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：

1. 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。
8. 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。

1. 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](http://baike.baidu.com/view/159864.htm)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](http://baike.baidu.com/subview/36190/12517929.htm)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：

1. 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）。
2. 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）。
3. 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],ethernet\_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],ethernet\_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：
         1. 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。

* + - 1. 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：

程序开始

创建套接字

收包并分析

获取MAC地址

协议类型

RARP处理

ARP处理

IP处理

再次收包

2.发包程序：

思路：首先根据参数判断发包的类型（ICMP或ARP）；

1. 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）。

1. 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：

开始程序

填充ICMP数据报

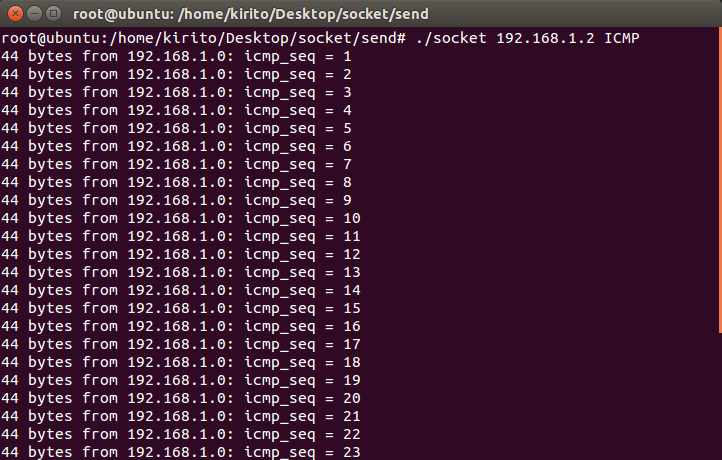
创建raw socket

sendto函数发包

**5.运行结果截图：**

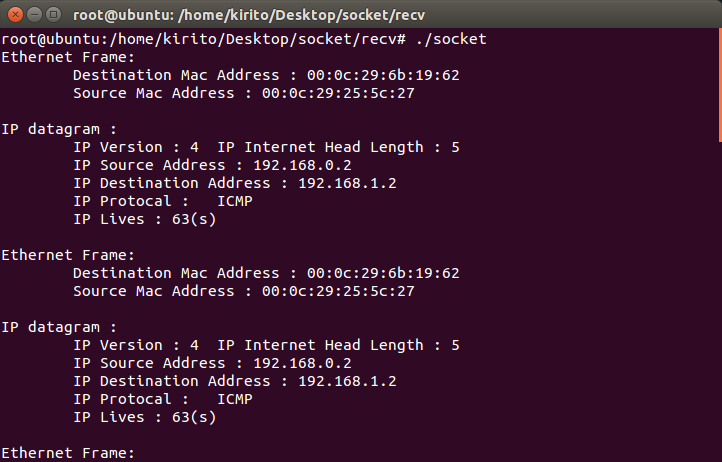
实验从PC1（192.168.0.2）向PC2（192.168.1.2）发送ICMP数据报，截图如下：

PC1：（send）



（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）



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。