《计算机网络实验》实验报告

实验名称：VPN 设计、实现与分析

姓名：胡育玮

学号：171860574

邮箱：yeevee@qq.com

班级：17 级计算机科学与技术系 2 班

一、实验目的

1、了解 VPN 的基本实现原理

2、了解 VPN 的内部实现细节

二、数据结构说明

（1）使用了 Linux 系统自带的 ip，icmp，ethhdr 结构体：用来解读接收到的包和填写要发送的包。

（2）设备信息表：

```
struct device_item //设备信息表包含的信息
{
    uint32_t ifIndex;      //接口信息
    uint8_t mac_addr[6];   //MAC addr
    char interface[16];    //接口名称
    uint32_t IP;           //IP地址:二进制表示
} device[MAX_DEVICE];
```

（3）路由表、ARP 表：我的实现中没有。因为在这个实验中 VPNserver 只有 2 个，只需要在每个 server 中存下要发往的 server 的 IP 地址即可，不需要路由表。（若 server 数量很多，则可以让 server 查类似路由表的东西，在“路由表”中以原来的 IP 包的目的地址来查对应的下一跳 server 的 IP 地址）。ARP 表同样也为了简化而不设立，改用数组来存储链路中下一跳的 MAC 地址。

三、配置文件说明

在我的实现中，不需要配置文件。但必要的配置信息是必须要有的，比如 VPN 子网内部主机的 IP 地址和 MAC 地址要配置好；VPNserver 往外发包的下一跳 IP 地址要指定；等等。

在我的实现中，采用在 2 个 VPNserver 的程序中设立初始化好的全局变量和初始化函数的方式来实现机器属性的配置：

VPNserver1：

```
char IPeth0[INET_ADDRSTRLEN] = "10.0.0.1";    //server1 中： 0为内网，1为外网
char IPeth1[INET_ADDRSTRLEN] = "192.168.0.2"; //
char vpn_next[INET_ADDRSTRLEN] = "172.0.0.2";  //对server1来说，vpnnext是路由器
uint32_t vpnnext;
uint32_t netmask = 0xFFFFFF00;

//00:0c:29:e4:41:e2
uint8_t destMACin[6] = {0x00, 0x0c, 0x29, 0xe4, 0x41, 0xe2};    //内网主机的MAC地址
//00:0c:29:3c:e1:47
//00:0c:29:4c:a7:03
uint8_t destMACout[6] = {0x00, 0x0c, 0x29, 0x4c, 0xa7, 0x03};  //外网下一跳的MAC地址：对server1
r1的网卡

char IFNAME0[] = "eth0";
char IFNAME1[] = "eth1";
struct sockaddr_ll from;       //存储recvfrom时对方的socket数据结构:这里是ll
char sendpacket[PACKET_SIZE]; //发送ICMP数据报缓存
char recvpacket[PACKET_SIZE]; //接收ICMP数据报缓存
int sockfd;
```

VPNserver2：

```
char IPeth0[INET_ADDRSTRLEN] = "172.0.0.2";    //server2中： 1为内网，0为外网
char IPeth1[INET_ADDRSTRLEN] = "10.0.1.1";  //
char vpn_next[INET_ADDRSTRLEN] = "192.168.0.2"; //对server2来说
uint32_t vpnnext;
uint32_t netmask = 0xFFFFFF00;

//00:0c:29:f5:53:01
uint8_t destMACin[6] = {0x00, 0x0c, 0x29, 0xf5, 0x53, 0x01};  //内网主机的MAC地址
//00:0c:29:3c:e1:51
//00:0c:29:4c:a7:0d
uint8_t destMACout[6] = {0x00, 0x0c, 0x29, 0x4c, 0xa7, 0x0d}; //外网下一跳的MAC地址：对server2
2的网卡

char IFNAME0[] = "eth0";
char IFNAME1[] = "eth1";
struct sockaddr_ll from;       //存储recvfrom时对方的socket数据结构:这里是ll
char sendpacket[PACKET_SIZE]; //发送ICMP数据报缓存
char recvpacket[PACKET_SIZE]; //接收ICMP数据报缓存
int sockfd;
```

初始化函数：getIfInfo()：获取当前主机（VPNserver）的 MAC 地址和 ifindex，并将其存储到设备信息表中。函数的具体实现与实验 4 中的完全一致，不再赘述。在 main 函数内一开始，先调用 getIfInfo()来获取本机 MAC 地址和 ifindex，然后不断循环运行我写的 VPN 路由程序。

四、程序设计思路及运行流程

该程序要求在 VPNserver 上实现 VPN 路由。程序运行流程描述如下：

1. 收到 IP 包（只收 IP 包）之后，判断是来自 VPN 子网内部还是外部互联网

2. 若是来自子网内部的 ICMP 包，则说明该包是要转发出去的，此时：

把原来的以太网包头去掉，把其 IP 包头、ICMP 包头、ICMP 数据段封装到一个新的 ICMP 包的数据段，然后添加新的 ICMP 包头、IP 包头、以太网包头，然后从指向外网的网卡发送。

3. 若是来自外部互联网的 ICMP 包，则把其封装在 ICMP 数据段的完整的 IP 包取出，作为真正的 IP 包，然后重新填充以太网包头，然后从指向内网的网卡发送至内网主机。

转发规则：收到源 IP 地址不是当前 VPN 子网的包，则从指向子网的网卡转发；

收到源 IP 地址是当前 VPN 子网的包，则从指向外网的网

卡转发至外网。

（这是一个简化版的转发规则，默认 VPNserver 接收到的包只有 PC1 ping PC2 的包。实际上做这个路由应当看收到的包的目的 IP：若目的 IP 是内网的，则解开封装，发至内网；若源 IP 是内网而目的 IP 是外网，则对包进行封装然后发送）

程序代码：该 VPN 路由程序写在 vpn_routing()函数内：

```
int n = 0;
int fromlen = sizeof(from);
extern int errno;
struct ip *ip; //此数据结构可以在netinet/ip.h中查看
struct ethhdr *eth;


while (1)  //循环收包
{
    if ((n = recvfrom(sockfd, recvpacket, sizeof(recvpacket), 0, (struct sockaddr *)&from, &fromlen)) < 0)
    {
        if (errno != EINTR)
        {
            perror("recvfrom error!\n");
            return;
        }
    }

    else
        break;
}
```

首先设立 ip，eth 这两个解读收到的字节的变量，然后循环收包：

```
eth = (struct ethhdr *)recvpacket;

switch (ntohs(eth->h_proto))
{
case ETH_P_IP:
    //首先判断源地址是不是同一内网的
    ip = (struct ip *)(eth + 1);
    if (ip->ip_p != IPPROTO_ICMP)
        break;
```

然后看收到的包是否为 ICMP 包，不是则退出。

```
printf("IP src addr: %s\n", inet_ntoa(ip->ip_src));
if (from_inner(ip->ip_src.s_addr, device[0].IP)) //是从内网发出去的包
{
    printf("Received packet from VPN subnet!\n");
    //把原来的以太网包头去掉，把其整个IP包的封装到一个新的IP包的内容字段，填充新的IP包头，以太网包头，然后用指向外网的端口发送
    memset(sendpacket, 0, sizeof(sendpacket));
    struct ethhdr *ethsend = (struct ethhdr *)sendpacket;
    struct ip *ipsend = (struct ip *)(ethsend + 1);
    struct icmp *icmpsend = (struct icmp *)(ipsend + 1);
    memcpy(sendpacket + 14 + 20 + 8, recvpacket + 14, 84);
```

然后判断**是否是从内网发来 server 的包**，若是则将该包的 **IP 头、ICMP 头、ICMP 数据段**复制到一个**新的 ICMP 包的数据段**。

```
struct icmp *icmpori = (struct icmp *)(recvpacket + 14 + 20);
icmpsend->icmp_type = icmpori->icmp_type;
icmpsend->icmp_code = 0;  //ECHO
icmpsend->icmp_cksum = 0;   //校验值 :后面再重新计算，因为要计算整个包的检验和
icmpsend->icmp_seq = 3;  //包序号
icmpsend->icmp_id = 2;
icmpsend->icmp_cksum = cal_chksum((unsigned short *)icmpsend, 8+84);

struct in_addr temp = {device[1].IP};
ipsend->ip_hl = 5;
ipsend->ip_v = 4;
ipsend->ip_id = htons(IPID);
ipsend->ip_ttl = ip->ip_ttl;
ipsend->ip_p = IPPROTO_ICMP; //IP包
//ipsend->ip_len = htons(sizeof(struct ip) + 8 + DATA_LEN);
ipsend->ip_len = htons(84 + 8 + 20);
ipsend->ip_sum = 0;
```

然后填充新的 ICMP 头、IP 头。其中新 IP 头的源 IP 为当前 VPNserver 的指向外网网卡的 IP 地址；目的 IP 为**原来的 IP 包的目的 IP 所在的 VPN 子网对应的 server 的连到外网的网卡的 IP**。新 IP 包的类型为 ICMP。

```
//ipsend->ip_src = ip->ip_src;

ipsend->ip_src = temp;
temp.s_addr = vpnnext;
ipsend->ip_dst = temp;

ipsend->ip_sum = cal_chksum((unsigned short *)ipsend, 20); //ip header 20

ethsend->h_proto = htons(ETH_P_IP);
memcpy(ethsend->h_source, device[1].mac_addr, 6);
memcpy(ethsend->h_dest, destMACout, 6); //发往外网
```

然后填充新的以太网头。本实验中这个**新以太网头的目的 MAC 地址为该 server 连到的路由器网卡的 MAC 地址**。

```
struct sockaddr_ll dest_addr =
{
    .sll_family = PF_PACKET,
    .sll_protocol = htons(ETH_P_IP),
    .sll_halen = ETH_ALEN,
    .sll_ifindex = device[1].ifIndex,
};
memcpy(&dest_addr.sll_addr, destMACout, ETH_ALEN);


int packetsize = 84 + 8 + 34;


if (sendto(sockfd, sendpacket, packetsize, 0, (struct sockaddr *)&dest_addr, sizeof(struct sockaddr_l

{
    perror("VPN sendto error\n");
    return;
}

else
{
    printf("向外网转发成功！\n");
}
```

然后正式向外网转发。

```
printf("Received packet from internet!\n");
memset(sendpacket, 0, sizeof(sendpacket));
struct ethhdr *ethsend = (struct ethhdr *)sendpacket;
//struct ip *ipsend = (struct ip *)(sendpacket + 14);
//struct icmp *icmp = (struct icmp *)(recvpacket + 14 + 20);
memcpy(sendpacket + 14, (recvpacket + 14 + 20 + 8), 84);
//memcpy((void *)ipsend, (const void *)(icmp + 1), 84); //包长度:8+56+20=84

//uint32_t destIP = ipsend->ip_dst;
ethsend->h_proto = htons(ETH_P_IP);
memcpy(ethsend->h_source, device[0].mac_addr, 6);
memcpy(ethsend->h_dest, destMACin, 6); //发往内网
```

如果判断**是从外网发来的包**，则将**其 ICMP 数据段的内容复制出来，作为新的 IP 包的 IP 头、ICMP 头、ICMP 数据段，然后添加新的以太网头**。这里新以太网头的目的 MAC 地址为内网主机的 MAC 地址。

```
struct sockaddr_ll dest_addr =
{
    .sll_family = PF_PACKET,
    .sll_protocol = htons(ETH_P_IP),
    .sll_halen = ETH_ALEN,
    .sll_ifindex = device[0].ifIndex,
};

int packetsize = 84 + 14;

memcpy(&dest_addr.sll_addr, destMACin, ETH_ALEN);
if (sendto(sockfd, sendpacket, packetsize, 0, (struct sockaddr *)&dest_addr, sizeof(struct sockadd

{
    perror("VPN sendto error\n");
    return;
}
else
    printf("向内网转发成功！\n");
```

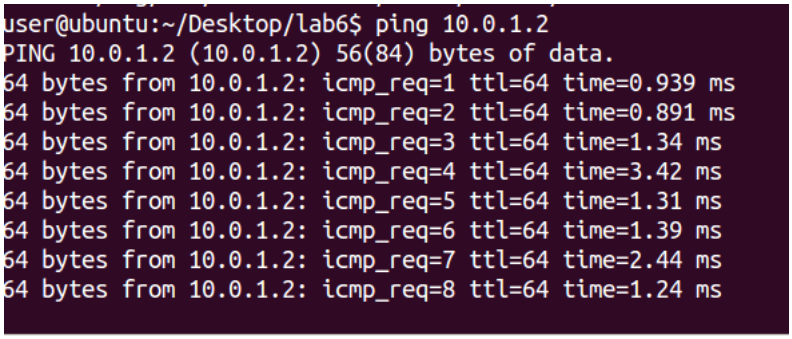
然后内网转发。

至此，VPN 路由程序已结束。在 main()函数中，建立 socket，然后循环执行 vpn_routing()路由函数。

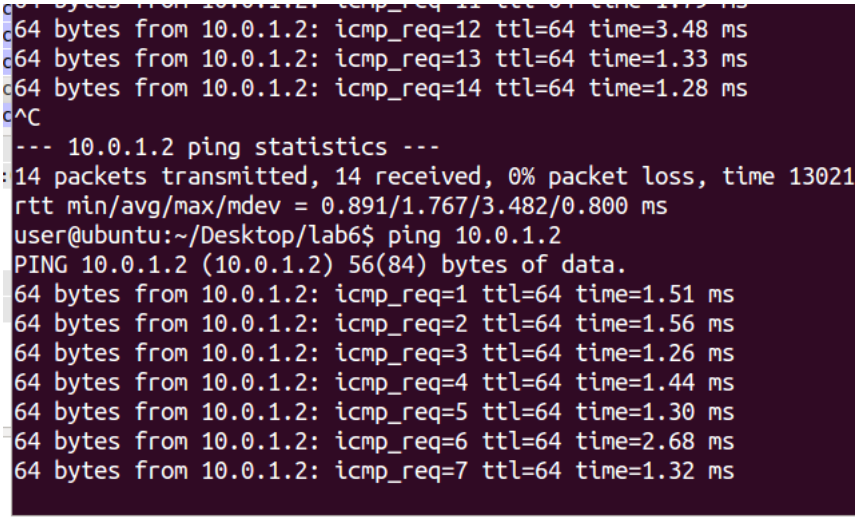
```
int main()
{
    init();
    if ((sockfd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_IP))) < 0) //只收IP包
    {
        perror("main() socket init error!\n");
        exit(1);
    }

    while (1)
    {
        memset(recvpacket, 0, sizeof(recvpacket));
        vpn_routing();
    }

    close(sockfd);

    return 0;
}
```

五、运行结果截图

在两个 VPNserver 上开启路由程序后，在 PC1 上使用系统自带的 ping 命令来 ping PC2：



可见成功 ping 通。此时关闭某个 VPNserver 中运行的路由程序：



```
user@ubuntu:~/Desktop/lab6$ 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=1.51 ms
64 bytes from 10.0.1.2: icmp_req=2 ttl=64 time=1.56 ms
64 bytes from 10.0.1.2: icmp_req=3 ttl=64 time=1.26 ms
64 bytes from 10.0.1.2: icmp_req=4 ttl=64 time=1.44 ms
64 bytes from 10.0.1.2: icmp_req=5 ttl=64 time=1.30 ms
64 bytes from 10.0.1.2: icmp_req=6 ttl=64 time=2.68 ms
64 bytes from 10.0.1.2: icmp_req=7 ttl=64 time=1.32 ms
^C
--- 10.0.1.2 ping statistics ---
15 packets transmitted, 7 received, 53% packet loss, time 14073ms
rtt min/avg/max/mdev = 1.262/1.586/2.681/0.459 ms
user@ubuntu:~/Desktop/lab6$
```

可见关闭后便无法 ping 通。说明该程序有效。

VPN 路由程序运行截图：

```
user@ubuntu: ~/Desktop/lab6
Received packet from internet!
向内网转发成功！
IP src addr: 172.0.0.2
Received packet from internet!
向内网转发成功！
IP src addr: 10.0.0.2
Received packet from VPN subnet!
向外网转发成功！
IP src addr: 172.0.0.2
Received packet from internet!
向内网转发成功！
IP src addr: 172.0.0.2
Received packet from internet!
向内网转发成功！
IP src addr: 10.0.0.2
Received packet from VPN subnet!
向外网转发成功！
IP src addr: 172.0.0.2
Received packet from internet!
向内网转发成功！
IP src addr: 172.0.0.2
Received packet from internet!
向内网转发成功！
```

六、参考资料

无

七、代码思考

在封装原来的 IP 包时若添加的新 IP 包头的协议字段填 ICMP，且直接将原 IP 包的 IP 包头、ICMP 包头、ICMP 数据段直接拷贝到新 IP 包头的后面，则路由器接收到这个封装好的包以后会判断这是一个错误的包而丢弃，不给转发。因此后面我采用了 新以太网包头-新 IP 包头-新 ICMP 包头-原来 IP 包头-原来 ICMP 包头-原来 ICMP 数据段 这样的封装模式，**把原来的 IP 包封装到新的 ICMP 包的数据段，同时正确地设置新 IP 包的大小**，这样路由器就不会判断是错误的包了。

其他遇到的 bug：

1. 重新封装的 IP 包头长度填错，导致路由器转发时包的尾部被截断

2. 重新封装的 ICMP 报头的 ICMP 类型字段：应随实际 ICMP 报文，而不是一直填 ICMP-request