计算机网络实验报告

实验二

RAW SOCKET 编程与以太网帧分析基础

学号：1412200065

姓名：刘博

时间：2016.3.20

1.实验目的：

1.熟悉 Linux 环境下基本的 raw socket 编程；
2.对以太网帧进行初步分析；
3.对数据报进行修改和发送；

2.网络拓扑配置：

节点名	虚拟设备名	IP	子网掩码	虚拟交换机
Router	U-572	eth0 192.168.0.1	255.255.255.0	Vnet2
		eth1 192.168.1.1	255.255.255.0	Vnet3
PC 1	U-574	eth0 192.168.0.2	255.255.255.0	Vnet2
PC 2	U-575	eth0 192.168.1.2	255.255.255.0	Vnet3

3.数据结构说明：

1.recv：（接受 raw socket 数据报并进行分析）
在 socket_head.h 头文件中定义了以下三种数据结构：

A． Ethernet（以太网帧）结构：

```
struct Ethernet {
    unsigned char preamble[7];
    unsigned char SFD;      //Start Frame Delimiter
    unsigned char DA[6];        //Destination Address
    unsigned char SA[6];        //Source Address
    unsigned short type_length;         //Type/Length
    char buffer[BUFFER_MAX];        //MAC Client Data
    unsigned int FCS;       //Frame Check Sequence
};
```

其中各项结构含义如下：

1. preamble[7]（前同步码）：由 7 个 8 位组的 0、1 交替序列组成，接收方用来建立同步比特；
2. SFD（帧开始界定符）：序列 10101011，指明帧真正开始，使接收方找出第一个比特的位置；
3. DA[6]（目的地址）：指定帧要去的地址。可能是唯一物理地址，或者是多播地址，或者是广播地址；
4. SA[6]（源地址）：给出传输该帧的站点的地址；
5. Type_length（长度/类型）：依据其数值大小，取两种含义之一。如果这个字段的值小于等于 1500，那么这个长度/类型字段指的是之后的 Mac 客户数据段字段内所包含的 Mac

客户数据八位组的数量（长度解释）。如果这个值大于等于 1536，那么这个长度/类型字段指的就是 Mac 客户协议本身（类型解释）。长度和类型这两种解释是互斥的，不是长度就是类型；

6. Buffer[BUFFER_MAX]（Mac 客户数据）：LLC 提交的数据单元。这个字段的最大长度为：基础帧 1500 个八位组，Q 标志帧 1504 个八位组，包装帧 1982 个八位组；
7. FCS（帧检测序列）：32 比特的循环冗余检验码，对除前同步码，SFD 和 FCS 以外的所有字段做检验。

B．IP 结构：

```
struct IP {
    unsigned char IHL : 4;          //Internet Head Length
    unsigned char version : 4;      //IP Version
    unsigned char ECN : 2;
    unsigned char DS : 6;           //Determine Service
    unsigned short length;      //Length of datagram
    unsigned short label;
    unsigned short offset : 13;
    unsigned char tag : 3;
    unsigned char live;
    unsigned char proto;
    unsigned short check_sum;
    unsigned char SA[4];        //Source Address
    unsigned char DA[4];        //Destination Address
};
```

其中各项结构含义如下：

1.IHL（互联网首部长度）：以 32 比特字为单位的首部长度。最小的值为 5，也就是说最小的首部长度为 20 个八位组；

2.version（版本）：指示的是版本号，使这个协议可以不断发展。其值是 4。

3.ECN：显示阻塞通知字段，在 RFC3168 中定义，它使路由器能够向端节点指出正在经历阻塞的分组，而不是立即丢弃此类分组。值 00 指的是分组没有使用 ECN。值 01 或 10 是由数据发送方设置的，指出运输协议的端点有 ECN 的能力。值 11 是由路由器设置的，表示已经遇到阻塞了；

4.DS：这个字段支持区分服务功能；

5.length（总长度）：数据报的总长度，包括首部和数据，以八位组为单位；

6.label（标识）：一个序号，它与源地址，目的地址以及用户协议结合起来使用，以便唯一地标识出一个数据报。因此，当数据报还存在于互联网中时，这个标识符在具有相同源地址，目的地址以及用户协议的数据报中是唯一的；

7.offset（数据报片偏移量）：指出这个数据报片在源数据报中的位置，以 64 比特为测量单位，其言外之意就是除了最后一个数据报片之外，所有的数据报片包含的数据字段长度都是 64 比特的倍数；

8.tag（标志）：目前只定义了两个比特。“后续”比特用于数据的分片和重装，如上所述。

9.live（生存时间）：规定一个数据报可以在互联网中存留多久，以秒为单位。每个处理数据报的路由器都必须减少生存时间（TTL）的值，减少量至少为 1，因此从某种程度上来看，它类似于跳数计数；

10.proto（协议）：指出目的系统中接受数据字段的上层协议。因此，这个字段指出了分组中 IP 首部之后的下一个首部的类型，比如值为 TCP=6，UDP=17；
11.check_sum（首部检验和）：仅仅对首部起作用的差错检验码。由于某些首部字段在传输途中会改变，所以需要在每个路由器上进行验证以及重新计算。检验和字段是首部所有 16 比特字的 16 比特二进制反码加法。为了方便计算，检验和自身的初始值设为 0；
12.SA[4]（源地址）：这个编码允许各种各样的比特配置，以便指明与某个网络相连的网络或端系统；
13.DA[4]（目的地址）：与源地址性质相同。

C．ARP 结构：

```
struct ARP {
    unsigned short Htype;      //Hardware Type
    unsigned short Ptype;      //Protocal Type
    unsigned char Mac_Length;
    unsigned char IP_Length;
    unsigned short OP;     //Operation Code
    unsigned char sender_Mac[6];
    unsigned char sender_IP[4];
    unsigned char recver_Mac[6];
    unsigned char recver_IP[4];
    char data[18];
};
```

其中各项结构含义如下：
1.Htype（硬件类型）：指明了发送方想知道的硬件接口类型，以太网的值为 1；
2.Ptype（协议类型）：指明了发送方提供的高层协议类型，IP 为 0800（16 进制）；
3.Mac_Length（硬件地址长度）：指明了硬件地址的长度。
4.IP_Length（协议长度）：指明了高层协议地址的长度，这样 ARP 报文就可以在任意硬件和任意协议的网络中使用
5.OP（操作类型）：用来表示这个报文的类型，ARP 请求为 1，ARP 响应为 2，RARP 请求为 3，RARP 响应为 4；
6.sender_Mac[6]（发送方硬件地址）：源主机硬件地址；
7.sender_IP[6]（发送方 IP 地址）：源主机的 IP 地址；
8.recver_Mac[6]（目标硬件地址）：目的主机硬件地址；
9.recver_IP[4]（目标 IP 地址）：目的主机的 IP 地址。

2.send：（发送 raw socket 数据包并输出显示）
在 socket_head.h 文件中定义了以下数据结构：

A．ICMP 结构：

```
struct ICMP {
    unsigned char type;
    unsigned char code;
    unsigned short check_sum;
    unsigned short id;
```

```
    unsigned short sequence;
    unsigned long timestamp;
};
```

1.type（类型）：定义了 ICMP 报文的类型；
2.code（编码）：用于定义这个报文的一些参数，这些参数可以被编码成一个或几个比特；
3.check_sum（检验和）：整个 ICMP 报文的检验和。与 IP 中使用的检验和算法相同；
4.id（程序号）：记录发送进程的 id；
5.sequence（序号）：用来记录分组的序号，每发送一个分组，其值加 1，ping 中显示该值以用来检查是否发生了丢包；
6.timestamp（时间戳）：用来记录分组发送的时间，继而可以计算出分组传输的时延。

4.程序设计的思路以及运行流程：

1.抓包程序：

思路：首先创建一个 socket 描述符，利用 socket 函数：

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

其中 socket_fd 是 socket 描述符，socket 函数参数含义如下：

A. PF_PACKET：表示接收数据的地址类型，PF_PACKET 表示数据报（包括 ARP 数据报和 IP 数据报）。
B. SOCK_RAW：表示 socket 的类型，我们这里使用的的 socket_raw 类型，可以直接接收到链路层的原始帧（以太网帧）。
C. ETH_P_ALL：制定接受的协议类型，ETH_P_ALL 表示接受所有的协议类型（如果是 ETH_P_IP 只会接受 IP 协议的数据报）。

然后根据这个 socket 描述符进行接受，使用 recvfrom 函数将数据帧全部写入 buffer 字符串中：

```
int n_read = recvfrom(socket_fd,buffer,2048,0,NULL,NULL);
```

将 buffer 数组中的数据分别用结构类型加以解释，首先将其解释为以太网帧的结构：

```
struct Ethernet *ethernet_frame1 = (void *)(buffer - 8);
    printf("Ethernet Frame: \n\t");
```

（注意这里不能讲buffer直接拷贝到结构的头部，这是因为在socket抓包时将前同步码和帧开始界定符已经略去，所以要省去这8个字节）

然后通过结构输出以太网帧中的 Mac 地址：

```
printf("Destination Mac
Address : %02x:%02x:%02x:%02x:%02x:%02x\n\t",ethernet_frame1->DA[0],ethernet_frame1
```

```
->DA[1],ethernet_frame1->DA[2],ethernet_frame1->DA[3],ethernet_frame1->DA[4],ethern
et_frame1->DA[5]);
printf("Source                                                    Mac
Address : %02x:%02x:%02x:%02x:%02x:%02x\n\n",ethernet_frame1->SA[0],ethernet_frame1
->SA[1],ethernet_frame1->SA[2],ethernet_frame1->SA[3],ethernet_frame1->SA[4],ethern
et_frame1->SA[5]);
```

然后对以太网帧中的类型成员进行分析，根据其值的不同，得知后面数据报的协议类型：

```
if(ntohs(ethernet_frame1->type_length) == 0x0800)
{
        printf("IP datagram : \n\t");
        analyse_IP(ethernet_frame1->buffer);
        printf("\n");
}    (0x0800表示IP数据报)
else if(ntohs(ethernet_frame1->type_length) == 0x0806)
{
        printf("ARP packet : \n\t");
        analyse_ARP(ethernet_frame1->buffer);
        printf("\n");
}    (0x0806表示ARP包)
else if(ntohs(ethernet_frame1->type_length) == 0x8035)
{
        printf("RARP packet : \n\t");
        analyse_RARP(ethernet_frame1->buffer);
        printf("\n");
}    (0x8035表示RARP包)
else
{
        printf("add yourself query\n");
}    (未完待续)
```

这里通过调用已经封装好的三个 analyse 函数进行分析和输出。

Anlyse 函数：

Analyse 函数是我自己封装的三个函数，定义在 analyse.c 中：

1. IP 分析：
   a) IP 分析函数代码如下：

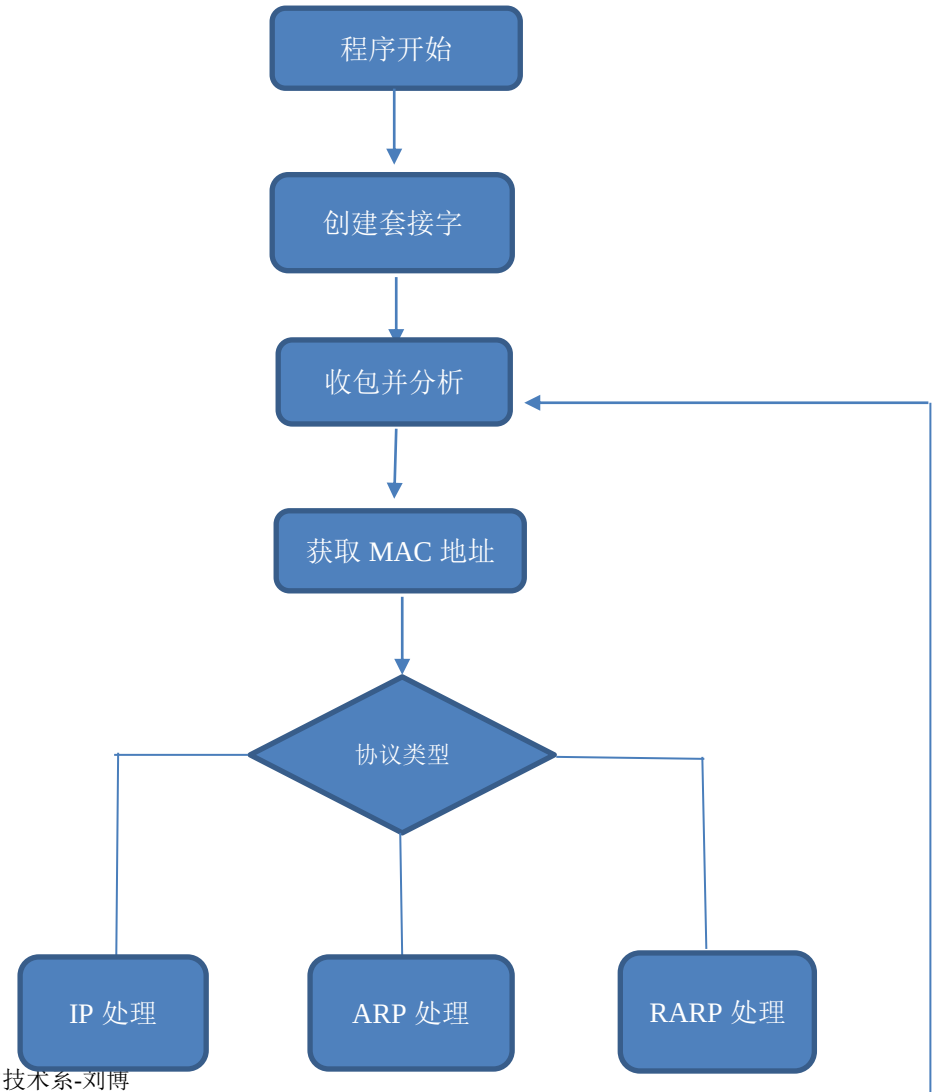
```
void analyse_IP(char *IP_buffer)
{
    struct IP *ip_datagram1 = (void *)IP_buffer;
    printf("IP Version : %d\t",ip_datagram1->version);
    printf("IP Internet Head Length : %d\n\t",ip_datagram1->IHL);
    printf("IP Source
Address : %d.%d.%d.%d\n\t",ip_datagram1->SA[0],ip_datagram1->SA[1],ip_datagram1->SA[2],
```

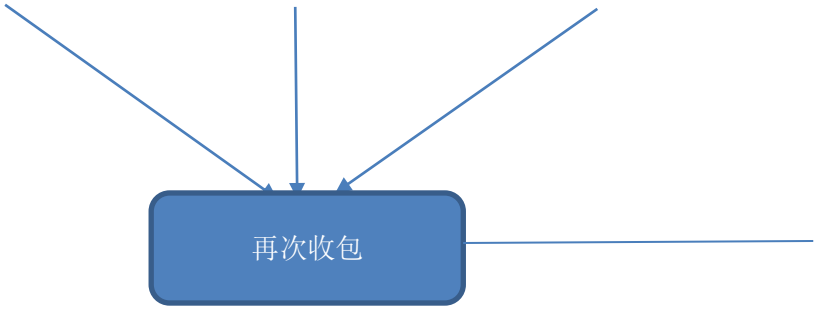
```
ip_datagram1->SA[3]);
                    printf("IP Destination
Address : %d.%d.%d.%d\n\t",ip_datagram1->DA[0],ip_datagram1->DA[1],ip_datagram1->DA[2],
ip_datagram1->DA[3]);
                    printf("IP Protocal : \t");
                    switch(ip_datagram1->proto)
                    {
                        case IPPROTO_ICMP:printf("ICMP\n\t");break;
                        case IPPROTO_IGMP:printf("IGMP\n\t");break;
                        case IPPROTO_IPIP:printf("IPIP\n\t");break;
                        case IPPROTO_TCP:printf("TCP\n\t");break;
                        case IPPROTO_UDP:printf("UDP\n\t");break;
                        default : printf("Need to add query\n\t");
                    }
                    printf("IP Lives : %d(s)\n",ip_datagram1->live);
                    return;
                }
```

根据 IP 结构定义将其中的每一个成员值输出即可。

2. ARP 与 RARP 分析同理，这里不进行赘述；

至此抓包程序处理结束。整个流程图如下：





2.发包程序：

思路：首先根据参数判断发包的类型（ICMP 或 ARP）；

A． ICMP 包的发送：

首先建立 ICMP 数据报的描述符：

```
if((socket_fd = socket(AF_INET,SOCK_RAW,IPPROTO_ICMP)) < 0)
{
    printf("ICMP : Error create raw socket!\n");
    return -1;
}
```

AF_INET：表示发送 IP 协议类型的数据报；

IPPROTO_ICMP：表示使用 ICMP 协议。

然后新建 sockaddr_in（由于使用 AF_INET）结构，用来表示 socket 结构地址信息：

```
struct sockaddr_in ip_addr;
bzero(&ip_addr,sizeof(struct sockaddr_in));
if(inet_aton(argv[1],&ip_addr.sin_addr) == 0)
{
    printf("Error IP target!\n");
    return -1;
}
ip_addr.sin_family = AF_INET;
ip_addr.sin_port = 0;
```

其中 bzero 函数用来将 sockaddr 结构所有值清零。然后将 main 函数的第一个参数赋值给 sockaddr 中的 sin_addr 成员（即将 IP 地址写入 sockaddr 结构中），将 sin_family 成员赋值为 AF_INET，sin_port 设置为 0（表示可以使用任意端口，由 socket 函数自行安排）。

定义好了 sockaddr_in 结构后，我们就需要自己编写一个 ICMP 数据包：

```
bzero(&buffer[sizeof(struct ICMP)],BUFFER_SEND - sizeof(struct ICMP));
struct ICMP *icmp = (void *)buffer;
icmp->type = ICMP_ECHO;
icmp->code = 0;
icmp->id = getpid();
icmp->sequence = htons(count++);
```

```
icmp->check_sum = checksum((void *)icmp,sizeof(struct ICMP) + 32);
icmp->timestamp = 0;
```

首先将 buffer 数组中 ICMP 位置之后的数据清零。

然后将 ICMP 结构中所有成员进行赋值：

1. type 为 echo，表示这是请求报文；
2. code 为 0；
3. id 为当前进程 id；
4. sequence 为循环次数的值，每发送一次便加 1；
5. check_sum：调用 check_sum 函数，计算其检验和；
6. timestamp：将时间戳的值记为 0（为了简略）。

定义好 ICMP 数据包后，通过 sendto 函数进行发送：

```
sendto(socket_fd,buffer,sizeof(struct ICMP) + 32,0,(struct sockaddr*)&ip_addr,sizeof(struct sockaddr_in));
```

然后通过 printf 进行显示输出：

```
printf("%d bytes from %d.%d.%d.%d: icmp_seq = %d\n",sizeof(struct ICMP) + 32,atoi(&argv[1][0]),atoi(&argv[1][4]),atoi(&argv[1][8]),atoi(&argv[1][11]),count);
```

最后调用 sleep 函数进行发包时间控制，使其每隔 1 秒发一次包（否则发包速度太快无法看清）。

B．ARP 包的发送：

发送 ARP 包过程思路与发送 ICMP 包基本相似，不同之处在于 ARP 包需要自己编写整个以太网帧结构，代码如下：

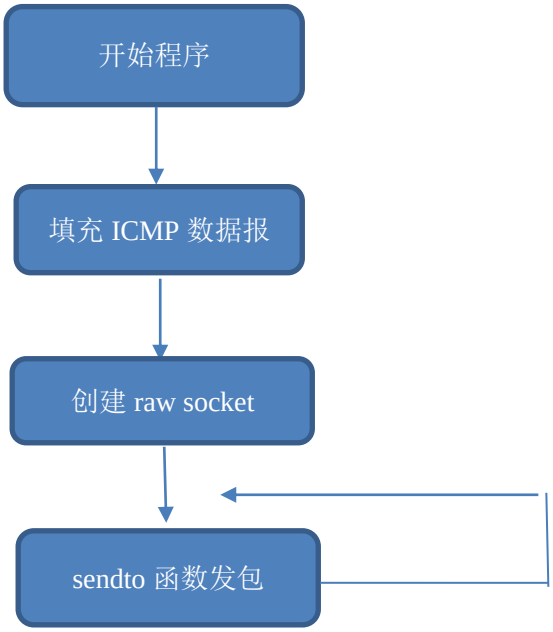
```
if((socket_fd = socket(PF_PACKET,SOCK_RAW,ETH_P_ARP)) < 0)
{
    printf("ARP : Error create raw socket!\n");
    return -1;
}
struct sockaddr_ll arp_addr,broad_addr;
bzero(&arp_addr,sizeof(struct sockaddr_ll));
bzero(&broad_addr,sizeof(struct sockaddr_ll));
arp_addr.sll_family = AF_PACKET;
arp_addr.sll_protocol = htons(ETH_P_ARP);
arp_addr.sll_ifindex = 2;
arp_addr.sll_hatype = ARPHRD_ETHER;
arp_addr.sll_pkttype = PACKET_HOST;
arp_addr.sll_halen = ETH_ALEN;

broad_addr.sll_family = AF_PACKET;
broad_addr.sll_protocol = htons(ETH_P_ARP);
broad_addr.sll_ifindex = 2;
broad_addr.sll_hatype = ARPHRD_ETHER;
```

```
broad_addr.sll_pkttype = PACKET_BROADCAST;
broad_addr.sll_halen = ETH_ALEN;
int i;
for(i = 0; i < 8; i++)
    broad_addr.sll_addr[i] = 'f';
bind(socket_fd,(struct sockaddr *)&arp_addr,sizeof(struct sockaddr_ll));
while(1)
{
    bzero(&buffer[sizeof(struct ARP)],BUFFER_SEND - sizeof(struct ARP));
    sendto(socket_fd,buffer,sizeof(struct ARP),0,(struct sockaddr
    *)&broad_addr,sizeof(struct sockaddr_ll));
    sleep(1);
}
```

首先建立 sockaddr_ll 结构（用来表示以太网帧地址），然后设置两个地址（由于 ARP 包是用来进行 ARP 表更新的，所以需要进行广播地址的发送），并且将其中的参数分别进行初始化。最后通过 sendto 函数发送即可。
（注意 ARP 包发送时需要进行 bind 函数绑定）

至此发包程序结束。其中 check_sum 函数定义在 build_packet.c 中，这里不进行解释，只是通过简单的取反加和操作即可。整个过程流程图如下：



5.运行结果截图：

实验从 PC1（192.168.0.2）向 PC2（192.168.1.2）发送 ICMP 数据报，截图如下：

PC1：（send）

```
root@ubuntu: /home/kirito/Desktop/socket/send
root@ubuntu:/home/kirito/Desktop/socket/send# ./socket 192.168.1.2 ICMP
44 bytes from 192.168.1.0: icmp_seq = 1
44 bytes from 192.168.1.0: icmp_seq = 2
44 bytes from 192.168.1.0: icmp_seq = 3
44 bytes from 192.168.1.0: icmp_seq = 4
44 bytes from 192.168.1.0: icmp_seq = 5
44 bytes from 192.168.1.0: icmp_seq = 6
44 bytes from 192.168.1.0: icmp_seq = 7
44 bytes from 192.168.1.0: icmp_seq = 8
44 bytes from 192.168.1.0: icmp_seq = 9
44 bytes from 192.168.1.0: icmp_seq = 10
44 bytes from 192.168.1.0: icmp_seq = 11
44 bytes from 192.168.1.0: icmp_seq = 12
44 bytes from 192.168.1.0: icmp_seq = 13
44 bytes from 192.168.1.0: icmp_seq = 14
44 bytes from 192.168.1.0: icmp_seq = 15
44 bytes from 192.168.1.0: icmp_seq = 16
44 bytes from 192.168.1.0: icmp_seq = 17
44 bytes from 192.168.1.0: icmp_seq = 18
44 bytes from 192.168.1.0: icmp_seq = 19
44 bytes from 192.168.1.0: icmp_seq = 20
44 bytes from 192.168.1.0: icmp_seq = 21
44 bytes from 192.168.1.0: icmp_seq = 22
44 bytes from 192.168.1.0: icmp_seq = 23
```

（其中 IP 192.168.1.0 是 router 0 的 IP 地址，icmp_seq 为序列号，连续证明其没有丢包）

PC2：（recv）

```
root@ubuntu: /home/kirito/Desktop/socket/recv
root@ubuntu:/home/kirito/Desktop/socket/recv# ./socket
Ethernet Frame:
        Destination Mac Address : 00:0c:29:6b:19:62
        Source Mac Address : 00:0c:29:25:5c:27

IP datagram :
        IP Version : 4  IP Internet Head Length : 5
        IP Source Address : 192.168.0.2
        IP Destination Address : 192.168.1.2
        IP Protocal :   ICMP
        IP Lives : 63(s)

Ethernet Frame:
        Destination Mac Address : 00:0c:29:6b:19:62
        Source Mac Address : 00:0c:29:25:5c:27

IP datagram :
        IP Version : 4  IP Internet Head Length : 5
        IP Source Address : 192.168.0.2
        IP Destination Address : 192.168.1.2
        IP Protocal :   ICMP
        IP Lives : 63(s)

Ethernet Frame:
```

可以看出 IP 数据报中的各种参数值，发送方为 PC1（192.168.0.2），接收方为 PC2（192.168.1.2），PC1 的 Mac 为 00:0c:29:25:5c:27，PC2 的 Mac 为 00:0c:29:6b:19:62，数据报版本为 IPV4，网络首部长度为 5bytes，协议为 ICMP，TTL 为 63s。

6.本次实验的创新与思考：

1.本次实验难点在于 socket 编程是第一次接触，所以对于其使用不是很熟悉，再加上对 IP 及 Ethernet 的数据包格式不熟，所以有了很大的困难，经过各种查阅书籍、资料，最终完成了对于数据报的封装，发送，接受，这是本次实验最大的收获。
2.在发送数据报时，要使用 sockaddr_in 和 sockaddr_ll 数据结构进行封装，而且在每次发送完一个数据报后要有一定的延迟时间（sleep 函数），否则会发送的太快。
3.使用时可以根据后面的参数决定是否发送 ARP 包。
4.通过这次学习，也然我对书籍内容有了一定了解，特此感谢百度，谷歌，以及《计算机网络（自顶向下的方法）》和《数据与计算机通信（第九版）》这些资料。