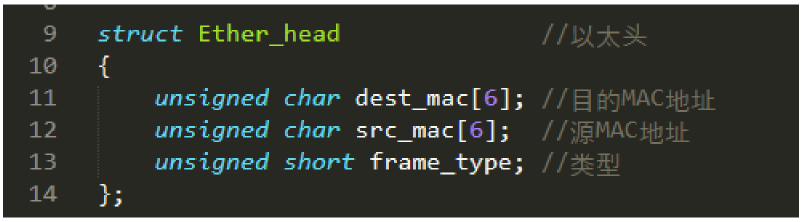
Lab6 实验报告

一、 实验目的

1.进一步理解 VPN 的工作原理和内部实现细节。
2.加强对 VPN 封包、解包的认知。
3.高度模拟 VPN 进行路由的方法。

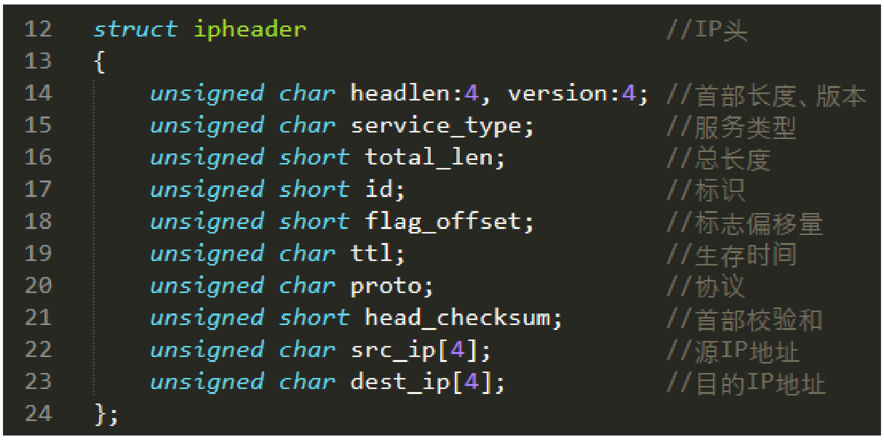
二、 数据结构说明

1、 以太头



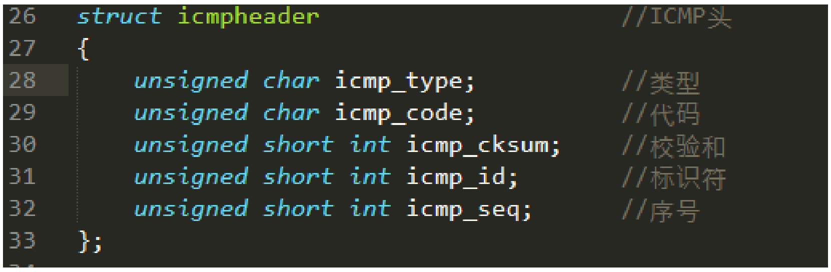
```c
struct Ether_head               //以太头
{
    unsigned char dest_mac[6]; //目的MAC地址
    unsigned char src_mac[6];  //源MAC地址
    unsigned short frame_type; //类型
};
```

2、 IP 头



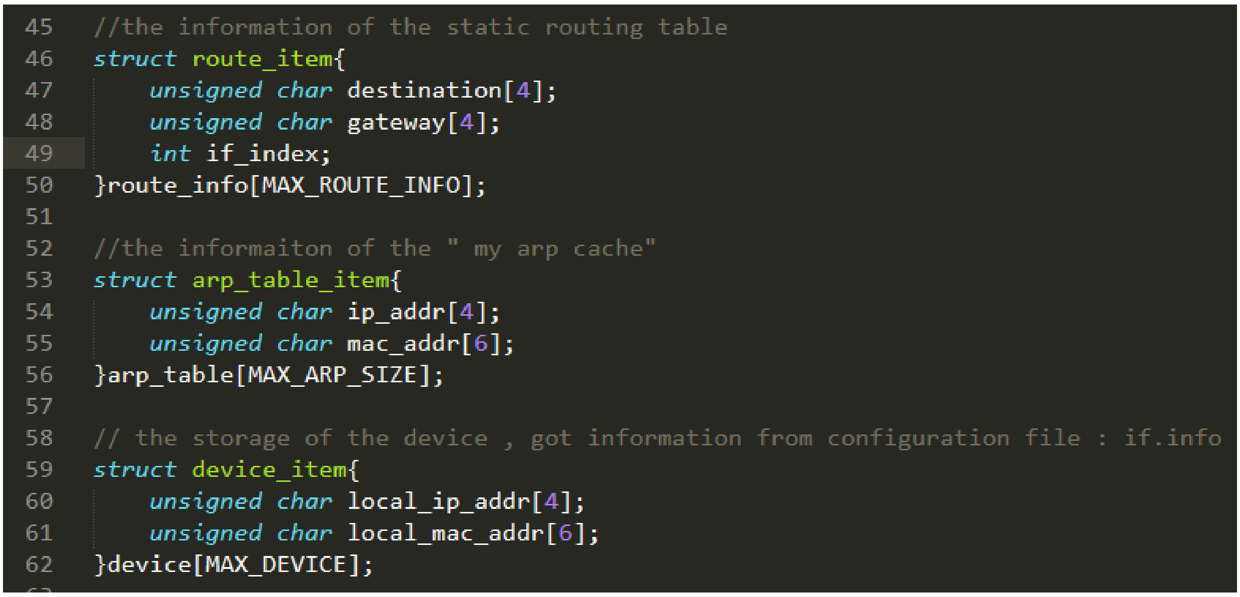
```c
struct ipheader                                 //IP头
{
    unsigned char headlen:4, version:4; //首部长度、版本
    unsigned char service_type;         //服务类型
    unsigned short total_len;           //总长度
    unsigned short id;                  //标识
    unsigned short flag_offset;         //标志偏移量
    unsigned char ttl;                  //生存时间
    unsigned char proto;                //协议
    unsigned short head_checksum;       //首部校验和
    unsigned char src_ip[4];            //源IP地址
    unsigned char dest_ip[4];           //目的IP地址
};
```

3、 ICMP 头



```c
struct icmpheader                               //ICMP头
{
    unsigned char icmp_type;            //类型
    unsigned char icmp_code;            //代码
    unsigned short int icmp_cksum;      //校验和
    unsigned short int icmp_id;         //标识符
    unsigned short int icmp_seq;        //序号
};
```

4、 路由表、ARP 缓存表、设备信息表



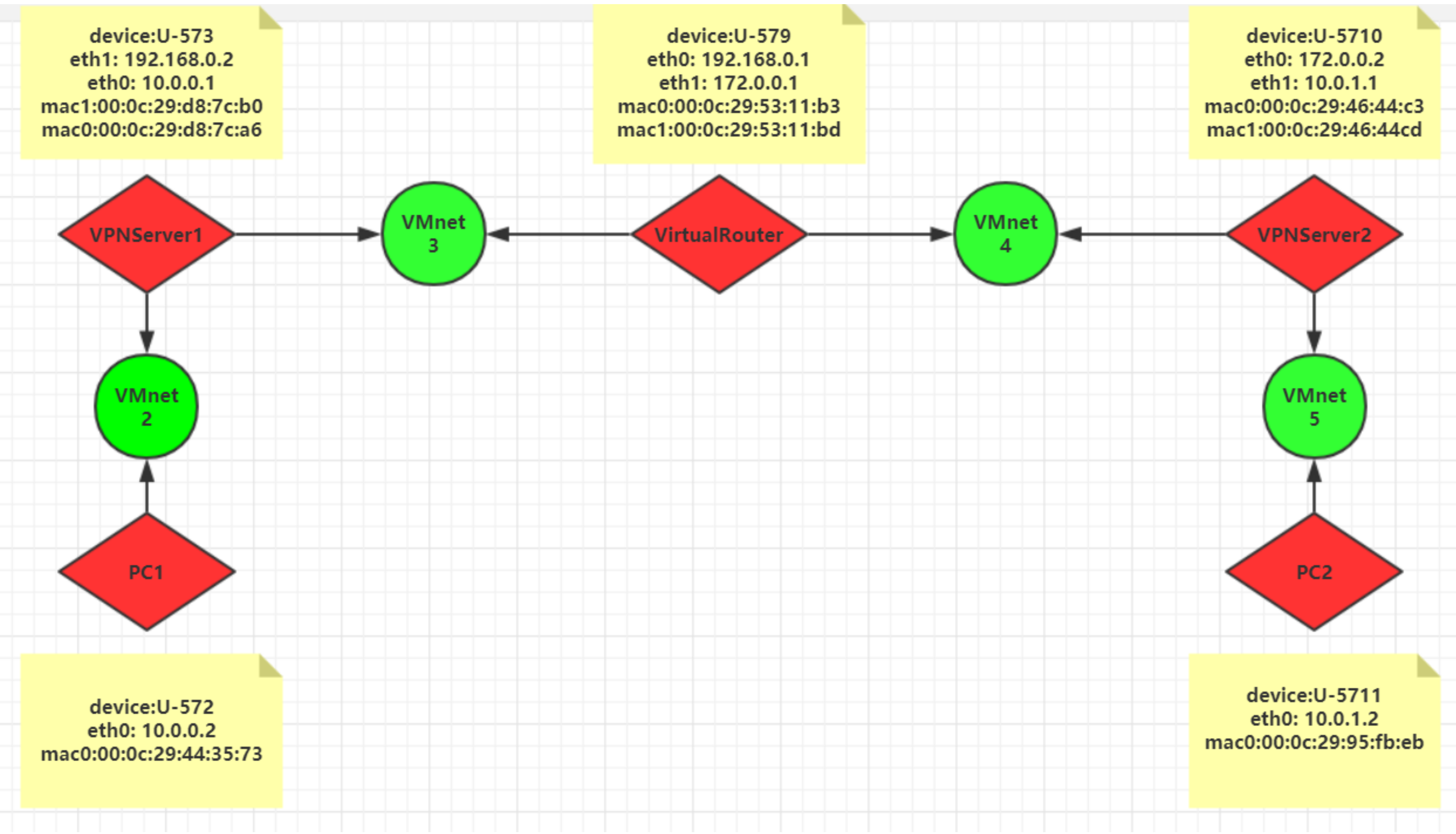
```c
//the information of the static routing table
struct route_item{
    unsigned char destination[4];
    unsigned char gateway[4];
    int if_index;
}route_info[MAX_ROUTE_INFO];

//the informaiton of the " my arp cache"
struct arp_table_item{
    unsigned char ip_addr[4];
    unsigned char mac_addr[6];
}arp_table[MAX_ARP_SIZE];

// the storage of the device , got information from configuration file : if.info
struct device_item{
    unsigned char local_ip_addr[4];
    unsigned char local_mac_addr[6];
}device[MAX_DEVICE];
```

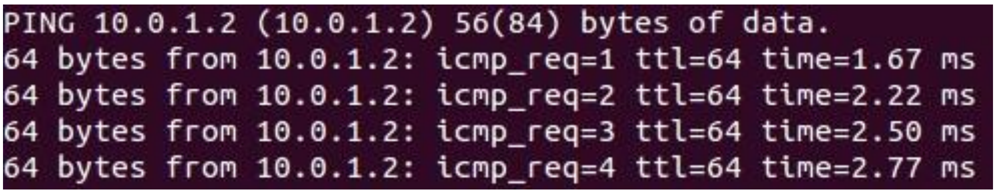
三、 实验思路

1、 实验搭建拓扑极其配置:



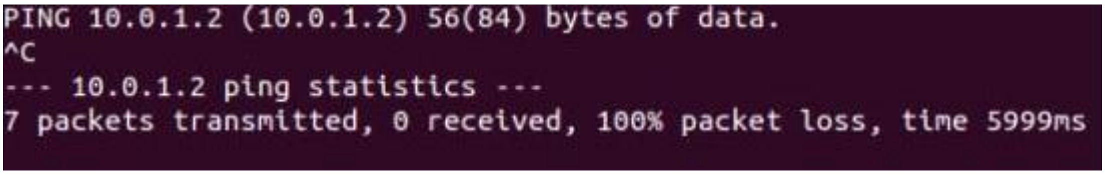
2、 运行结果：

VPN1 和 VPN2 都打开：



```
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=1.67 ms
64 bytes from 10.0.1.2: icmp_req=2 ttl=64 time=2.22 ms
64 bytes from 10.0.1.2: icmp_req=3 ttl=64 time=2.50 ms
64 bytes from 10.0.1.2: icmp_req=4 ttl=64 time=2.77 ms
```

关掉其中一个 VPN：

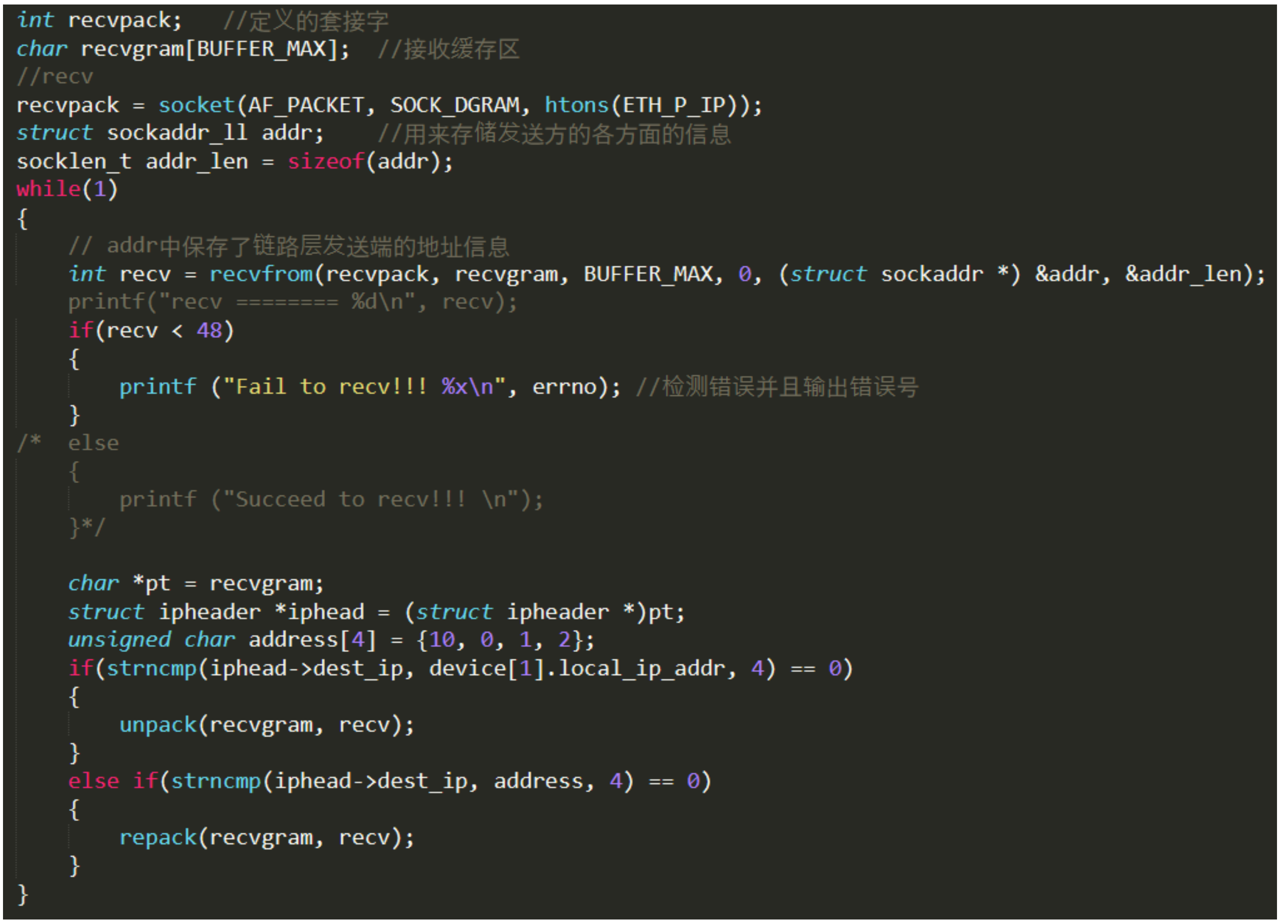


```
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
^C
--- 10.0.1.2 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 5999ms
```

3、 核心代码分析

收包：

收下包之后主要根据包来自外部还是内部来判断是重打包还是解包。

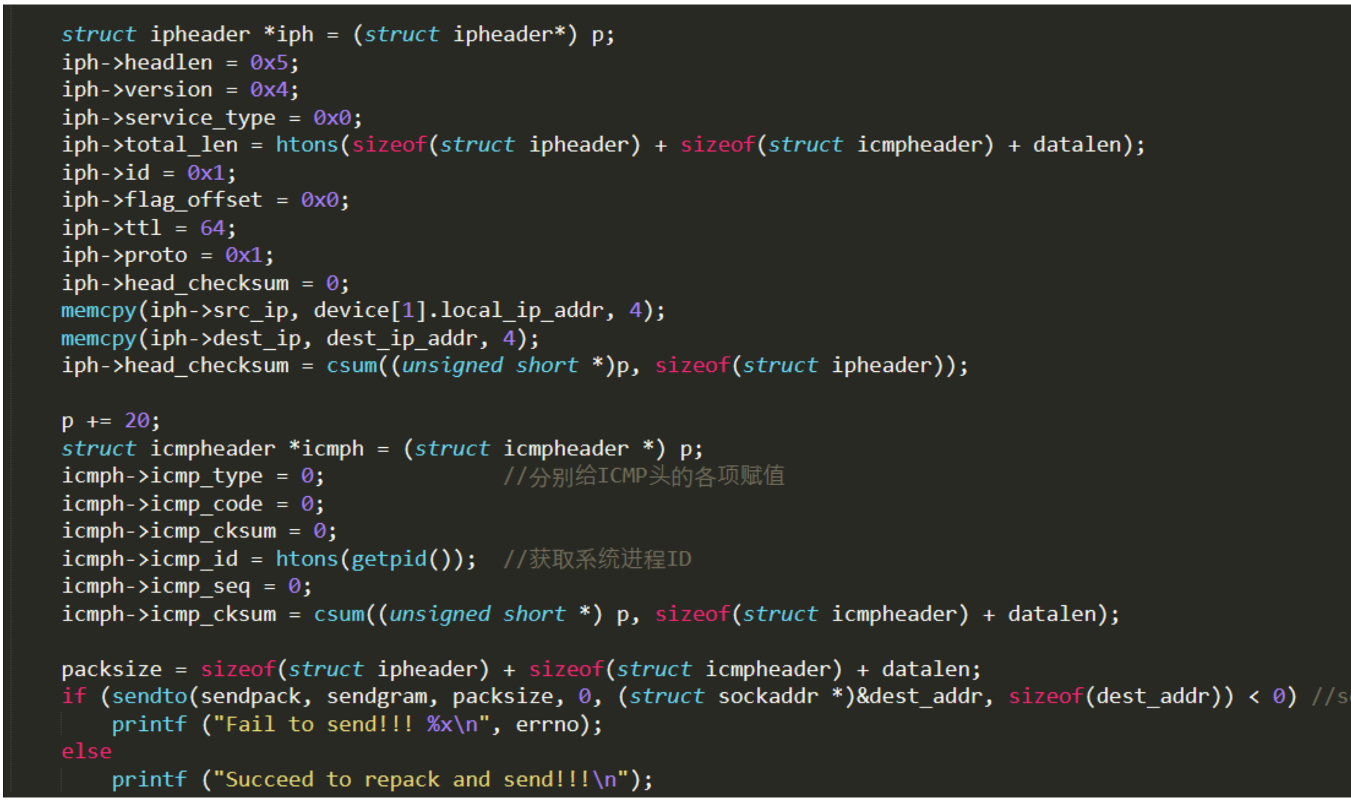


```c
int recvpack;    //定义的套接字
char recvgram[BUFFER_MAX];  //接收缓存区
//recv
recvpack = socket(AF_PACKET, SOCK_DGRAM, htons(ETH_P_IP));
struct sockaddr_ll addr;    //用来存储发送方的各方面的信息
socklen_t addr_len = sizeof(addr);
while(1)
{
    // addr中保存了链路层发送端的地址信息
    int recv = recvfrom(recvpack, recvgram, BUFFER_MAX, 0, (struct sockaddr *) &addr, &addr_len);
    printf("recv ======== %d\n", recv);
    if(recv < 48)
    {
        printf ("Fail to recv!!! %x\n", errno); //检测错误并且输出错误号
    }
/*  else
    {
        printf ("Succeed to recv!!! \n");
    }*/

    char *pt = recvgram;
    struct ipheader *iphead = (struct ipheader *)pt;
    unsigned char address[4] = {10, 0, 1, 2};
    if(strncmp(iphead->dest_ip, device[1].local_ip_addr, 4) == 0)
    {
        unpack(recvgram, recv);
    }
    else if(strncmp(iphead->dest_ip, address, 4) == 0)
    {
        repack(recvgram, recv);
    }
}
```

重打包：

主要是在原包的基础上添加了新的 IP 头和 ICMP 头。

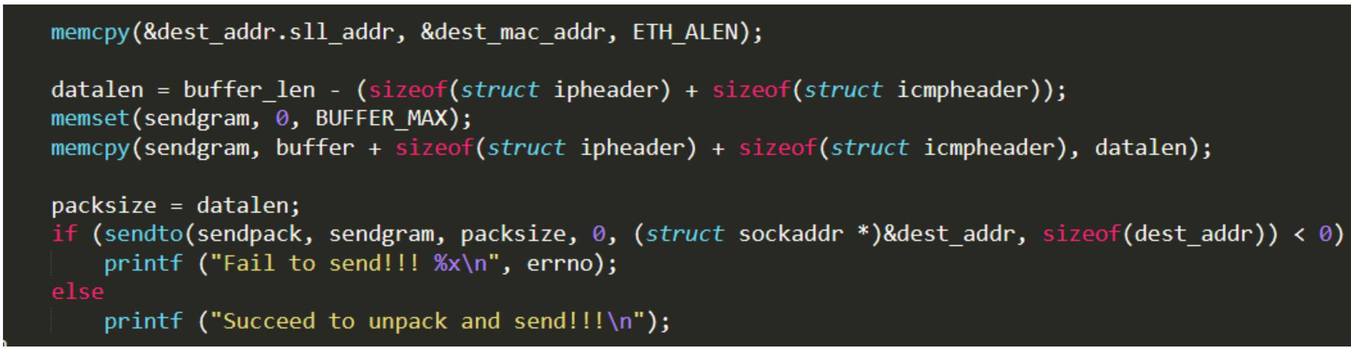


```
struct ipheader *iph = (struct ipheader*) p;
iph->headlen = 0x5;
iph->version = 0x4;
iph->service_type = 0x0;
iph->total_len = htons(sizeof(struct ipheader) + sizeof(struct icmpheader) + datalen);
iph->id = 0x1;
iph->flag_offset = 0x0;
iph->ttl = 64;
iph->proto = 0x1;
iph->head_checksum = 0;
memcpy(iph->src_ip, device[1].local_ip_addr, 4);
memcpy(iph->dest_ip, dest_ip_addr, 4);
iph->head_checksum = csum((unsigned short *)p, sizeof(struct ipheader));

p += 20;
struct icmpheader *icmph = (struct icmpheader *) p;
icmph->icmp_type = 0;           //分别给ICMP头的各项赋值
icmph->icmp_code = 0;
icmph->icmp_cksum = 0;
icmph->icmp_id = htons(getpid());  //获取系统进程ID
icmph->icmp_seq = 0;
icmph->icmp_cksum = csum((unsigned short *) p, sizeof(struct icmpheader) + datalen);

packsize = sizeof(struct ipheader) + sizeof(struct icmpheader) + datalen;
if (sendto(sendpack, sendgram, packsize, 0, (struct sockaddr *)&dest_addr, sizeof(dest_addr)) < 0) //s
    printf ("Fail to send!!! %x\n", errno);
else
    printf ("Succeed to repack and send!!!\n");
```

解包：
创建新的包，将 IP 头和 ICMP 头去掉就将原来的包还原了出来。



```
memcpy(&dest_addr.sll_addr, &dest_mac_addr, ETH_ALEN);

datalen = buffer_len - (sizeof(struct ipheader) + sizeof(struct icmpheader));
memset(sendgram, 0, BUFFER_MAX);
memcpy(sendgram, buffer + sizeof(struct ipheader) + sizeof(struct icmpheader), datalen);

packsize = datalen;
if (sendto(sendpack, sendgram, packsize, 0, (struct sockaddr *)&dest_addr, sizeof(dest_addr)) < 0)
    printf ("Fail to send!!! %x\n", errno);
else
    printf ("Succeed to unpack and send!!!\n");
```

4、参考资料

主要参考了实验 4 的链路层传输的一些函数的格式。

四、 实验的创新点

这次实验的话感觉就是在实验 4 的基础上做的，VPN 的作用就是添加包头信息，然后进行内部传输，到达指定的服务器后再进行解包，将原来的信息还原出来。所以看懂了 VPN 的原理后写起来还是很轻松的。