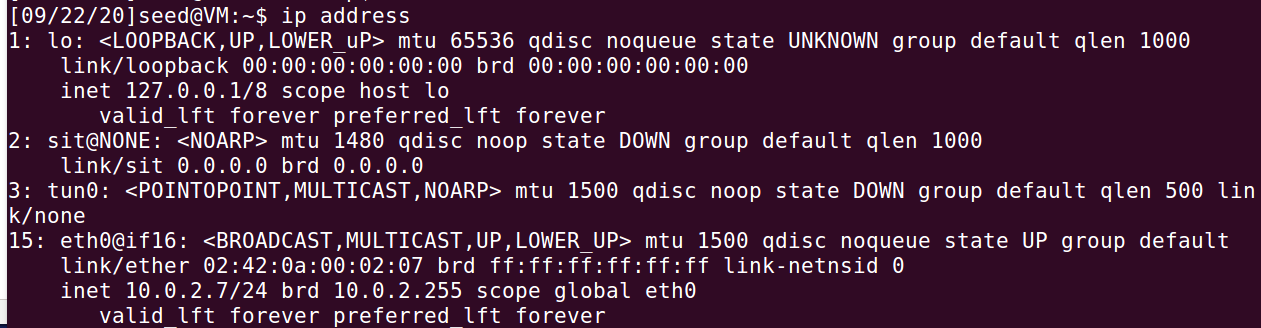
1. 最后我们使用主机A连接主机B，发现无法连接



**任务二：Createand Configure TUN Interface**

**任务二a：Name of the Interface**

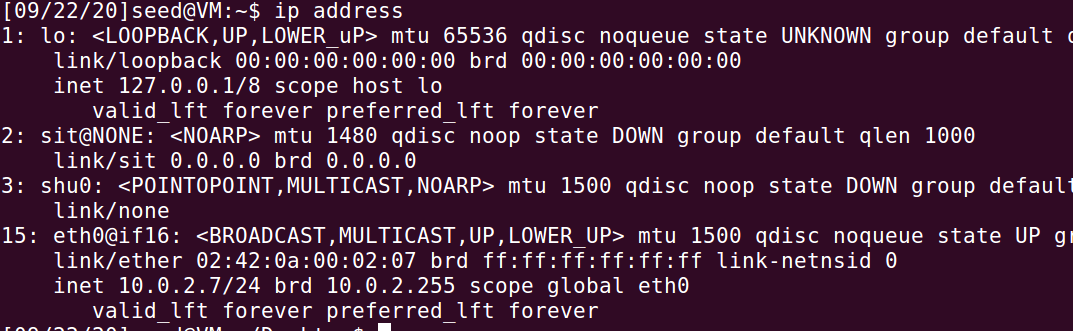
1. 运行代码，然后查看所有的网口信息，如下所示：



1. 我们发现成功注册了网口tun0，我们将代码ifr中对应部分更改为了：

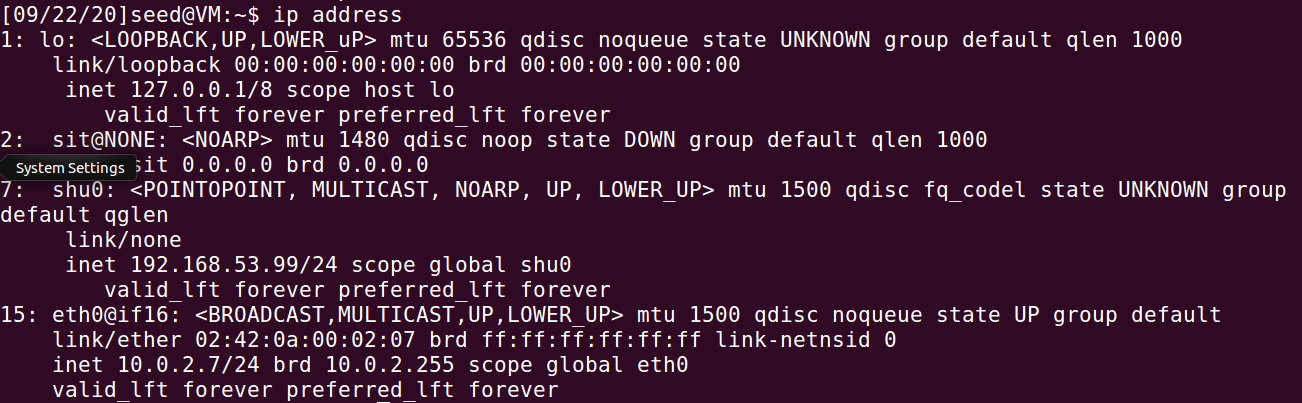
ifr = struct.pack('16sH', b'shu%d', IFF\_TUN | IFF\_NO\_PI)

1. 可以看到1我们成功注册了shu0这个网口



**任务二b：Set up the TUN Interface**

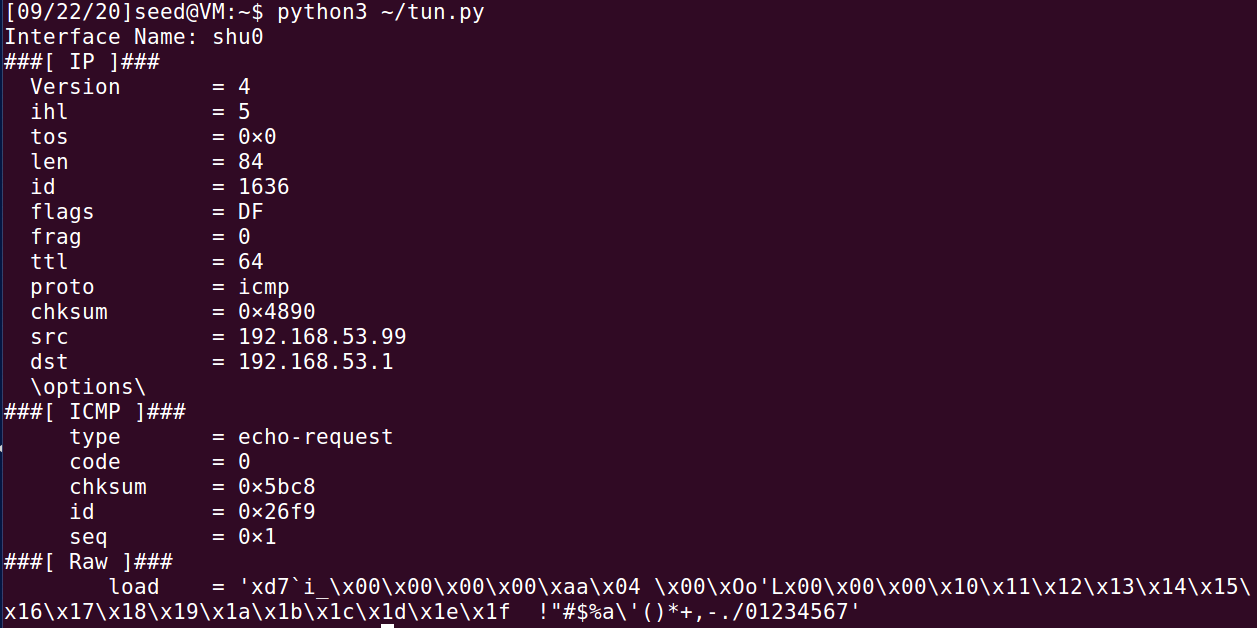
1. 运行题目中给出的代码，我们可以看到：



1. 其中的区别为：shu0的状态FLAG多了UP以及LOWER U，shu0前面的数字从4变成了7，shu0有了inet的ip

**任务二c：Read from the TUN interface**

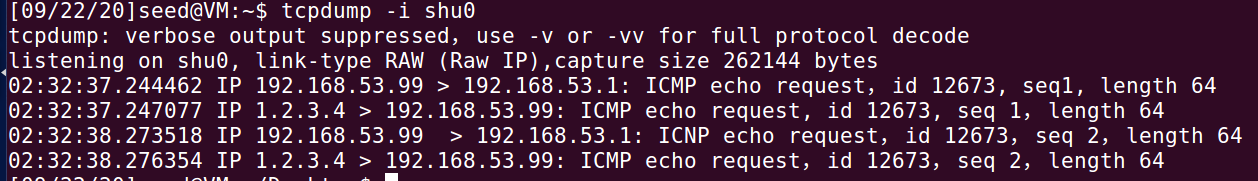
1. 根据题目要求运行了更改之后的代码，在主机A中使用ping 192.168.53.1命令
2. 我们可以得到如图所示的结果：



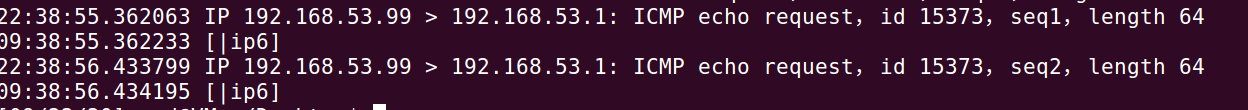
1. 其中确实存在ICMP报文，因为shu0位于当前子网，所以ping就是使用shu0这个网口发出命令
2. 如果我们这时ping另外一个地址 192.168.60.0.我们可以发现不存在相应的输出，因为如果ping找不到子网中的网口就会转而寻找第一个网口，而不是我们所指定的shu0这个网口

**任务二d：Write to the TUN Interface**

1. 修改代码运行之后使用ping命令去ping地址192.168.53.1，接着我们使用命令sudo tcpdump查看当前报文信息

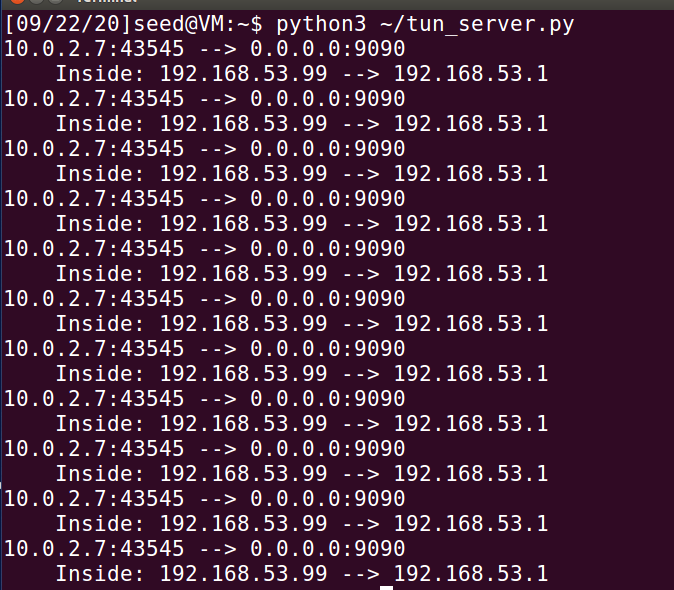


1. 可以发现伪造的报文写入了shu0这个网口
2. 如果这个时候将对应的代码进行更改，修改为了os.write(tun, b'szz')，那么执行同样的命令只会发现其被错误识别



**任务三：Send the IP Packet to VPN Server Through a Tunnel**

1. 更改对应代码运行，在主机A中我们ping地址192.168.53.1
2. 在VPN服务器中可以看到如图所示结果



1. 其原因所在于主机A对192.168.53.1进行ping的时候会使用TUN网口，但是tun\_client把TUN网口收到的报文封装成为了UDP报文，因此VPN服务器可以直接收到，但是如果ping 192.168.60.101，那么VPN不会收到报文
2. 因此在这里为了解决问题，我们添加对应的静态路由：route add -net 192.168.60.0/24 shu0，然后VPN可以收到ICMP请求

**任务四：Set Up the VPN Server**

1. 执行如下所示的代码：

import fcntl

import struct

import os

import time

from scapy.all import \*

TUNSETIFF = 0x400454ca

IFF\_TUN = 0x0001

IFF\_TAP = 0x0002

IFF\_NO\_PI = 0x1000

tun = os.open("/dev/net/tun", os.O\_RDWR)

ifr = struct.pack('16sH', b'tun%d', IFF\_TUN | IFF\_NO\_PI)

ifname\_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)

ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00")

print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.53.98/24 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)

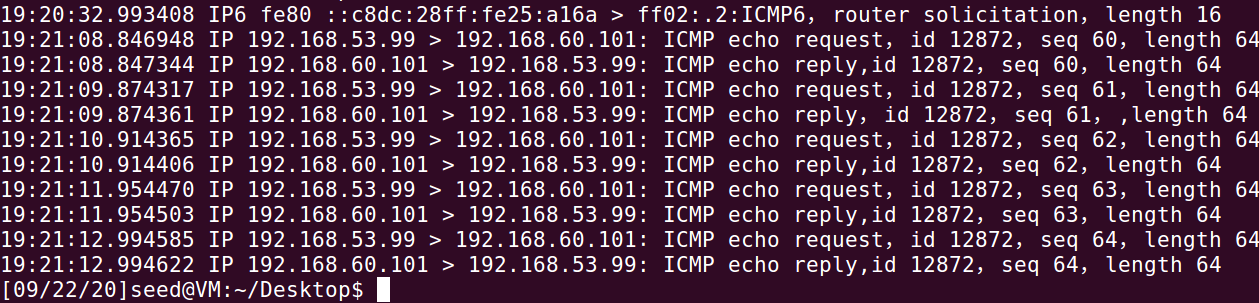
print("{}:{} --> {}:{}".format(ip, port, IP\_A, PORT))

pkt = IP(data)

print(" Inside: {} --> {}".format(pkt.src, pkt.dst)) print("Sending raw: {}".format(data))

os.write(tun, data)

1. 在主机A中ping主机B的地址192.168.60.101，在主机B可以看到对应的结果：



**任务五：Handling Traffic in Both Directions**

1. 修改对应代码如下所示：

**tun\_server.py**

import fcntl

import struct

import os

import time

import select

from scapy.all import \*

TUNSETIFF = 0x400454ca

IFF\_TUN = 0x0001

IFF\_TAP = 0x0002

IFF\_NO\_PI = 0x1000

tun = os.open("/dev/net/tun", os.O\_RDWR)

ifr = struct.pack('16sH', b'tun%d', IFF\_TUN | IFF\_NO\_PI)

ifname\_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)

ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00")

print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.53.98/24 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))

port

while True:

ready, \_, \_ = select.select([sock, tun], [], [])

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)

pkt = IP(packet)

print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))

sock.sendto(packet, ("10.0.2.7", port))

**tun\_client.py**

import os

import time

import select

from scapy.all import \*

TUNSETIFF = 0x400454ca

IFF\_TUN = 0x0001

IFF\_TAP = 0x0002

IFF\_NO\_PI = 0x1000

SERVER\_IP = "10.0.2.8"

SERVER\_PORT = 9090

tun = os.open("/dev/net/tun", os.O\_RDWR)

ifr = struct.pack('16sH', b'zhang%d', IFF\_TUN | IFF\_NO\_PI)

ifname\_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)

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("route add -net 192.168.60.0/24 {}".format(ifname))

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

while True:

ready, \_, \_ = select.select([sock, tun], [], [])

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:

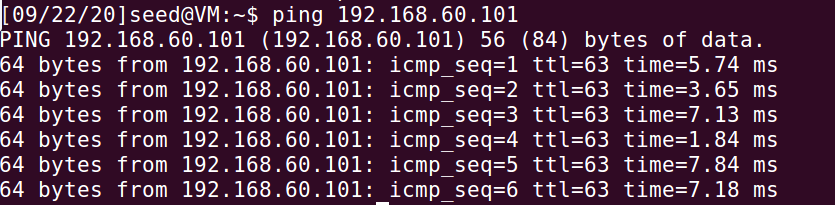
packet = os.read(tun, 2048)

pkt = IP(packet)

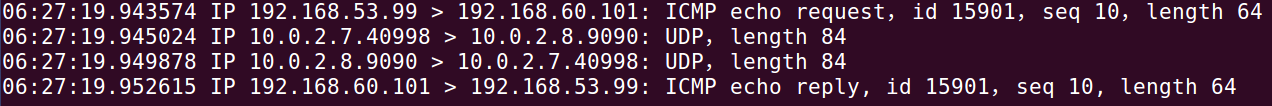
print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))

sock.sendto(packet, (SERVER\_IP, SERVER\_PORT))

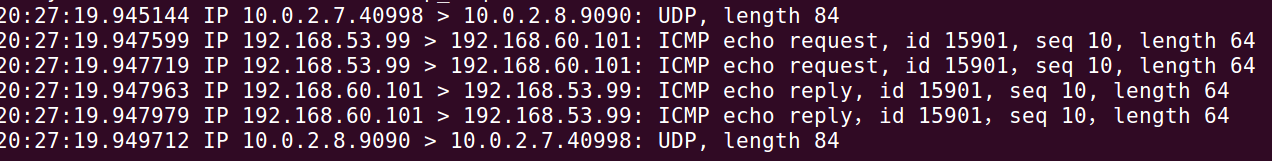
1. 然后在主机B添加路由表项：ip route add 192.168.53.0/24 gw 192.168.60.1 eth0
2. 在主机A ping主机B，发现成功：



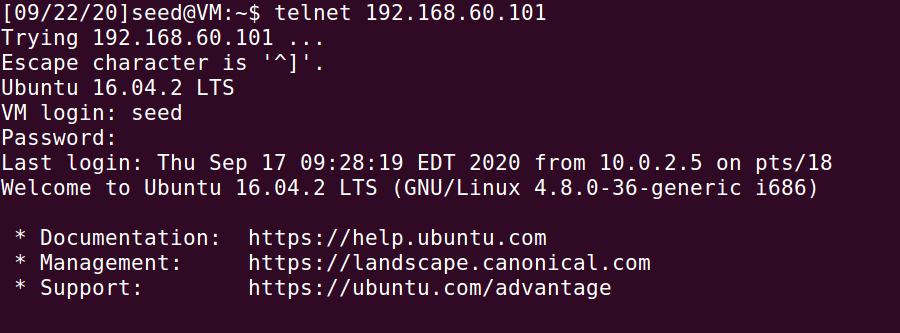
1. 查看主机A上运行tcpdump的结果：



1. 说明由TUN网口向主机V发送ICMP请求， tun\_client.py将这个请求封装成UDP报文后，转发给位于10.0.2.8的VPN服务器。VPN服务器返回给internet端口一个UDP报文，tun\_client.pxy将这个报文解析后，把其中的IP报文写入TUN网口，我们就可以看到最后一行的主机B向TUN网口发送了ICMP 响应。
2. 我们可以看到VPN服务器上面的tcpdump为：



1. 可以看到
2. 综上所述
3. 此外，使用telnet也能成功：



**任务六：Tunnel-Breaking Experiment**

1. 主机A与B建立远程连接之后，关闭掉tun\_client.py，发现无论输入什么，主机A的Telnet界面都没有显示。但此时TCP连接没有断开。再次运行tun\_client.py，输入几个字符，等待一会儿，会发现之前输入的字符重新出现在了Telnet界面中。
2. 这是因为，当关闭tun\_client.py后，之前建立的TCP连接会将内容缓存进缓冲区进入重连状态。如果我们及时恢复tun\_client.py，那么缓冲区的字符又会重新通过TCP连接发送出去

**任务七：Routing Experiment on Host V**

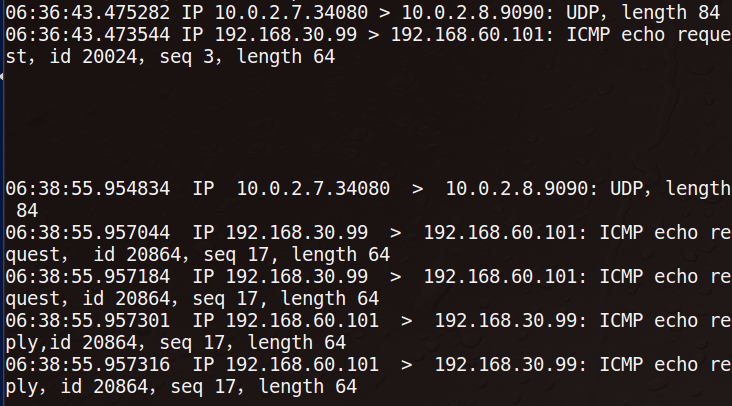
1. 因为再开始的主机A上我们就使用了命令：ip route add 192.168.53.0/24 gw 192.168.60.1 eth0

2. 因此，运行是正常的

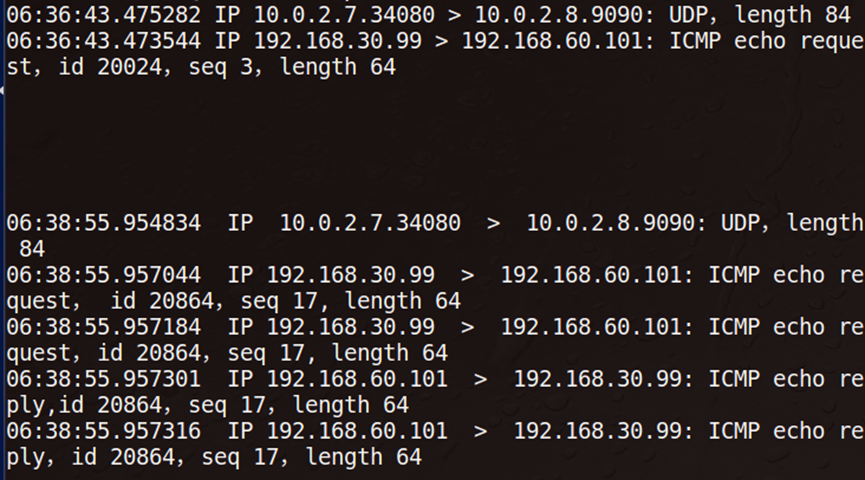
**任务八：Experiment with TUN IP Address**

1. 把主机A上面的tun网口ip地址更改为192.168.30.99后，无法ping通

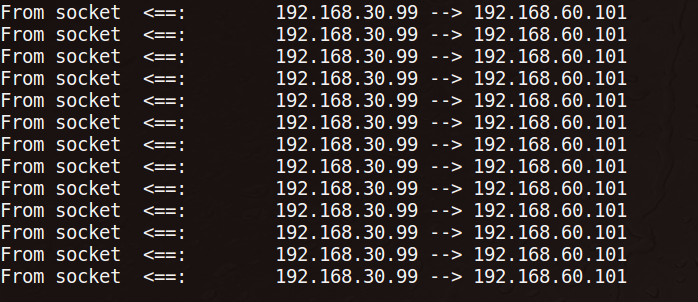
2. 我们再查看A上面的tcpdump的结果：



1. 只是将其封装为了UDP发送出去，但是并未收到返回的udp，然后我们在VPN服务器上面查看tcpdump的情况



1. 收到了UDP报文，将其发送给了B，同时也收到了B的ICMP响应报文，但是没有发出返回UDP报文
2. 接着我们查看tun\_server.py的输出

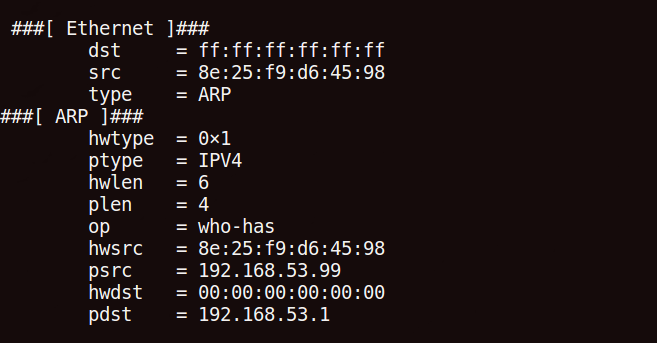


1. 确实没有从TUN口收到包，说明在VPN服务器的返回UDP阶段丢包了。
2. 理由是因为Linux中反向路径过滤功能中,会查看192.1 68.60.101, 192.168.30.99在路由表中是否匹配对应的网口，但这里不匹配，所以包会被丢弃

**任务九：Experiment with the TAP Interface**

1. 编译代码运行，在主机A上面ping192.168.53.1

2. 然后查看Python脚本输出:



1. 发现TAP口确实收到了一个ARP请求。这是因为ping 命令首先会根据192.168.53.1发出一个ARP请求，查看是否在局域网内就有这个IP的设备。而我们的TAP网口的IP 与其在同一子网内，所以命令就转发到了TAP网口中。从而我们就可以查看到，这是一个ARP的who-has请求，查看哪个设备拥有IP 192.168.53.1 。