实验二 RAW SOCKET 编程与以太网帧分析基础

邱梓豪

141130077

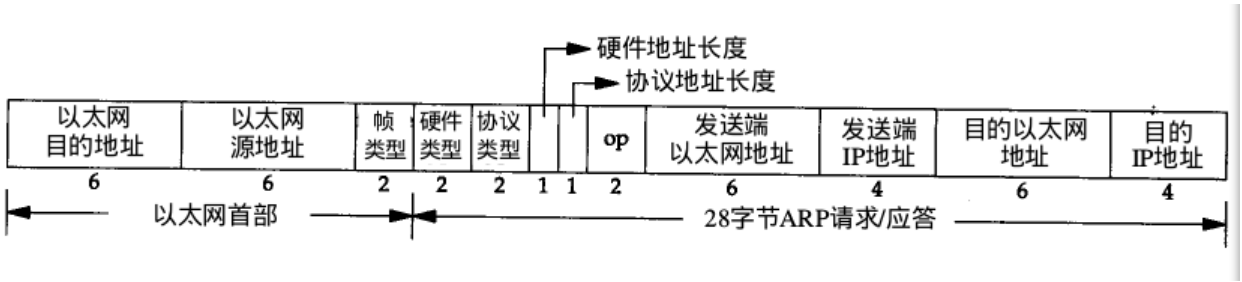
一、实验目的

本实验的主要目的是了解在 linux 系统中的 Raw Socket 编程，并抓取、分析以太网帧，从以太网帧中找出不同层次 PDU 的有关信息。利用 Raw Socket 封装和发送以太网帧的功能，实现 ICMP 包的发送和接收。

二、数据结构说明

在框架代码中，给出了对以太网帧和 ip 数据包的解析，解析采用的是偏移量的方式。我在解析 arp/rarp 数据包时，由于这种数据包的格式较为固定，所以我将先定义数据包的结构体，利用结构体指针完成对数据的解析。

arp/rarp 的数据包结构如下：

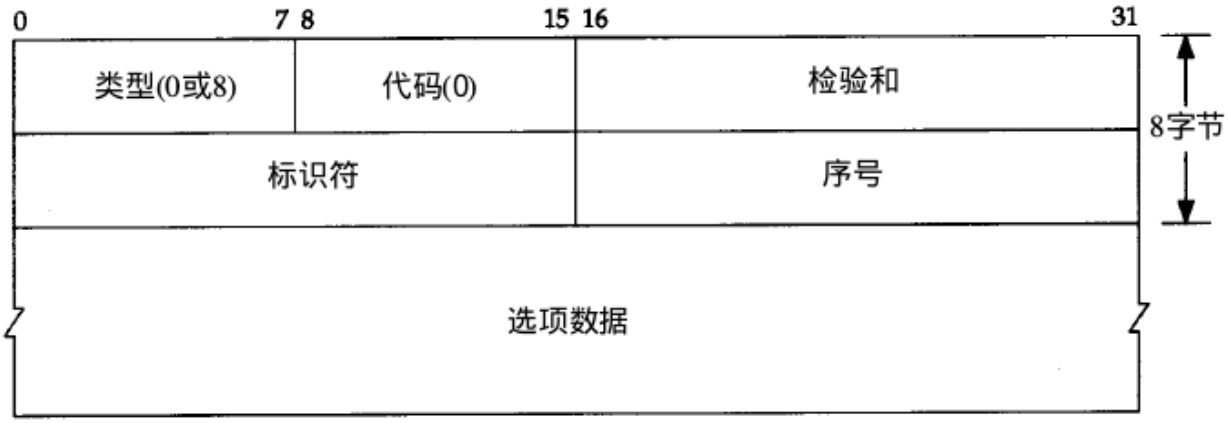


我由此定义的 ARP 结构体如下：

```
struct ARP
{
    unsigned short hw_type;
    unsigned short proto_type;
    unsigned char hw_addr_len;
    unsigned char proto_addr_len;
    unsigned short op_code;
    unsigned char sender_mac[6];
    unsigned char sender_ip[4];
    unsigned char receiver_mac[6];
    unsigned char receiver_ip[4];
};
```

上述结构体表示的是 28 字节的 ARP 请求/应答。其中，hw_type 表示硬件类型，proto_type 表示协议类型，hw_addr_len 表示硬件地址长度，proto_addr_len 表示协议地址长度，op_code 表示操作码（具体的，1 表示 arp 请求，2 表示 arp 应答，3 表示 rarp 请求，4 表示 rarp 应答），sender_mac 为发送者 MAC 地址，sender_ip 为发送者 IP 地址 receiver_mac 为接受者 MAC 地址，receiver_ip 为接受者 IP 地址。

ICMP 数据包结构体如下：



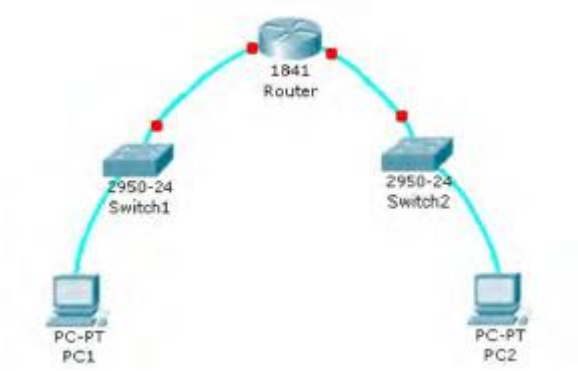
其结构体的定义如下：

```
struct icmp {
    uint8_t  icmp_type;
    uint8_t  icmp_code;
    uint16_t icmp_cksum;
    uint16_t icmp_id;
    uint16_t icmp_seq;
};
```

其中 icmp_type 表示类型，icmp_code 表示代码，icmp_cksum 表示校验和，icmp_id 表示标识符，icmp_seq 表示序号。这里我没有处理选项数据。

三、虚拟机配置

网络拓扑结构如下：



搭建命令：

PC1:　sudo ifconfig eth0 192.168.0.2 netmask 255.255.255.0

sudo route add default gw 192.168.0.1

sudo /etc/init.d/networking restart

PC2:　sudo ifconfig eth0 192.168.1.2 netmask 255.255.255.0

sudo route add default gw 192.168.1.1

sudo /etc/init.d/networking restart

Router:　sudo ifconfig eth0 192.168.0.1 netmask 255.255.255.0

sudo ifconfig eth1 192.168.1.1 netmask 255.255.255.0

sudo echo 1 > /proc/sys/net/ipv4/ip_forward

sudo /etc/init.d/networking restart

测试：在 PC1 上 ping 192.168.1.2，如果能 ping 通，说明连接已经建立。

四、程序设计思路以及运行流程

（1）抓包实验

抓包程序设计思路：

```
if ((sock_fd=socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL)))<0)
{
    printf("error create raw socket\n");
    return -1;
}
```

先建立 socket，第三个参数设置为 htons(ETH_P_ALL)，以接受所有种类的数据包。

```
struct ifreq ethreq;
strncpy(ethreq.ifr_name, "ens33", IFNAMSIZ);
ethreq.ifr_flags |= IFF_PROMISC;
ioctl(sock_fd, SIOCGIFFLAGS, &ethreq);
```

将网卡设置为混杂模式。

```
while(1)
{
    n_read=recvfrom(sock_fd, buffer, 2048, 0, NULL, NULL);
    if(n_read<42)
    {
        printf("n_read: %d\n", n_read);
        printf("error when recv msg\n");
        continue;
    }
    printf("*****************************\n");
```

启动监听，不断从 socket 中读取数据，进行解析。

```
        p=eth_head;
        printf("MAC address: %.2x:%02x:%02x:%02x:%02x:%02x ==> %.2x:%02x:%02x:%
02x:%02x:%02x\n",
            p[6],p[7],p[8],p[9],p[10],p[11],
            p[0],p[1],p[2],p[3],p[4],p[5]);

        // judge what kind of proto is
        char* p_proto=eth_head+12;
        short proto_type=((short)p_proto[0]<<8)+((short)p_proto[1]);
        printf("Protocol (in frame):");
        switch(proto_type){
            case 0x0800: printf("IPv4\n");break;
            case 0x0806: printf("ARP\n");break;
            case 0x8035: printf("RARP\n");break;
            case 0x8864: printf("PPPoE\n");break;
            case 0x86dd: printf("IPv6\n");break;
            case 0x8847: printf("MPLS\n");break;
            default: printf("Other Proto: 0x%x\n", proto_type);
        }
```

从以太网帧头解析出源 MAC 地址，目的 MAC 地址以及协议类型。

```
if (proto_type==0x0800) // IP
{
    ip_head=eth_head+14;
    p=ip_head+12;
    printf("IP: %d.%d.%d.%d ==> %d.%d.%d.%d\n",
        p[0],p[1],p[2],p[3],p[4],p[5],p[6],p[7]);
    proto=(ip_head+9)[0];
    p=ip_head+12;
    printf("Protocol:");
    switch(proto){
        case IPPROTO_ICMP:printf("icmp\n");break;
        case IPPROTO_IGMP:printf("igmp\n");break;
        case IPPROTO_IPIP:printf("ipip\n");break;
        case IPPROTO_TCP:printf("tcp\n");break;
        case IPPROTO_UDP:printf("udp\n");break;
        default:printf("Pls query yourself about: 0x%x\n", proto);
    }
}
```

如果是 IP 协议，则从该 UDP 种解析出源 IP，目的 IP 及上层协议类型。

```
if (proto_type==0x0806 || proto_type==0x8035) // ARP/RARP
{
    struct ARP* arp=eth_head+14;
    switch(arp->op_code){
        case 0x1: printf("ARP request\n"); break;
        case 0x2: printf("ARP respons\n"); break;
        case 0x3: printf("RARP request\n"); break;
        case 0x4: printf("RARP respons\n"); break;
    }
    p=arp->sender_mac;
    printf("Sender MAC: %.2x:%02x:%02x:%02x:%02x:%02x\n",
            p[0],p[1],p[2],p[3],p[4],p[5]);
    p=arp->sender_ip;
    printf("Sender IP: %d.%d.%d.%d\n",
            p[0],p[1],p[2],p[3]);
    p=arp->receiver_mac;
    printf("Reciever MAC: %.2x:%02x:%02x:%02x:%02x:%02x\n",
            p[0],p[1],p[2],p[3],p[4],p[5]);
    p=arp->receiver_ip;
    printf("Reciever IP: %d.%d.%d.%d\n",
            p[0],p[1],p[2],p[3]);
}
```

如果是 ARP 或 RARP 协议，则从中解析出操作码，源 IP，目的 IP，源 MAC，目的 MAC。

（2）发送接收 icmp 包实验

```
// Computes the checksum of a packet
uint16_t compute_checksum(void *buf, size_t size) {
    size_t i;
    uint64_t sum = 0;

    for (i = 0; i < size; i += 2) {
        sum += *(uint16_t *)buf;
        buf = (uint8_t *)buf + 2;
    }

    if (size - i > 0) {
        sum += *(uint8_t *)buf;
    }

    while ((sum >> 16) != 0) {
        sum = (sum & 0xffff) + (sum >> 16);
    }

    return (uint16_t)~sum;
}
```

这个函数用于计算 icmp 包的校验和。

```
char *host;
int sockfd;
int error;
struct addrinfo addrinfo_hints;
struct addrinfo *addrinfo_head;
struct addrinfo *addrinfo;
char addrstr[INET6_ADDRSTRLEN];
struct timeval start_time;
struct icmp request;
struct icmp reply;
struct ip_icmp reply_wrapped;
uint16_t id = (uint16_t)getpid();
uint16_t seq = 0;
int bytes_transferred;
```

main()的开头设置了这些变量，其中：host 指向输入的域名或者 ip；sockfd 是建立的 socket；error 作为 getaddrinfo()函数的返回值，通过该返回值判断是否获得了正确的结果；aadrinfo_hints 是一个 addrinfo 型结构体，用于指导产生 socket；getaddrinfo()需要传入一个链表首节点，addrinfo_head 就是这样的结点；addrstr 用于存储目的 ip；request 和 reply 分别是指向发送和接收的 icmp 包。

```
memset(&addrinfo_hints, 0, sizeof(addrinfo_hints));
addrinfo_hints.ai_family = AF_INET;
addrinfo_hints.ai_socktype = SOCK_RAW;
addrinfo_hints.ai_protocol = IPPROTO_ICMP;
```

下面开始设置 addrinfo_hints 的内容：地址族设置为 AF_INET，也就是 ipv4；套接字类型设置为 SOCK_RAW；协议类型设置为 icmp。

```
error = getaddrinfo(host, NULL, &addrinfo_hints, &addrinfo_head);
```

下一步，利用 getaddrinfo()函数将主机名 host 转换成该主机对应的 addrinfo 链表。

```
for (addrinfo = addrinfo_head;
     addrinfo != NULL;
     addrinfo = addrinfo->ai_next) {
    sockfd = socket(addrinfo->ai_family,
                    addrinfo->ai_socktype,
                    addrinfo->ai_protocol);
    if (sockfd == -1) {
        fprintf(stderr, "Error: socket: %s\n", strerror(errno));
        continue;
    }

    /* Socket was successfully created. */
    break;
}
```

利用链表中的每一项的信息尝试建立套接字。

```
switch (addrinfo->ai_family) {
    case AF_INET:
        inet_ntop(AF_INET,
                  &((struct sockaddr_in *)addrinfo->ai_addr)->sin_addr,
                  addrstr,
                  sizeof(addrstr));
        break;
    case AF_INET6:
        inet_ntop(AF_INET6,
                  &((struct sockaddr_in6 *)addrinfo->ai_addr)->sin6_addr

                  addrstr,
                  sizeof(addrstr));
        break;
    default:
        strncpy(addrstr, "<unknown address>", sizeof(addrstr));
}
```

根据不同协议族，对建立 socket 的 addrinfo 结构体进行解析，将 ip 地址放入 addrstr 中。

下面开始不断发送 icmp 数据包，填充 request 中的相关字段，发送到所给的地址。

之后不断监听 sockfd，将监听到的以太网帧数据包放入 reply_wrapped 中，

```
for (seq = 0; ; seq++) {
    long delay = 0;

    memset(&request, 0, sizeof(request));
    request.icmp_type = ICMP_ECHO;
    request.icmp_code = 0;
    request.icmp_cksum = 0;
    request.icmp_id = htons(id);
    request.icmp_seq = htons(seq);
    request.icmp_cksum = compute_checksum(&request, sizeof(request));

    bytes_transferred = sendto(sockfd,
                               &request,
                               sizeof(request),
                               0,
                               addrinfo->ai_addr,
                               addrinfo->ai_addrlen);
    if (bytes_transferred < 0) {
        fprintf(stderr, "Error: sendto: %s\n", strerror(errno));
        exit(EXIT_FAILURE);
    }
```

```
for (;;) {
    struct timeval cur_time;

    gettimeofday(&cur_time, NULL);
    delay = TIMEVAL_TO_USEC(cur_time) - TIMEVAL_TO_USEC(start_t

    memset(&reply_wrapped, 0, sizeof(reply_wrapped));
    bytes_transferred = recvfrom(sockfd,
                                 &reply_wrapped,
                                 sizeof(reply_wrapped),
                                 0,
                                 NULL,
                                 NULL);
```

从 reply_wrapped 中解析出校验和，id，seq 等信息，计算这个包校验和应有的值，若相符，则成功响应，输出相关信息，一轮结束。

```
  reply = reply_wrapped.icmp;
  reply.icmp_cksum = ntohs(reply.icmp_cksum);
  reply.icmp_id = ntohs(reply.icmp_id);
  reply.icmp_seq = ntohs(reply.icmp_seq);
```

五、运行结果截图

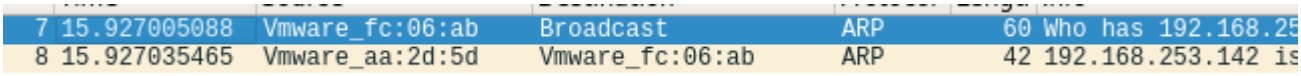
（1）抓包实验

抓到的 icmp 数据包

```
***************************
No: 21 Sat Mar 18 03:08:16 2017
n_read: 98
buffer:00 50 56 fc 06 ab 00 0c 29 aa 2d 5d 08 00 45 00 00 54 4c 02 40 00 40 01 4
5 f1 c0 a8 fd 8e ca 77 20 07 08 00 2f ad 46 7f 00 05 a0 a4 cc 58 1b ce 0e 00 08
09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23
 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37
MAC address: 00:0c:29:aa:2d:5d ==> 00:50:56:fc:06:ab
Protocol (in frame):IPv4
IP: 192.168.253.142 ==> 202.119.32.7
Protocol:icmp
***************************
No: 22 Sat Mar 18 03:08:16 2017
n_read: 98
buffer:00 0c 29 aa 2d 5d 00 50 56 fc 06 ab 08 00 45 00 00 54 c6 41 00 00 80 01 c
b b1 ca 77 20 07 c0 a8 fd 8e 00 00 37 ad 46 7f 00 05 a0 a4 cc 58 1b ce 0e 00 08
09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23
 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37
MAC address: 00:50:56:fc:06:ab ==> 00:0c:29:aa:2d:5d
Protocol (in frame):IPv4
IP: 202.119.32.7 ==> 192.168.253.142
Protocol:icmp
***************************
```

抓到的 arp 数据包（上图是我的程序抓到的包，下图是用 wireshark 抓到的包）

```
****************************
No: 13 Sat Mar 18 04:19:29 2017
n_read: 60
buffer:ff ff ff ff ff ff 00 50 56 fc 06 ab 08 06 00 01 08 00 06 04 00 01 00
 50 56 fc 06 ab c0 a8 fd 02 00 00 00 00 00 00 c0 a8 fd 8e 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00
MAC address: 00:50:56:fc:06:ab ==> ff:ff:ff:ff:ff:ff
Protocol (in frame):ARP
ARP request
Sender MAC: 00:50:56:fc:06:ab
Sender IP: 192.168.253.2
Reciever MAC: 00:00:00:00:00:00
Reciever IP: 192.168.253.142
****************************
No: 14 Sat Mar 18 04:19:29 2017
n_read: 42
buffer:00 50 56 fc 06 ab 00 0c 29 aa 2d 5d 08 06 00 01 08 00 06 04 00 02 00
 0c 29 aa 2d 5d c0 a8 fd 8e 00 50 56 fc 06 ab c0 a8 fd 02
MAC address: 00:0c:29:aa:2d:5d ==> 00:50:56:fc:06:ab
Protocol (in frame):ARP
ARP respons
Sender MAC: 00:0c:29:aa:2d:5d
Sender IP: 192.168.253.142
Reciever MAC: 00:50:56:fc:06:ab
Reciever IP: 192.168.253.2
****************************
```



7	15.927005088	Vmware_fc:06:ab	Broadcast	ARP	60 Who has 192.168.25
8	15.927035465	Vmware_aa:2d:5d	Vmware_fc:06:ab	ARP	42 192.168.253.142 is

两个结果对比可知我的抓包结果是正确的。
用浏览器浏览网页时抓包：

```
****************************
No: 2280 Sat Mar 25 14:19:11 2017
n_read: 54
buffer:00 50 56 fc 06 ab 00 0c 29 aa 2d 5d 08 00 45 00 00 28 f6 e0 40 00 40
 cf c0 a8 fd 8e de c7 bf 20 e6 2a 01 bb d7 5e 97 a0 2d 40 70 f2 50 10 9d 08
 00 00
MAC address: 00:0c:29:aa:2d:5d ==> 00:50:56:fc:06:ab
Protocol (in frame):IPv4
IP: 192.168.253.142 ==> 222.199.191.32
Protocol:tcp
****************************
No: 2281 Sat Mar 25 14:19:11 2017
n_read: 60
buffer:00 0c 29 aa 2d 5d 00 50 56 fc 06 ab 08 00 45 00 00 28 11 fd 00 00 80
 b2 de c7 bf 21 c0 a8 fd 8e 01 bb 89 d0 6e 77 15 22 99 5f 0f 1a 50 19 fa ef
 00 00 00 00 00 00 00 00
MAC address: 00:50:56:fc:06:ab ==> 00:0c:29:aa:2d:5d
Protocol (in frame):IPv4
IP: 222.199.191.33 ==> 192.168.253.142
Protocol:tcp
****************************
No: 2282 Sat Mar 25 14:19:11 2017
n_read: 54
buffer:00 50 56 fc 06 ab 00 0c 29 aa 2d 5d 08 00 45 00 00 28 98 bc 40 00 40
 f3 c0 a8 fd 8e de c7 bf 21 89 d0 01 bb 99 5f 0f 1a 6e 77 15 23 50 10 9d 08
 00 00
MAC address: 00:0c:29:aa:2d:5d ==> 00:50:56:fc:06:ab
Protocol (in frame):IPv4
IP: 192.168.253.142 ==> 222.199.191.33
Protocol:tcp
```

用浏览器访问百度，一瞬间会抓到非常多的包。。。

（2）发送接收 icmp 数据包实验

```
qiuzihao@qiuzihao-virtual-machine:~/computer_network$ sudo ./icmp www.baidu.com
Sent ICMP echo request to 119.75.218.70
Received ICMP echo reply from 119.75.218.70: seq=0, time=33.457 ms
Sent ICMP echo request to 119.75.218.70
Received ICMP echo reply from 119.75.218.70: seq=1, time=30.164 ms
Sent ICMP echo request to 119.75.218.70
Received ICMP echo reply from 119.75.218.70: seq=2, time=35.215 ms
Sent ICMP echo request to 119.75.218.70
Received ICMP echo reply from 119.75.218.70: seq=3, time=34.353 ms
^C
qiuzihao@qiuzihao-virtual-machine:~/computer_network$ sudo ./icmp 119.75.218.70
Sent ICMP echo request to 119.75.218.70
Received ICMP echo reply from 119.75.218.70: seq=0, time=32.013 ms
Sent ICMP echo request to 119.75.218.70
Received ICMP echo reply from 119.75.218.70: seq=1, time=30.180 ms
Sent ICMP echo request to 119.75.218.70
Received ICMP echo reply from 119.75.218.70: seq=2, time=30.094 ms
Sent ICMP echo request to 119.75.218.70
Received ICMP echo reply from 119.75.218.70: seq=3, time=36.841 ms
```

系统的 ping：

```
qiuzihao@qiuzihao-virtual-machine:~/computer_network$ ping www.baidu.com
PING www.baidu.com (119.75.218.70) 56(84) bytes of data.
64 bytes from 119.75.218.70: icmp_seq=1 ttl=128 time=79.4 ms
64 bytes from 119.75.218.70: icmp_seq=2 ttl=128 time=31.9 ms
64 bytes from 119.75.218.70: icmp_seq=3 ttl=128 time=38.6 ms
64 bytes from 119.75.218.70: icmp_seq=4 ttl=128 time=33.7 ms
^C
--- www.baidu.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 31.929/45.943/79.423/19.484 ms
```

系统的 ping 在记录数据上比我的程序多了 ttl，然后 ping 程序结束时会对所有包进行统计，统计有多少包已接受，多少包丢失。这点是我应该学习的。

六、对比样例程序

在我的代码中参考样例程序的部分：
我主要学习了样例程序中对以太网帧以及 ip 包中各字段抽取方式，这种方式的有点我觉得是方便灵活，代码的理解也比较容易，如果要分析的字段比较少的话，我觉得这是种不错的方式。

```
        printf("n_read: %d\n", n_read);
        printf("buffer:");
        unsigned char* buf=buffer;
        for (int i=0;i<n_read;i++){
            printf("%02x ", *buf);
            buf++;
        }
        printf("\n");

        p=eth_head;
        printf("MAC address: %.2x:%02x:%02x:%02x:%02x:%02x ==> %.2x:%02x:%02
x:%02x:%02x:%02x\n",
            p[6],p[7],p[8],p[9],p[10],p[11],
            p[0],p[1],p[2],p[3],p[4],p[5]);
```

```
    ip_head=eth_head+14;
    p=ip_head+12;
    printf("IP: %d.%d.%d.%d ==> %d.%d.%d.%d\n",
        p[0],p[1],p[2],p[3],p[4],p[5],p[6],p[7]);
    proto=(ip_head+9)[0];
    p=ip_head+12;
    printf("Protocol:");
    switch(proto){
        case IPPROTO_ICMP:printf("icmp\n");break;
        case IPPROTO_IGMP:printf("igmp\n");break;
        case IPPROTO_IPIP:printf("ipip\n");break;
        case IPPROTO_TCP:printf("tcp\n");break;
        case IPPROTO_UDP:printf("udp\n");break;
        default:printf("Pls query yourself about: 0x%x\n", proto);
    }
```

七、代码的个人创新及思考

（1）我的创新

i）如果要添加对多种协议的解析支持，那么如果还采用移动指针解析的方式，那么代码就会变得比较臃肿，可维护性也会变差，所以我这里采用的方式是先定义 ARP/RARP 结构体，在将这种结构体的指针指向数据的首地址，这样通过引用结构体中的成员便可获得所有的字段。

```
struct ARP
{
    unsigned short hw_type;
    unsigned short proto_type;
    unsigned char hw_addr_len;
    unsigned char proto_addr_len;
    unsigned short op_code;
    unsigned char sender_mac[6];
    unsigned char sender_ip[4];
    unsigned char receiver_mac[6];
    unsigned char receiver_ip[4];
};
```

```
struct ARP* arp=eth_head+14;
```

ii）ARP 和 RARP 报文的主要差别仅在于操作码的不同，操作码 1,2 表示的是 ARP，3,4 表示的是 RARP，所以对这两个协议的分析可以合在一起。

```
   case 1: printf("ARP request\n"); break;
   case 2: printf("ARP respons\n"); break;
   case 3: printf("RARP request\n"); break;
   case 4: printf("RARP respons\n"); break;
```

iii）在终端下的抓包输出模仿 wireshark 的输出，尽可能输出多的信息供开发人员参考。

```
****************************
No: 258 Sat Mar 18 08:31:20 2017
n_read: 60
buffer:00 0c 29 aa 2d 5d 00 50 56 fc 06 ab 08 00 45 00 00 28 c7 90 00 00 80 06
 52 d8 75 12 ed 1d c0 a8 fd 8e 00 50 d2 7a 53 9d 4c 24 ce 35 fc 15 50 19 fa ef
 57 9c 00 00 00 00 00 00 00 00
MAC address: 00:50:56:fc:06:ab ==> 00:0c:29:aa:2d:5d
Protocol (in frame):IPv4
IP: 117.18.237.29 ==> 192.168.253.142
Protocol:tcp
****************************
No: 259 Sat Mar 18 08:31:20 2017
n_read: 54
buffer:00 50 56 fc 06 ab 00 0c 29 aa 2d 5d 08 00 45 00 00 28 19 0f 40 00 40 06
 01 5a c0 a8 fd 8e 75 12 ed 1d d2 7a 00 50 ce 35 fc 15 53 9d 4c 25 50 10 82 98
 cf fb 00 00
MAC address: 00:0c:29:aa:2d:5d ==> 00:50:56:fc:06:ab
Protocol (in frame):IPv4
IP: 192.168.253.142 ==> 117.18.237.29
Protocol:tcp
****************************
```

iv）在 icmp 包的接受和发送程序中，为了可以添加对 ping 主机名的支持，我参考了 github 上的代码，使用了 linux 的 getaddrinfo()这个 api，从而实现了可以 ping 主机名。

```
qiuzihao@qiuzihao-virtual-machine:~/computer_network$ sudo ./icmp cs.nju.edu.cn
Sent ICMP echo request to 202.119.32.7
Received ICMP echo reply from 202.119.32.7: seq=0, time=0.958 ms
Sent ICMP echo request to 202.119.32.7
Received ICMP echo reply from 202.119.32.7: seq=1, time=1.139 ms
```

（2）我的思考

i）在我学习 getaddrinfo()这个函数的用法时，我觉得这个函数设计得很精巧。

首先这个函数可以有多个返回对象：既可以返回反应操作结果的数，亦可以返回作为地址信息的 addrinfo 链表，这个设计表现了 c 语言强大的表达能力；

其次可以通过传入 addrinfo_hint 这个结构体来指定想要的结果的类型，这启发我在设计函数时，如果有很多的参数需要传入，不妨把这些参数集合为一个结构体，这样既提高了程序的可读性，也降低了编程的难度。

ii）我觉得计算机网络上的各种协议虽然纷繁复杂，但基本上都是为了处理计算机之间交流这一问题。所以如果将协议当做人类交流方式来理解，就会变得比较容易。

八、参考资料

1.getaddrinfo()函数详解
http://www.cnblogs.com/cxz2009/archive/2010/11/19/1881693.html
2. icmp 协议　ping 编程实现
http://blog.sina.com.cn/s/blog_5d0d91ba0100v8j1.html
3. https://github.com/sryze/ping/tree/master/src