实验六 VPN 设计、实现与分析

邱梓豪

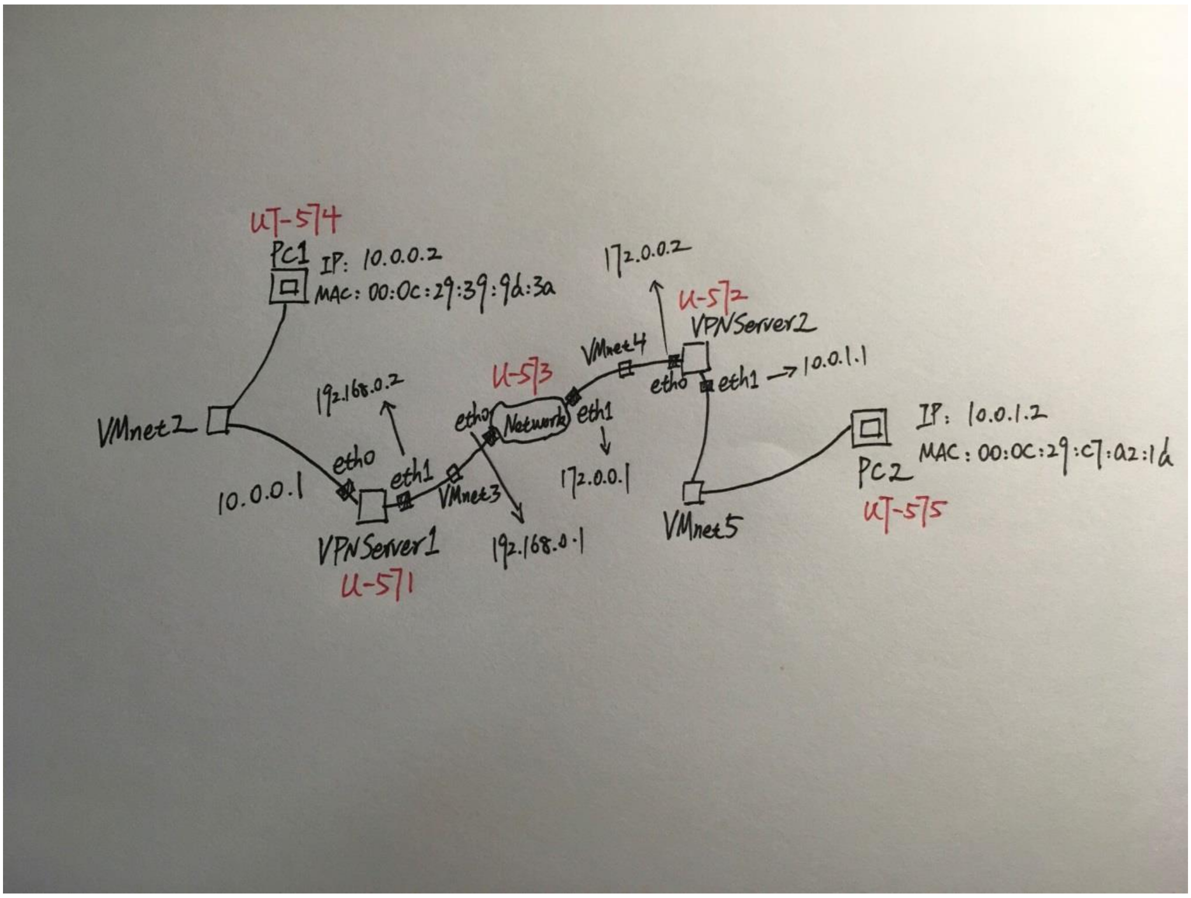
141130077

一、实验目的

本实验主要目的是设计和实现一个简单的虚拟专用网络的机制，并与已有的标准实现（如 PPTP）进行比较，进而进一步了解 VPN 的工作原理和内部实现细节。

二、网络拓扑配置

网络拓扑结构如下：



各节点 IP，掩码，默认网关的设置如实验手册，这里不再赘述。

相关结点的虚拟设备名、IP 及 Netmask 如下

节点名	虚拟设备名	IP	Netmask
VPNServer1	U-571	eth0:10.0.0.1	255.255.255.0
		eth1:192.168.0.2	255.255.255.0
VPNServer2	U-572	eth0:172.0.0.2	255.255.255.0
		eth1:10.0.1.1	255.255.255.0
Network	U-573	eth0:192.168.0.1	255.255.255.0
		eth1:172.0.0.1	255.255.255.0
PC1	UT-574	eth0:10.0.0.2	255.255.255.0
PC2	UT-575	eth0:10.0.1.2	255.255.255.0

三、相关数据结构的定义

(1)静态路由表

```
struct route_item{
    char dest[16];              //  目的 ip
    char gateway[16];            //  网关 ip
    char netmaskl[16];          //  子网掩码
    char interface[16];         //    端口
};
```

(2)ARP 表

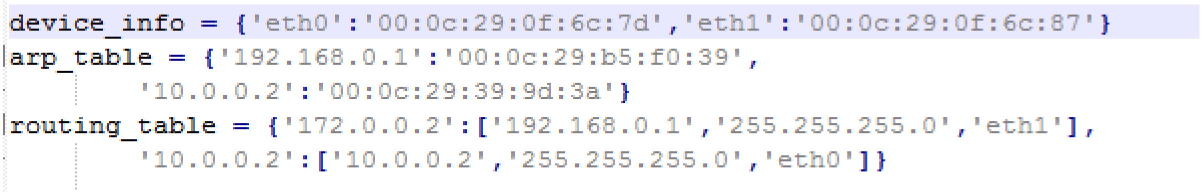
```
struct arp_table{
    char ip_addr[16];           // ip 地址
    char mac_addr[18];          // mac 地址
}
```

(3)路由器信息表

```
struct device_info{
    char interface[14];         //  端口
    char mac_addr[18];          //  端口对应 IP
}
```

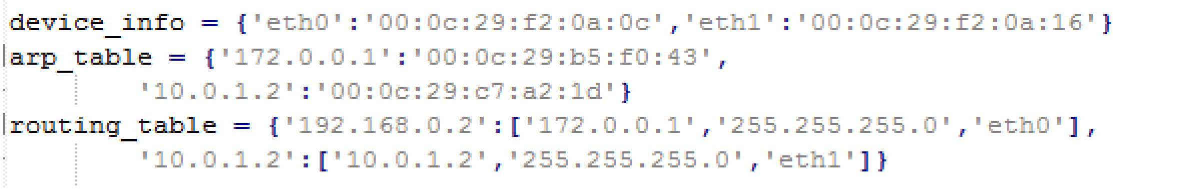
四、VPNServer 上的 vpn 程序中相关表项的设置

（1）VPNServer1 中的设置



```
device_info = {'eth0':'00:0c:29:0f:6c:7d','eth1':'00:0c:29:0f:6c:87'}
arp_table = {'192.168.0.1':'00:0c:29:b5:f0:39',
        '10.0.0.2':'00:0c:29:39:9d:3a'}
routing_table = {'172.0.0.2':['192.168.0.1','255.255.255.0','eth1'],
        '10.0.0.2':['10.0.0.2','255.255.255.0','eth0']}
```

（1）VPNServer2 中的设置



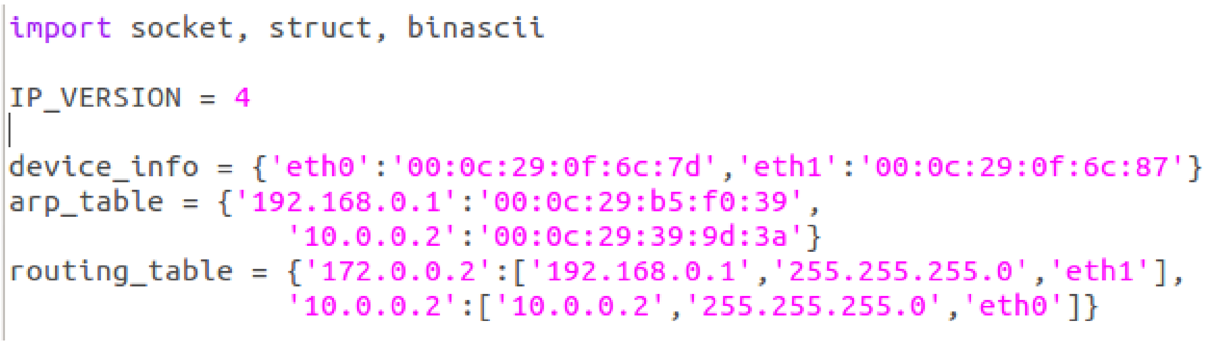
```
device_info = {'eth0':'00:0c:29:f2:0a:0c','eth1':'00:0c:29:f2:0a:16'}
arp_table = {'172.0.0.1':'00:0c:29:b5:f0:43',
        '10.0.1.2':'00:0c:29:c7:a2:1d'}
routing_table = {'192.168.0.2':['172.0.0.1','255.255.255.0','eth0'],
        '10.0.1.2':['10.0.1.2','255.255.255.0','eth1']}
```

五、程序运行说明

在本实验中，python 程序运行在两台 VPNServer 上。两台主机的 ip、mac，两台 VPNServer 的 ip、mac 都被写死在程序中。python 程序的运行使用 python2.x，运行时在终端下输入的命令为：sudo python 程序名

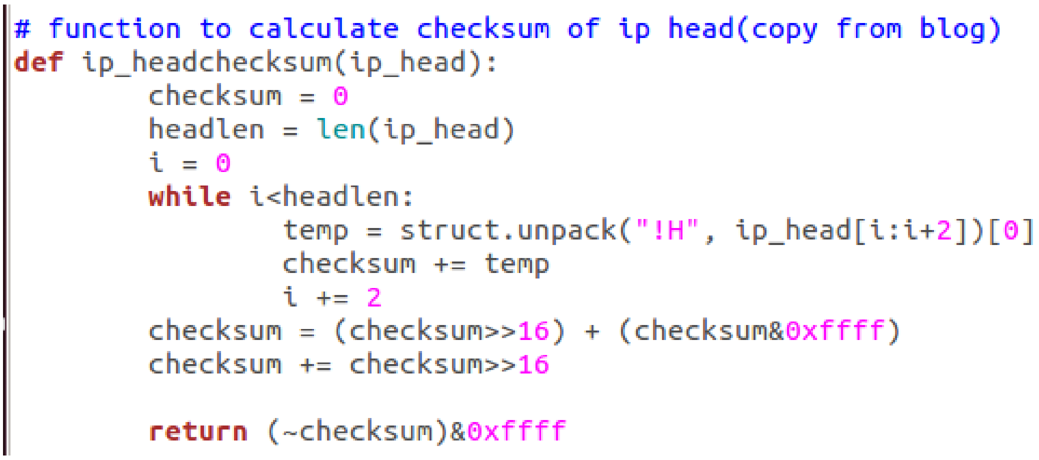
mac 地址在程序中被写在这个位置，如果想运行，必须要根据设备的情况修改该源和目的 mac 地址

六、程序说明：



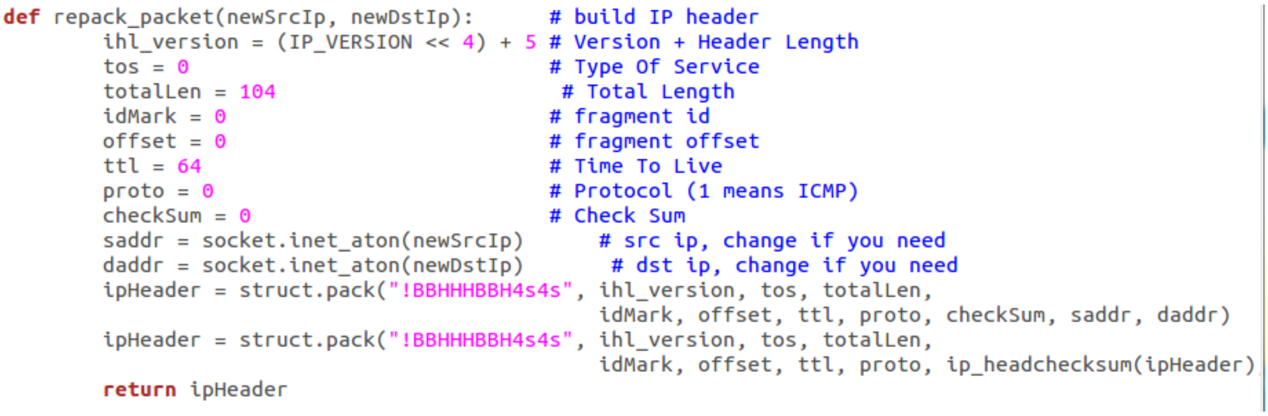
```
import socket, struct, binascii

IP_VERSION = 4

device_info = {'eth0':'00:0c:29:0f:6c:7d','eth1':'00:0c:29:0f:6c:87'}
arp_table = {'192.168.0.1':'00:0c:29:b5:f0:39',
                '10.0.0.2':'00:0c:29:39:9d:3a'}
routing_table = {'172.0.0.2':['192.168.0.1','255.255.255.0','eth1'],
                '10.0.0.2':['10.0.0.2','255.255.255.0','eth0']}
```

首先是导入相关模块。然后定义路由器的 device_info 表，arp 表和路由表，用来对往来的包进行转发。



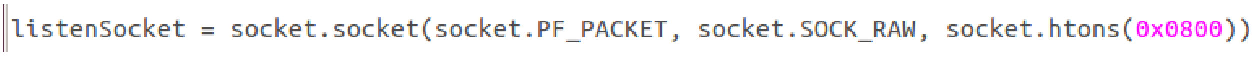
```
# function to calculate checksum of ip head(copy from blog)
def ip_headchecksum(ip_head):
        checksum = 0
        headlen = len(ip_head)
        i = 0
        while i<headlen:
                temp = struct.unpack("!H", ip_head[i:i+2])[0]
                checksum += temp
                i += 2
        checksum = (checksum>>16) + (checksum&0xffff)
        checksum += checksum>>16

        return (~checksum)&0xffff
```

这个函数用来计算 ip 头的校验和，因为使用 vpn 时，原 ip 包要再用一个 ip 头进行封装。



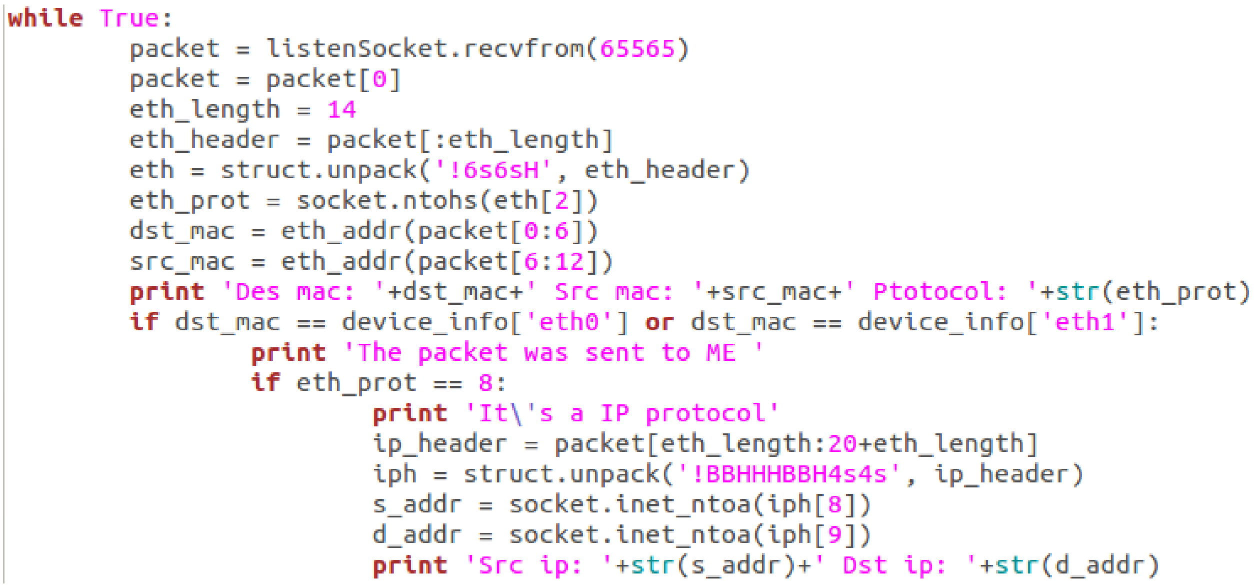
```
def repack_packet(newSrcIp, newDstIp):      # build IP header
        ihl_version = (IP_VERSION << 4) + 5 # Version + Header Length
        tos = 0                             # Type Of Service
        totalLen = 104                       # Total Length
        idMark = 0                          # fragment id
        offset = 0                          # fragment offset
        ttl = 64                            # Time To Live
        proto = 0                           # Protocol (1 means ICMP)
        checkSum = 0                        # Check Sum
        saddr = socket.inet_aton(newSrcIp)      # src ip, change if you need
        daddr = socket.inet_aton(newDstIp)       # dst ip, change if you need
        ipHeader = struct.pack("!BBHHHBBH4s4s", ihl_version, tos, totalLen,
                                        idMark, offset, ttl, proto, checkSum, saddr, daddr)
        ipHeader = struct.pack("!BBHHHBBH4s4s", ihl_version, tos, totalLen,
                                        idMark, offset, ttl, proto, ip_headchecksum(ipHeader)
        return ipHeader
```

这个函数用来构造要加到原 ip 包上的 ip 头，传入的参数 newSrcIp 是发送方 VPNServer 的 ip，newDstIp 是接受方 VPNServer 的 ip，这个 vpn 包就通过这组 ip 来互联网上进行路由。



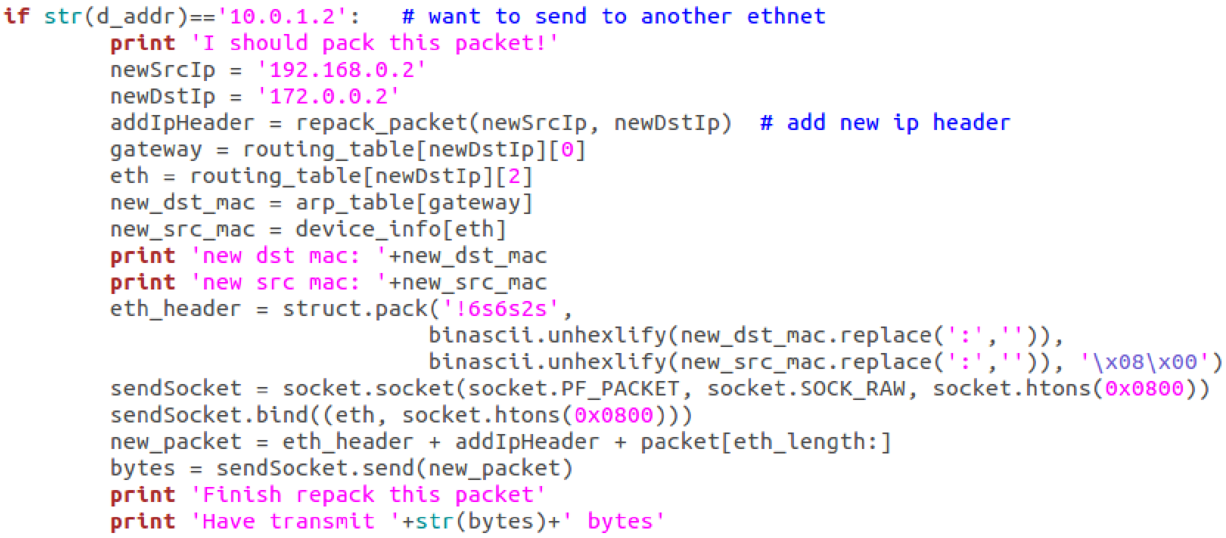
```
listenSocket = socket.socket(socket.PF_PACKET, socket.SOCK_RAW, socket.htons(0x0800))
```

建立一个用于监听端口的 socket。



```
while True:
        packet = listenSocket.recvfrom(65565)
        packet = packet[0]
        eth_length = 14
        eth_header = packet[:eth_length]
        eth = struct.unpack('!6s6sH', eth_header)
        eth_prot = socket.ntohs(eth[2])
        dst_mac = eth_addr(packet[0:6])
        src_mac = eth_addr(packet[6:12])
        print 'Des mac: '+dst_mac+' Src mac: '+src_mac+' Ptotocol: '+str(eth_prot)
        if dst_mac == device_info['eth0'] or dst_mac == device_info['eth1']:
                print 'The packet was sent to ME '
                if eth_prot == 8:
                        print 'It\'s a IP protocol'
                        ip_header = packet[eth_length:20+eth_length]
                        iph = struct.unpack('!BBHHHBBH4s4s', ip_header)
                        s_addr = socket.inet_ntoa(iph[8])
                        d_addr = socket.inet_ntoa(iph[9])
                        print 'Src ip: '+str(s_addr)+' Dst ip: '+str(d_addr)
```

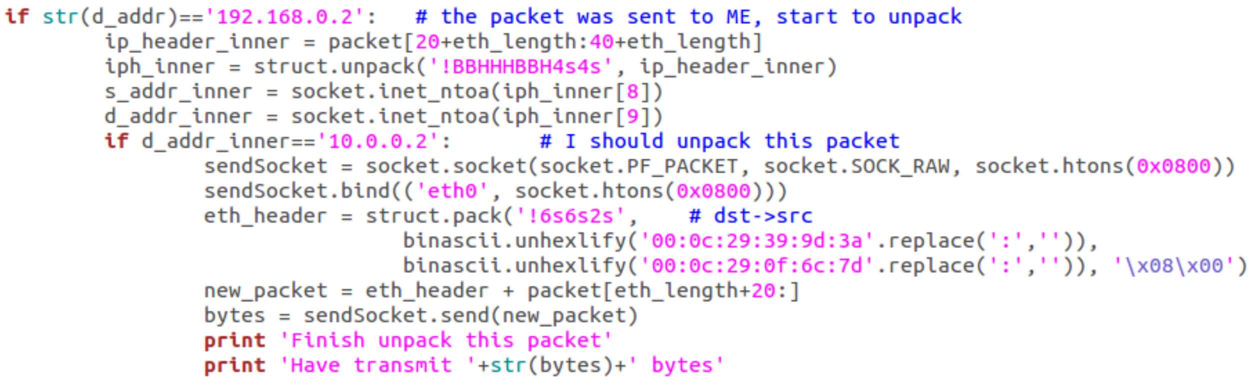
下面开始循环进行抓包分析。每次捕获一个包时，就从中解析出其源 mac 地址和目的 mac 地址及三层协议类型。如果这个包是发给自己的，且上次协议是 ip，就继续进行解析。首先从 ip 头中解析出源 ip 和目的 ip。



```
if str(d_addr)=='10.0.1.2':    # want to send to another ethnet
        print 'I should pack this packet!'
        newSrcIp = '192.168.0.2'
        newDstIp = '172.0.0.2'
        addIpHeader = repack_packet(newSrcIp, newDstIp)  # add new ip header
        gateway = routing_table[newDstIp][0]
        eth = routing_table[newDstIp][2]
        new_dst_mac = arp_table[gateway]
        new_src_mac = device_info[eth]
        print 'new dst mac: '+new_dst_mac
        print 'new src mac: '+new_src_mac
        eth_header = struct.pack('!6s6s2s',
                                binascii.unhexlify(new_dst_mac.replace(':','')),
                                binascii.unhexlify(new_src_mac.replace(':','')), '\x08\x00')
        sendSocket = socket.socket(socket.PF_PACKET, socket.SOCK_RAW, socket.htons(0x0800))
        sendSocket.bind((eth, socket.htons(0x0800)))
        new_packet = eth_header + addIpHeader + packet[eth_length:]
        bytes = sendSocket.send(new_packet)
        print 'Finish repack this packet'
        print 'Have transmit '+str(bytes)+' bytes'
```

如果目的 ip 是一个属于互联网另一端的子网，那么此时就要新构造一个 ip 头，并用这个 ip 头封装这个 ip 包。为了在互联网上成功进行路由，该 VPNServer 还要修改原 ip 包的 mac 地址。修改方式是：将原 mac 改成自己出端口的 mac，将目的 mac 改成

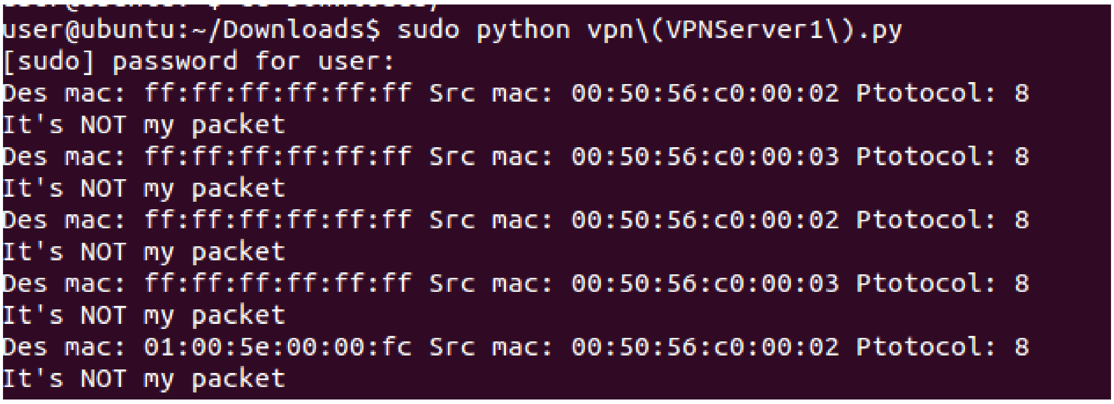
自己默认网关的 mac。之后将新的以太网头加上已被分装的 ip 头一起从端口发送出去。



```
if str(d_addr)=='192.168.0.2':    # the packet was sent to ME, start to unpack
        ip_header_inner = packet[20+eth_length:40+eth_length]
        iph_inner = struct.unpack('!BBHHHBBH4s4s', ip_header_inner)
        s_addr_inner = socket.inet_ntoa(iph_inner[8])
        d_addr_inner = socket.inet_ntoa(iph_inner[9])
        if d_addr_inner=='10.0.0.2':        # I should unpack this packet
                sendSocket = socket.socket(socket.PF_PACKET, socket.SOCK_RAW, socket.htons(0x0800))
                sendSocket.bind(('eth0', socket.htons(0x0800)))
                eth_header = struct.pack('!6s6s2s',     # dst->src
                                binascii.unhexlify('00:0c:29:39:9d:3a'.replace(':','')),
                                binascii.unhexlify('00:0c:29:0f:6c:7d'.replace(':','')), '\x08\x00')
                new_packet = eth_header + packet[eth_length+20:]
                bytes = sendSocket.send(new_packet)
                print 'Finish unpack this packet'
                print 'Have transmit '+str(bytes)+' bytes'
```

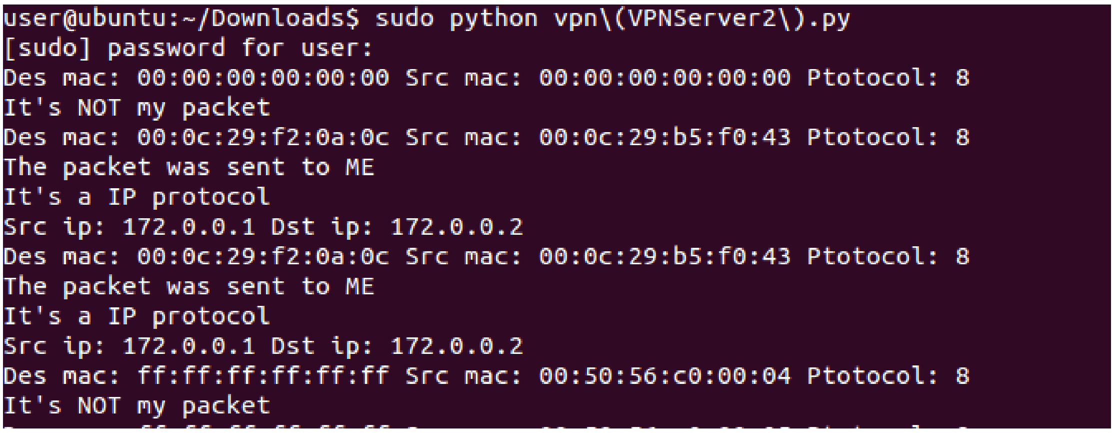
如果目的 ip 是自己，说明这是别人发给自己的包，那么就将这个包的 ip 数据部分进行解析。根据 vpn 的结构可知，ip 数据部分的前 20 个字节是发送方在子网中的原 ip 头，故从中可以得到目的 ip 和源 ip（子网中的），为了能让此包在子网中能顺利到达目的地，还要重新设置 mac 地址。设置完成后，从端口发出即可。

七、运行结果



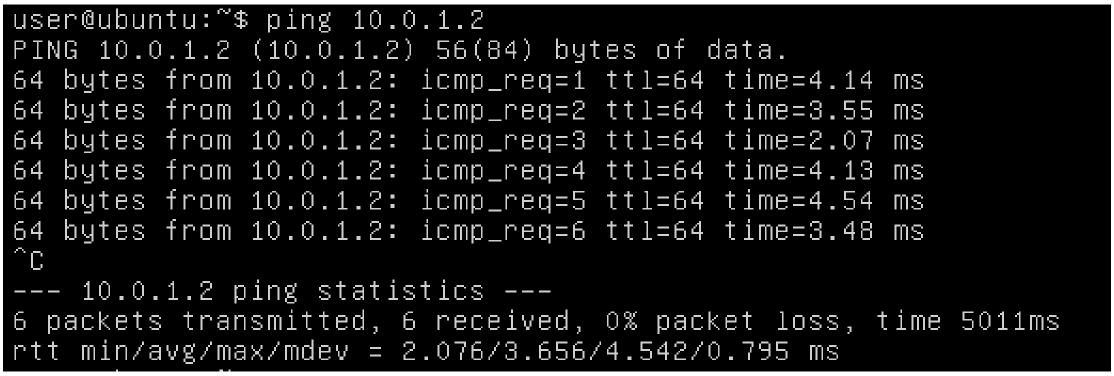
```
user@ubuntu:~/Downloads$ sudo python vpn\(VPNServer1\).py
[sudo] password for user:
Des mac: ff:ff:ff:ff:ff:ff Src mac: 00:50:56:c0:00:02 Ptotocol: 8
It's NOT my packet
Des mac: ff:ff:ff:ff:ff:ff Src mac: 00:50:56:c0:00:03 Ptotocol: 8
It's NOT my packet
Des mac: ff:ff:ff:ff:ff:ff Src mac: 00:50:56:c0:00:02 Ptotocol: 8
It's NOT my packet
Des mac: ff:ff:ff:ff:ff:ff Src mac: 00:50:56:c0:00:03 Ptotocol: 8
It's NOT my packet
Des mac: 01:00:5e:00:00:fc Src mac: 00:50:56:c0:00:02 Ptotocol: 8
It's NOT my packet
```

在 VPNServer1 中启动该程序。



```
user@ubuntu:~/Downloads$ sudo python vpn\(VPNServer2\).py
[sudo] password for user:
Des mac: 00:00:00:00:00:00 Src mac: 00:00:00:00:00:00 Ptotocol: 8
It's NOT my packet
Des mac: 00:0c:29:f2:0a:0c Src mac: 00:0c:29:b5:f0:43 Ptotocol: 8
The packet was sent to ME
It's a IP protocol
Src ip: 172.0.0.1 Dst ip: 172.0.0.2
Des mac: 00:0c:29:f2:0a:0c Src mac: 00:0c:29:b5:f0:43 Ptotocol: 8
The packet was sent to ME
It's a IP protocol
Src ip: 172.0.0.1 Dst ip: 172.0.0.2
Des mac: ff:ff:ff:ff:ff:ff Src mac: 00:50:56:c0:00:04 Ptotocol: 8
It's NOT my packet
```

在 VPNServer2 中启动该程序。

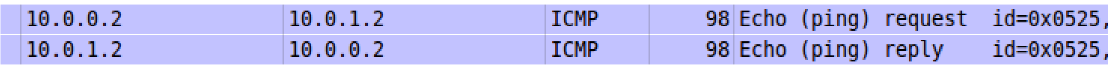


```
user@ubuntu:~$ ping 10.0.1.2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
64 bytes from 10.0.1.2: icmp_req=1 ttl=64 time=4.14 ms
64 bytes from 10.0.1.2: icmp_req=2 ttl=64 time=3.55 ms
64 bytes from 10.0.1.2: icmp_req=3 ttl=64 time=2.07 ms
64 bytes from 10.0.1.2: icmp_req=4 ttl=64 time=4.13 ms
64 bytes from 10.0.1.2: icmp_req=5 ttl=64 time=4.54 ms
64 bytes from 10.0.1.2: icmp_req=6 ttl=64 time=3.48 ms
^C
--- 10.0.1.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5011ms
rtt min/avg/max/mdev = 2.076/3.656/4.542/0.795 ms
```

在 PC1 中 ping PC2，可以顺利 ping 通。

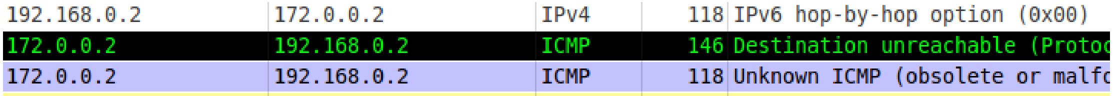
此时对 VPNServer1 的抓包结果如下：

Eth0:



10.0.0.2	10.0.1.2	ICMP	98	Echo (ping) request id=0x0525,
10.0.1.2	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0525,

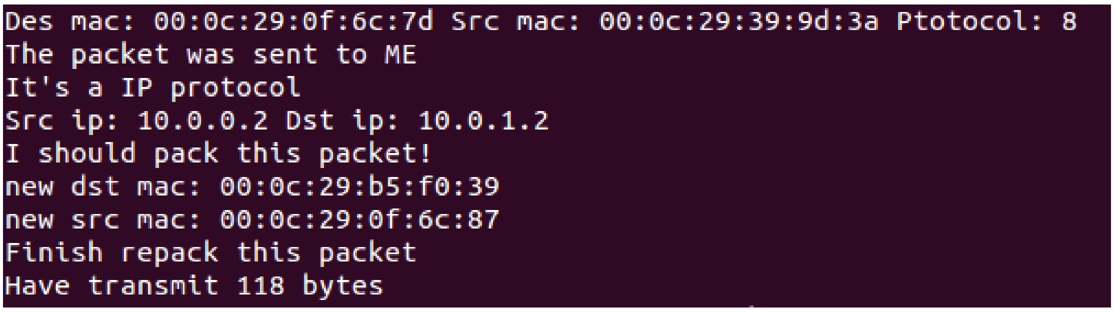
Eht1:



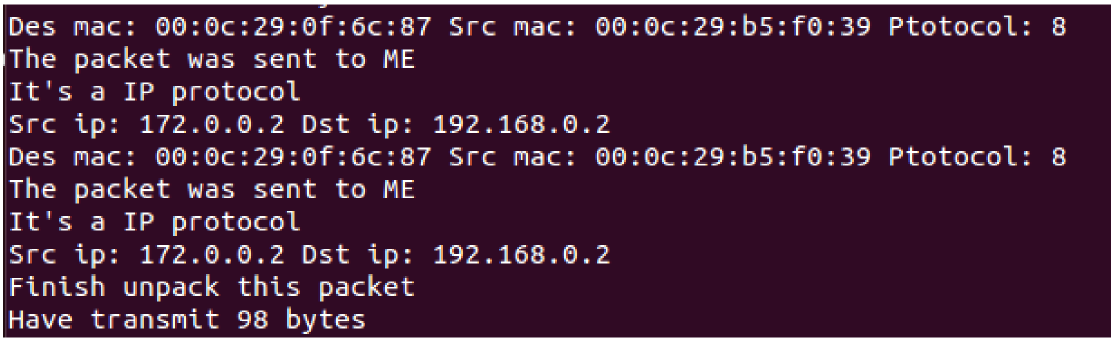
192.168.0.2	172.0.0.2	IPv4	118	IPv6 hop-by-hop option (0x00)
172.0.0.2	192.168.0.2	ICMP	146	Destination unreachable (Protoc
172.0.0.2	192.168.0.2	ICMP	118	Unknown ICMP (obsolete or malfo

由大小可以判断出，字节数为 118 的正是我发送的 vpn 包！

vpn(VPNServer1).py 的输出：



```
Des mac: 00:0c:29:0f:6c:7d Src mac: 00:0c:29:39:9d:3a Ptotocol: 8
The packet was sent to ME
It's a IP protocol
Src ip: 10.0.0.2 Dst ip: 10.0.1.2
I should pack this packet!
new dst mac: 00:0c:29:b5:f0:39
new src mac: 00:0c:29:0f:6c:87
Finish repack this packet
Have transmit 118 bytes
```

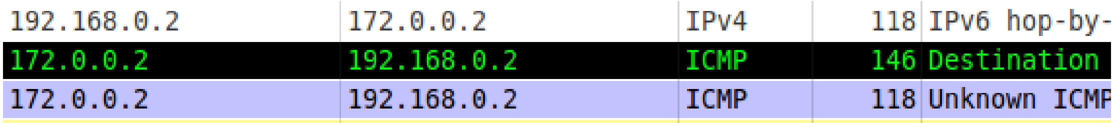


```
Des mac: 00:0c:29:0f:6c:87 Src mac: 00:0c:29:b5:f0:39 Ptotocol: 8
The packet was sent to ME
It's a IP protocol
Src ip: 172.0.0.2 Dst ip: 192.168.0.2
Des mac: 00:0c:29:0f:6c:87 Src mac: 00:0c:29:b5:f0:39 Ptotocol: 8
The packet was sent to ME
It's a IP protocol
Src ip: 172.0.0.2 Dst ip: 192.168.0.2
Finish unpack this packet
Have transmit 98 bytes
```

能清楚看到 repack 和 unpack 的过程。

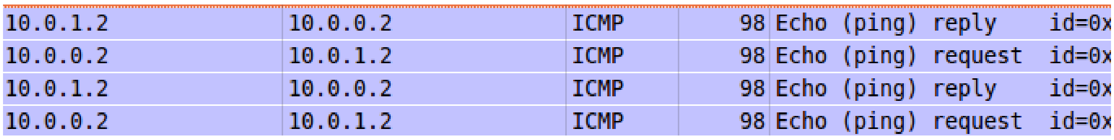
此时对 VPNServer2 的抓包结果如下：

Eth0:



192.168.0.2	172.0.0.2	IPv4	118	IPv6 hop-by-
172.0.0.2	192.168.0.2	ICMP	146	Destination
172.0.0.2	192.168.0.2	ICMP	118	Unknown ICMP

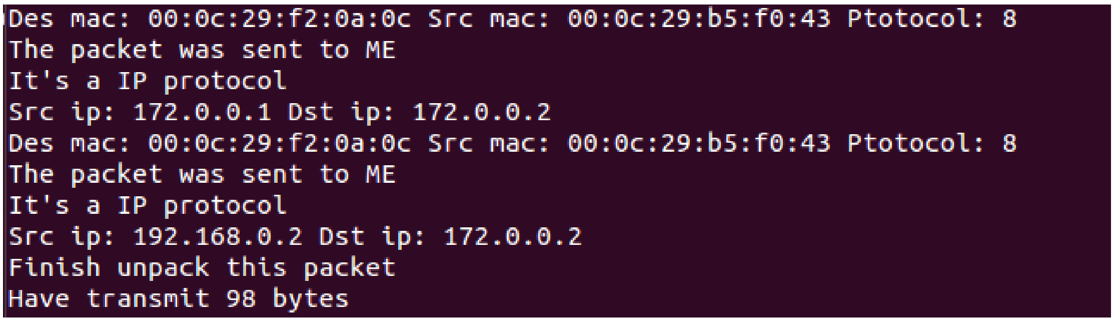
Eht1:



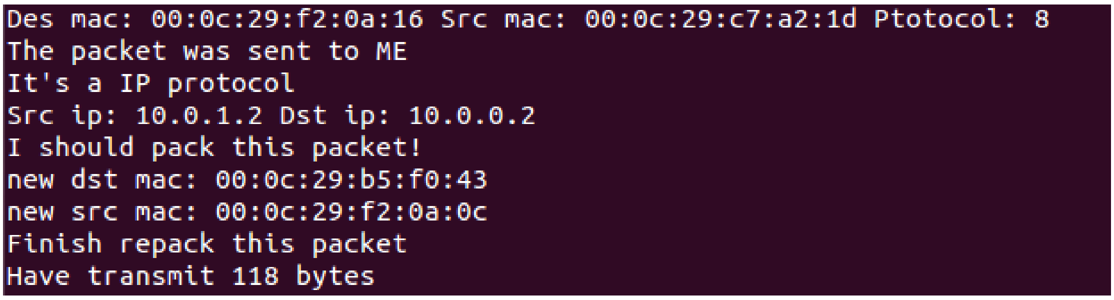
10.0.1.2	10.0.0.2	ICMP	98	Echo (ping) reply id=0x
10.0.0.2	10.0.1.2	ICMP	98	Echo (ping) request id=0x
10.0.1.2	10.0.0.2	ICMP	98	Echo (ping) reply id=0x
10.0.0.2	10.0.1.2	ICMP	98	Echo (ping) request id=0x

和 VPNServer2 的抓包结果相似。

vpn(VPNServer2).py 的输出：



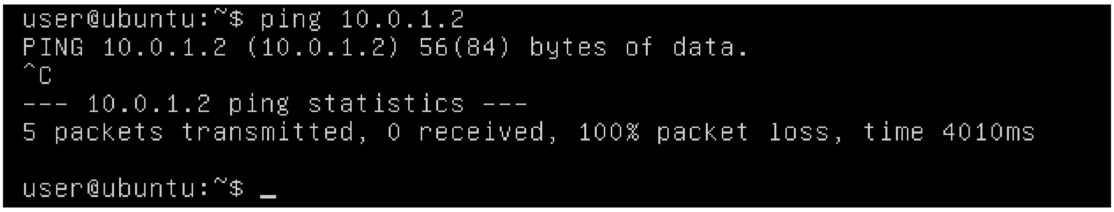
```
Des mac: 00:0c:29:f2:0a:0c Src mac: 00:0c:29:b5:f0:43 Ptotocol: 8
The packet was sent to ME
It's a IP protocol
Src ip: 172.0.0.1 Dst ip: 172.0.0.2
Des mac: 00:0c:29:f2:0a:0c Src mac: 00:0c:29:b5:f0:43 Ptotocol: 8
The packet was sent to ME
It's a IP protocol
Src ip: 192.168.0.2 Dst ip: 172.0.0.2
Finish unpack this packet
Have transmit 98 bytes
```



```
Des mac: 00:0c:29:f2:0a:16 Src mac: 00:0c:29:c7:a2:1d Ptotocol: 8
The packet was sent to ME
It's a IP protocol
Src ip: 10.0.1.2 Dst ip: 10.0.0.2
I should pack this packet!
new dst mac: 00:0c:29:b5:f0:43
new src mac: 00:0c:29:f2:0a:0c
Finish repack this packet
Have transmit 118 bytes
```

能清楚看到 repack 和 unpack 的过程。

如果停止运行 vpn(VPNServer1).py，结果如下：



```
user@ubuntu:~$ ping 10.0.1.2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
^C
--- 10.0.1.2 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4010ms

user@ubuntu:~$ _
```

不能再 ping 通

八、个人思考

有了之前静态路由实验的基础，感觉做这次实验还是比较轻松的，在这次实验过程中我又加深了对二三层协议之间衔接的认识。同时也了解了 VPN 大概的工作原理。