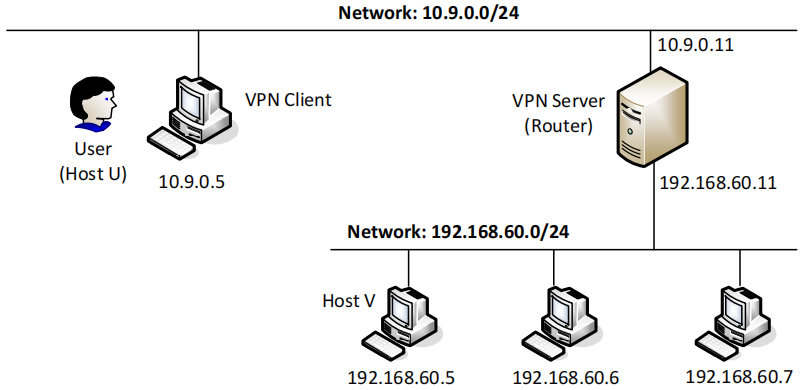
57118130 王嘉麟 lab7

网络结构：



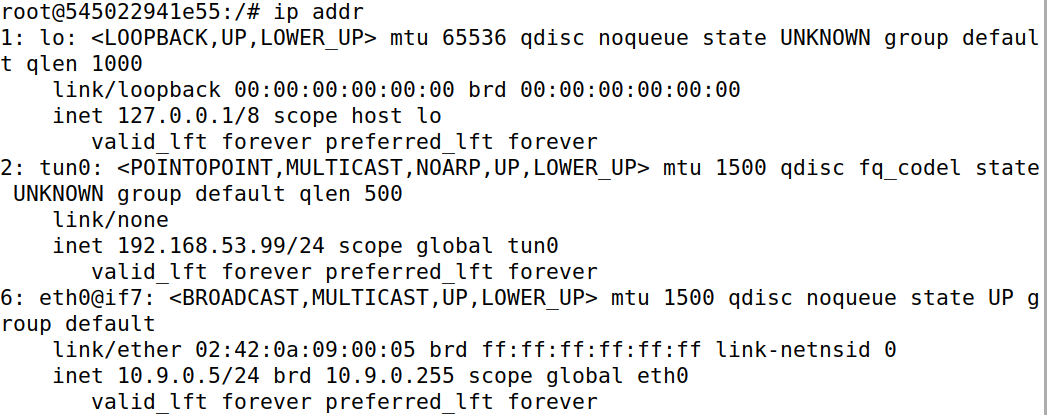
**Task2.a & Task2.b**

在hostU上运行tun.py建立一个tun，在tun.py中添加两行代码，实现自动配置IP地址到tun0的同时，将tun0打开

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

os.system("ip link set dev {} up".format(ifname))

运行程序，然后使用#ip addr查看网络接口如下



**Task2.c**

将原来tun.py中while循环中的time.sleep(10)替换为如下，把从tun接口收到的报文的IP信息输出，然后再次运行tun.py

while True:

packet = os.read(tun, 2048)

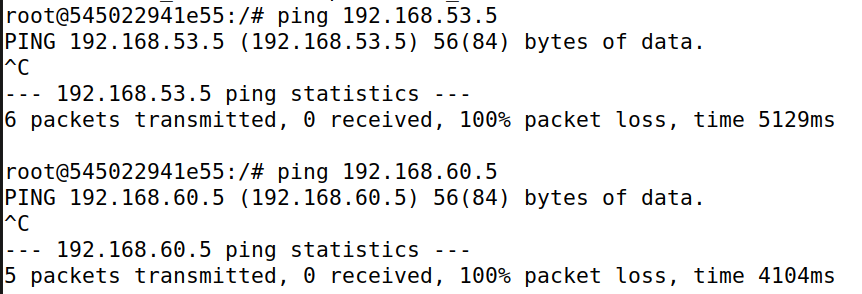
if packet:

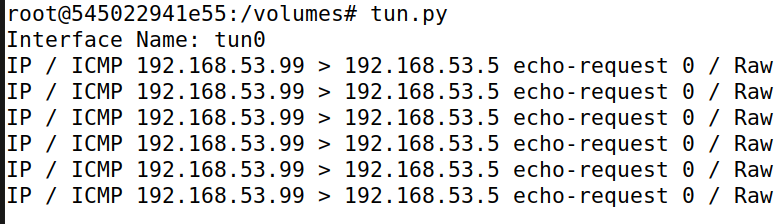
ip = IP(packet)

print(ip.summary())

首先在hostU上ping 192.168.53.5，看到没有收到reply，查看tun.py的输出，看到tun0接口中发出的报文的IP信息，因为发送给192.168.53网段的报文都交给了tun，所以能够看到报文是从tun0的地址放给192.168.53.5传递，但是ruoter和192.168.53.5的tun没有配置，就没有reply

然后在hostU上ping 192.168.60.5，既没有收到reply，在tun.py的输出中也没有看到相关报文，因为发送给192.168.60网段的报文直接给了路由器





**Task2.d**

在while循环中增加如下语句，在读取到tun中的报文的时候构造echo-reply报文

while True:

packet = os.read(tun, 2048)

if packet:

ip = IP(packet)

print(ip.summary())

newip = IP(src=ip.dst, dst=ip.src,ttl = 24)

icmp = ip[ICMP]

if icmp:

if icmp.type == 8 and icmp.code == 0:

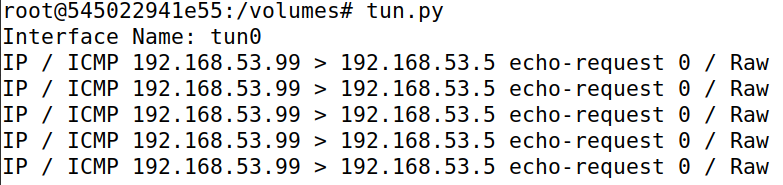
newicmp = ICMP(type=0,id = icmp.id,seq = icmp.seq)

newpkt = newip/newicmp/ip[Raw].load

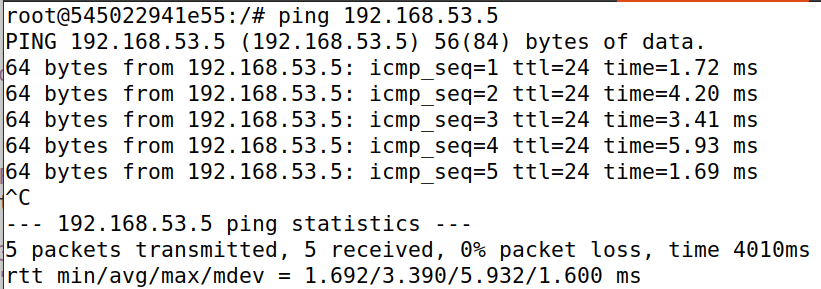
os.write(tun, bytes(newpkt))

os.read读取的报文是byte形式的，将IP头的源地址更改为原ip头的目的地址，目的地址改为原ip头的源地址就行，通过IP(packet)[ICMP]得到ICMP字段，然后构造echo-reply报文

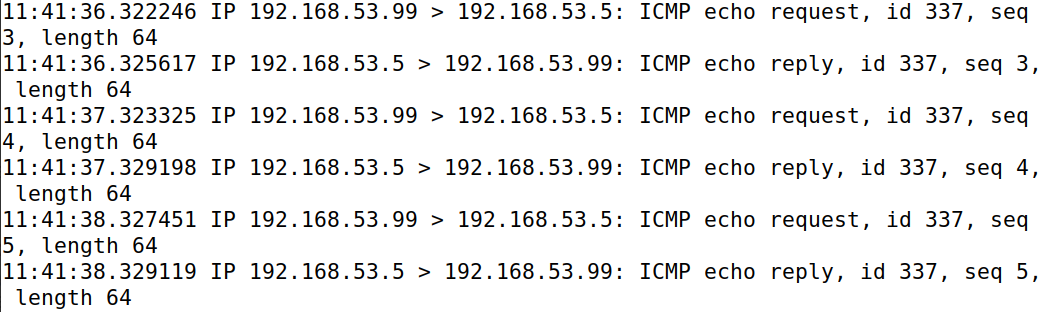
在hostU上执行tun.py输出如下



同样在hostU上，看到tun构造的回复



在hostU上执行#tcpdump -i tun0 -n输出tun0上报文，看到reply报文构造没问题

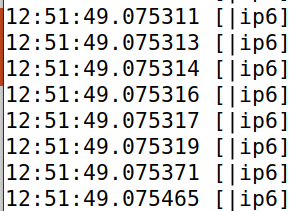


while True:

a = "dfsdfs"

os.write(tun, bytes(a,'ascii'))

用上面两条语句向tun0中写入a字符串，再用tcpdump监听tun0端口，看到大段意义不明的语句



**Task3**

在VPN server上部署tun\_server.py程序，输出所有接收到的udp报文，然后在hostU上部署tun\_client.py程序，将tun中的报文用udp报头封装发送给给10.9.0.11，所以在Task2的基础上改动如下

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

while True:

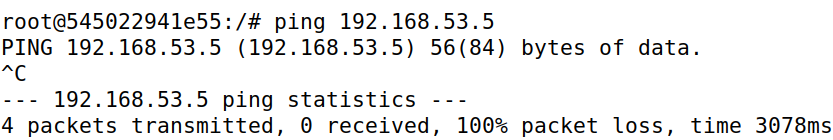
packet = os.read(tun, 2048)

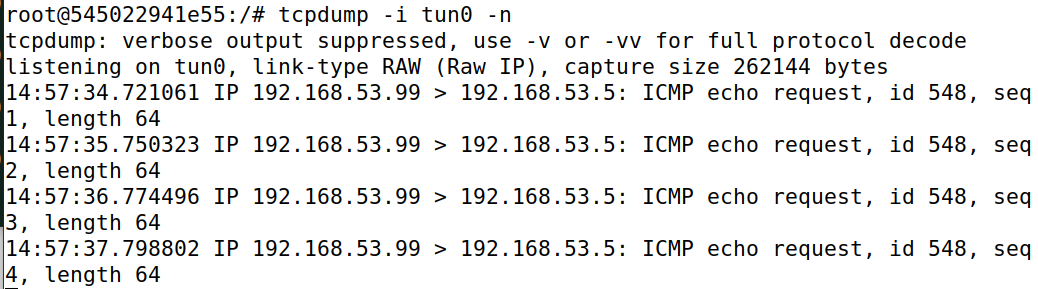
if packet:

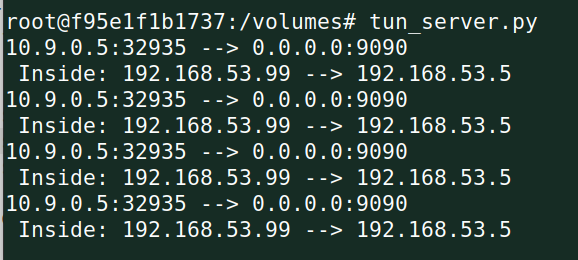
# Send the packet via the tunnel

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

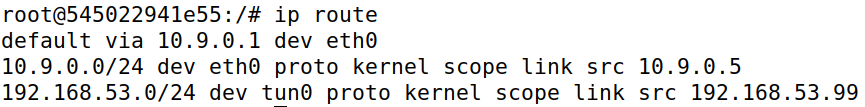
然后在hostU上执行#ping 192.168.53.5测试，虽然没有回显，但是tun\_srever.py收到并输出了报文







通过查看hostU的路由配置#ip route，发现192.168.53网段的报文都会交给192.168.53.99，也就是tun0，所以tun\_client.py收到了icmp echo-request请求，发送给了10.9.0.11



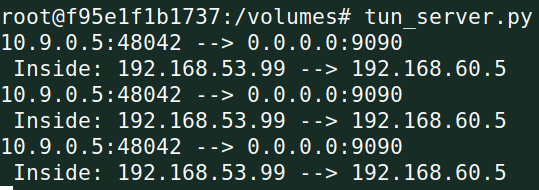
此时在hostU上执行#ping 192.168.60.5则所有窗口都没信息输出，报文没有进入tun，解决方法是在hostU上加上一条路由信息，使192.168.60网段的报文都交给tun0路由

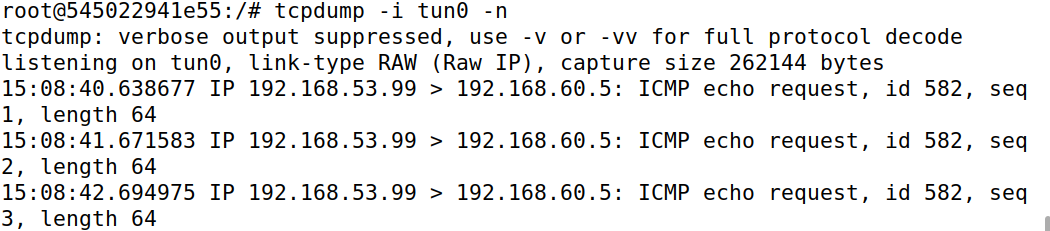
#ip route del 192.168.60.0/24 dev tun0 via 192.168.53.99

将其加入tun\_client.py中，自动配置路由信息

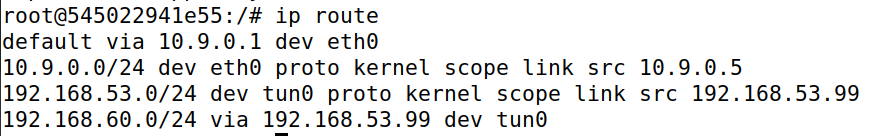
os.system("ip route add 192.168.60.0/24 dev {} via 192.168.53.99".format(ifname))

虽然回显问题没有解决，但是tun\_server.py成功收到报文信息





此时路由信息多了一条



**Task4**

将tun\_server.py改为如下，使其能够创建一个tun，同时把监听到的报文写入到tun中

#!/usr/bin/env python3

import fcntl

import struct

import os

import time

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))

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'tun%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 192.168.53.100/24 dev {}".format(ifname))

os.system("ip link set dev {} up".format(ifname))

while True:

data, (ip, port) = sock.recvfrom(2048)

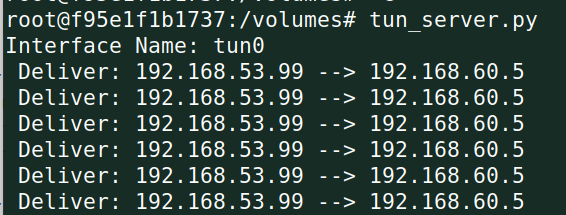
if data:

pkt = IP(data)

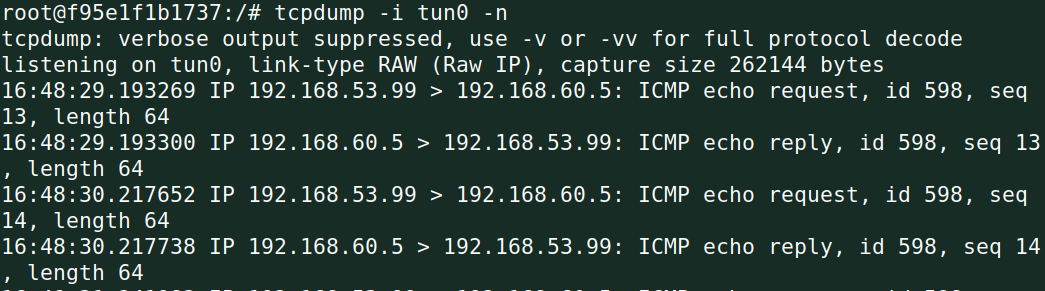
os.write(tun,data)

print(" Deliver: {} --> {}".format(pkt.src, pkt.dst))

在hostU上ping 192.168.60.5后，tun\_server.py的输出，至此单向隧道搭建成功



通过#tcpdump -i tun0 -n查看tun0中的报文，可以看到hostV发送的ICMP reply报文



同时查看hostU上回显，任然没有收到



**Task5**

利用select函数，实现当文件descriptors收到数据的时候，触发类似中断的机制，来配置双向的vpn tunnel

tun\_server2.py的循环

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, ("10.9.0.5", 9090))

tun\_client3.py的循环

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)

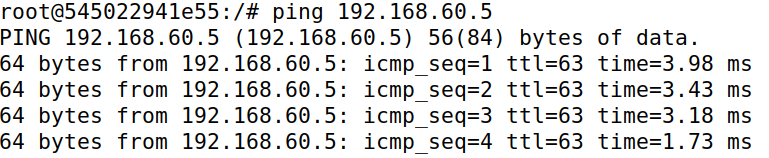
pkt = IP(packet)

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

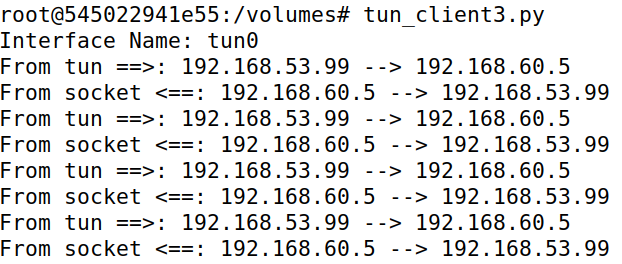
sock.sendto(packet, ("10.9.0.11", 9090))

测试如下

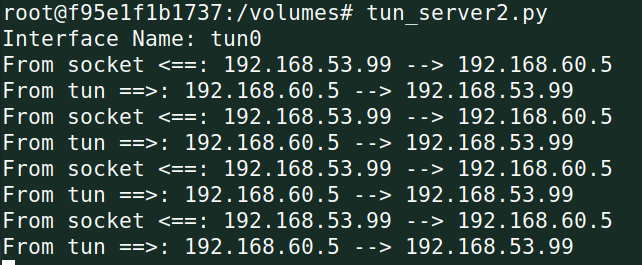
在hostU下执行#ping 192.168.60.5，看到回显



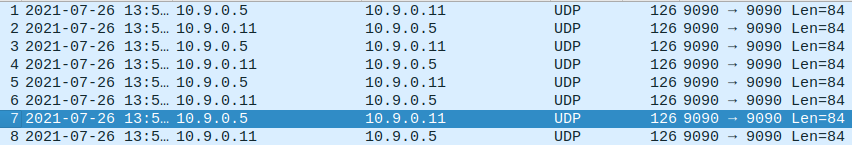
tun\_client3.py的输出，from tun即ICMP echo-request，从本机先将报文发到tun0中，然后读取tun0中的报文向10.9.0.11转发，from socket即ICMP echo-reply，本机监听9090端口，收到来自10.9.0.11转发过来的报文



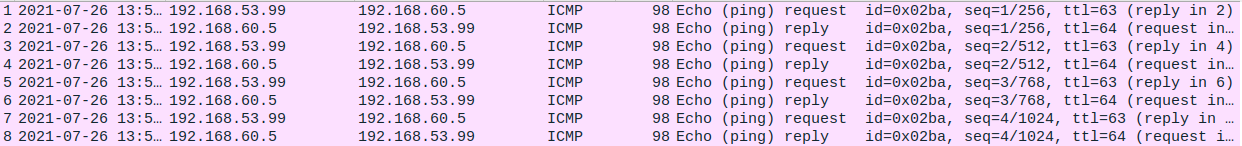
tun\_server2.py的输出，from socket是监听本机9090端口，将从10.9.0.5上来的报文写入到自己的tun中，from tun是读取本机tun0中的报文，echo-reply会发回tun0的地址，然后转发给10.9.0.5



Wireshark录制的10.9.0网段流量，观察到全是udp流量，无法看清目的地，看起来是主机和路由器之间的通信



192.168网段流量，看到就是ICMP报文的正常通信，不知道192.168.53.99主机在哪里

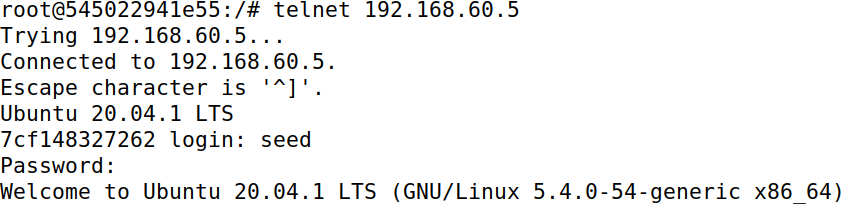


Ping命令中完整的报文流向，

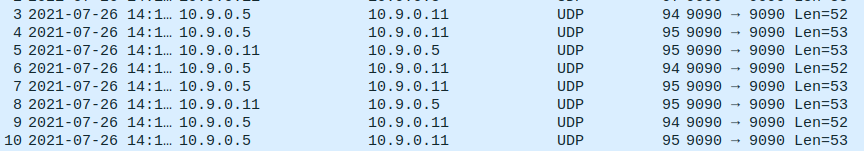
**Echo-request: hostU->hostU tun0->router->router tun0 ->hostV**

**Echo-reply: hostV->router tun0->router->hostU tun0 ->hostU**

在hostU上#telnet 192.168.60.5



此时10.9.0.5上流量



192.168.60.5上流量

