第三次实验报告

通过编程获取IP地址与MAC地址的对应关系

郭裕彬 2114052 物联网工程

实验要求

（1）在IP数据报捕获与分析编程实验的基础上，学习NPcap的数据包发送方法。

（2）通过NPcap编程，获取IP地址与MAC地址的映射关系。

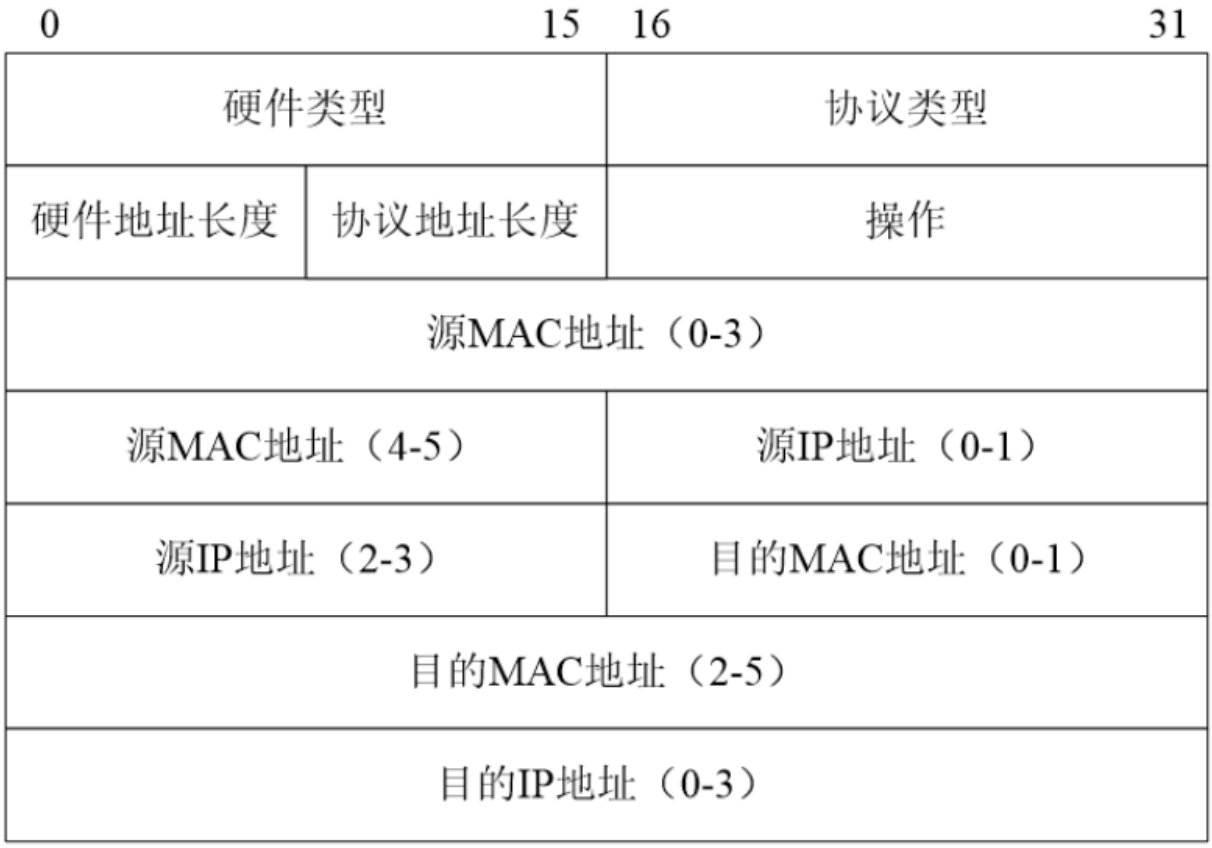
（3）程序要具有输入IP地址，显示输入IP地址与获取的MAC地址对应关系界面。界面可以是命令行界面，也可以是图形界面，但应以简单明了的方式在屏幕上显示。

（4）编写的程序应结构清晰，具有较好的可读性。

协议内容

设备获取同一个子网内的IP地址与MAC地址对应关系主要通过向其他设备发送ARP请求包，获取ARP响应包解包实现。

ARP报文的格式



- 硬件类型：以太网接口类型为1
- 协议类型：IP协议类型为0800
- 操作：ARP请求为1，ARP响应为2
- 硬件地址长度：MAC地址长度为6Bytes
- 协议地址长度：IP地址长度为4Bytes
- 源MAC地址：发送方的MAC地址
- 源IP地址：发送方的IP地址
- 目的MAC地址：ARP请求中该字段无意义，ARP响应中为接收方的MAC地址
- 目的IP地址：ARP请求中为请求的IP地址，ARP响应中为接收方的IP地址

ARP报文的总长度为28字节

代码实现

```
/* 4 bytes IP address */
typedef struct ip_address {
    u_long ip;
```

```
}ip_address;
#pragma pack(1)
typedef struct ehter_header {
    u_char    ether_dhost[6];    //目的MAC地址
    u_char    ether_shost[6];    //源MAC地址
    u_short   ether_type;        //帧类型
}ether_header;

/* ARP header */
typedef struct ARP_frame {
    ether_header header;     //以太帧首部
    u_short hardware;        //硬件类型
    u_short protocol;        //协议类型
    u_char hardware_size;    //硬件地址长度
    u_char protocol_size;    //协议地址长度
    u_short opcode;          //操作码
    u_char sender_mac[6];    //源MAC地址
    ip_address sender_ip;    //源IP地址
    u_char target_mac[6];    //目的MAC地址
    ip_address target_ip;    //目的IP地址
}ARP_frame;
```

其中，ip_address的结构设计由讲解实验时的u_char byte1~u_char byte4优化为一个u_long类型变量ip。

程序设计

获取设备、设置过滤器、监听设备

这些过程的设计与实验二中基本一致，过滤器静态设置为`ether proto \\arp`，只过滤出ARP包进行后续处理。

```
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING,
        NULL, &alldevs, errbuf) == -1)
    {
        fprintf(stderr, "在检测设备时出现错误: %s\n", errbuf);
        exit(1);
```

```
    }
    //打印列表
    for (d = alldevs; d; d = d->next)
    {
        printf("%d. %s", ++i, d->name);
        if (d->description)
            printf(" (%s)\n", d->description);
        else
            printf(" 无可用设备! \n");
    }
    if (i == 0)  return -1;
    printf("请输入设备编号 (1-%d):", i);
    scanf("%d", &inum);
    if (inum < 1 || inum > i)
    {
        printf("\n超出可用范围\n");
        pcap_freealldevs(alldevs);
        return -1;
    }

    //跳转至想要监听的设备
    for (d = alldevs, i = 0; i < inum - 1; d = d->next, i++);
    //打开设备
    if ((adhandle = pcap_open(d->name, // name of the device
        65536, // portion of the packet to capture
               // 65536 guarantees that the whole packet will
               // be captured on all the link layers
        PCAP_OPENFLAG_PROMISCUOUS, // promiscuous mode
        1000, // read timeout
        NULL, // authentication on the remote machine
        errbuf // error buffer
    )) == NULL)
    {
        fprintf(stderr,
            "\n%s设备不支持! %\n",
            d->name);
        pcap_freealldevs(alldevs);
        return -1;
    }
```

```
    printf("\n正在 %s 上监听...\n", d->description);
    cout << "正在设置过滤条件...限定为ARP......"<<endl;
    char filter[40] = "ether proto \\arp";
    u_int netmask;
    struct bpf_program fcode;
    if (d->addresses != NULL)
        //获取掩码
        netmask = ((struct sockaddr_in*)(d->addresses->netmask))-
>sin_addr.S_un.S_addr;
    else
        //C类设备
        netmask = 0xffffff;
    if (pcap_compile(adhandle, &fcode, filter, 1, netmask) < 0)
    {
        fprintf(stderr,"\n无法解析过滤器\n");
        pcap_freealldevs(alldevs);
        return -1;
    }
    else
    {
        if (pcap_setfilter(adhandle, &fcode) < 0)
        {
            cout << "过滤器发生错误! \n" << endl;
            return -1;
        }
        cout << "正在监听" << d->description << endl;
    }
```

获取选中网卡设备的MAC地址

通过调用win32 app位于iphlpapi.h的GetAdapterAddresses函数来进行网卡设备MAC地址的获取。GetAdaptersAddresses和pcap_findalldevs_ex获取到的设备类型和数量顺序不同，正常情况下需要通过设备GUID来定位两处出现的相同设备，因为本次实验只要用到网卡设备，所以这里简化设计为只比较网卡设备的IP地址来确定设备。找到设备后，`pAddresses->PhysicalAddress`数组即存储了对应设备的MAC地址，待用作后续封装ARP包。

```
//获取网卡设备MAC地址
    PIP_ADAPTER_ADDRESSES pAddresses=nullptr;
    ULONG outbuflen=0;
    GetAdaptersAddresses(AF_UNSPEC,0,NULL,pAddresses,&outbuflen);
    pAddresses=(IP_ADAPTER_ADDRESSES*)malloc(outbuflen);
    GetAdaptersAddresses(AF_INET,NULL,NULL,pAddresses,&outbuflen);
    while(1)
    {
        //简化成使用IP地址而不是设备GUID标识找到该设备
        if(strncmp((pAddresses->FirstUnicastAddress-
>Address.lpSockaddr->sa_data+2),(d->addresses->addr-
>sa_data+2),4)==0)
        break;
        pAddresses = pAddresses->Next;
    }
```

封装和发送ARP请求包

ARP请求包的以太网帧头部目的MAC地址为ff:ff:ff:ff:ff:ff的广播地址，源MAC地址为上一步得到的设备MAC地址；ARP报文中各个字段的设置同协议内容部分所述，此外目的MAC地址设置为00:00:00:00:00:00；使用inet_addr将用户输入的IP地址存放成u_long数据类型，用作源IP地址。注释部分为实验讲解时比较繁杂的代码，后续通过修改数据结构和使用函数简化了对应过程。

```
//封装ARP包
    for(int i=0;i<6;i++)
    {
        ARPframe.header.ether_shost[i]=pAddresses-
>PhysicalAddress[i];
        ARPframe.header.ether_dhost[i]=0xff;
        ARPframe.sender_mac[i]=pAddresses->PhysicalAddress[i];
        ARPframe.target_mac[i]=0x00;
    }
    // ARPframe.sender_ip.byte1=d->addresses->addr->sa_data[2];
    // ARPframe.sender_ip.byte2=d->addresses->addr->sa_data[3];
    // ARPframe.sender_ip.byte3=d->addresses->addr->sa_data[4];
    // ARPframe.sender_ip.byte4=d->addresses->addr->sa_data[5];
```

```
    strncpy((char*)&ARPframe.sender_ip.ip,(char*)(d->addresses-
>addr->sa_data+2),4);
    char ipInput[20];
    cout<<"输入IP地址：";
    cin>>ipInput;
    u_char addr=0;
    // for(int i=0,j=0;i<ipInput.length();i++)
    // {
    //     if(ipInput[i]!='.')
    //         addr=addr*10+ipInput[i]-'0';
    //     else
    //     {
    //         switch(j)
    //         {
    //             case 0:ARPframe.target_ip.byte1=addr;break;
    //             case 1:ARPframe.target_ip.byte2=addr;break;
    //             case 2:ARPframe.target_ip.byte3=addr;break;
    //         }
    //         addr=0;
    //         j++;
    //     }
    // }
    //ARPframe.target_ip.byte4=addr;
    char *ipad = ipInput;
    ARPframe.target_ip.ip=inet_addr(ipad);
    ARPframe.header.ether_type=htons(0x0806);
    ARPframe.hardware=htons(0x0001);
    ARPframe.protocol=htons(0x0800);
    ARPframe.hardware_size=6;
    ARPframe.protocol_size=4;
    ARPframe.opcode=htons(0x0001);
    //发送ARP请求
    pcap_sendpacket(adhandle,(u_char*)&ARPframe,sizeof(ARPframe));
```

接收和解析ARP响应包

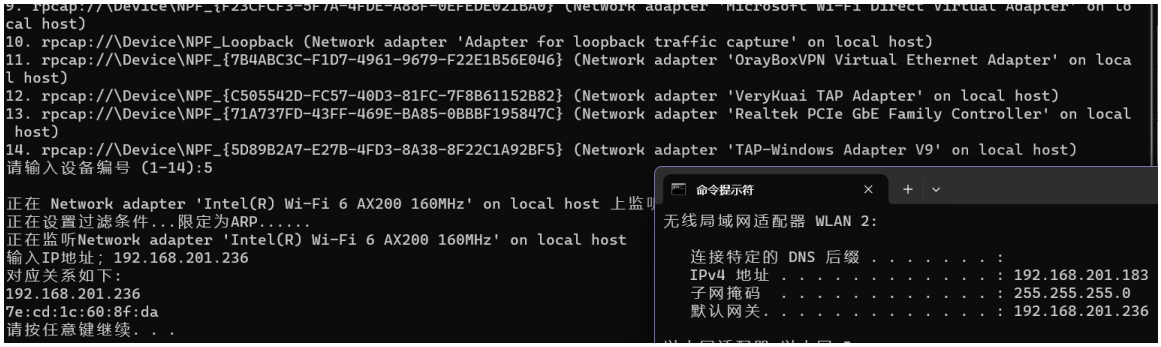
循环捕获ARP响应包直至捕获成功或用户退出。每捕获一个ARP数据包，检测包内的源IP地址是否为请求包中的目的IP地址，目的IP地址是否为请求包中的源IP地址也就是本机，检测全是则解析出包中的源MAC地址打印出来。注释部分为讲解实验时比较繁琐的比较条件，后续将IP地址数据类型变更为u_long，直接比较即可，简化了程序语句。

```
    //循环捕获ARP应答
    while(1)
    {
        switch(pcap_next_ex(adhandle,&pkt_header,&pkt_data))
        {
        case -1:
            cout<<"捕获错误"<<endl;
            return 0;
        case 0:
            cout<<"未捕获到数据报"<<endl;
            break;
        default:
            RecFrame=(ARP_frame*)pkt_data;
            if(
            //     RecFrame-
>target_ip.byte1==ARPframe.sender_ip.byte1
            // &&RecFrame-
>target_ip.byte2==ARPframe.sender_ip.byte2
            // &&RecFrame-
>target_ip.byte3==ARPframe.sender_ip.byte3
            // &&RecFrame-
>target_ip.byte4==ARPframe.sender_ip.byte4
            // &&RecFrame-
>sender_ip.byte1==ARPframe.target_ip.byte1
            // &&RecFrame-
>sender_ip.byte2==ARPframe.target_ip.byte2
            // &&RecFrame-
>sender_ip.byte3==ARPframe.target_ip.byte3
            // &&RecFrame-
>sender_ip.byte4==ARPframe.target_ip.byte4
            (RecFrame->target_ip.ip==ARPframe.sender_ip.ip)&&
(RecFrame->sender_ip.ip==ARPframe.target_ip.ip)
```

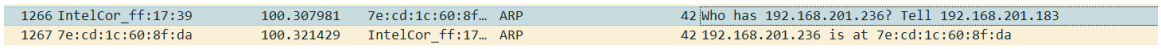
```
            )
            {
                cout<<"对应关系如下:"<<endl;
                printf("%d.%d.%d.%d\n",
(u_char)ARPframe.target_ip.ip,(ARPframe.target_ip.ip & 0x0000FF00)
>> 8,(ARPframe.target_ip.ip & 0x00FF0000) >> 16,
(ARPframe.target_ip.ip & 0xFF000000)>>24);
                printf("%02x:%02x:%02x:%02x:%02x:%02x\n",RecFrame-
>sender_mac[0],RecFrame->sender_mac[1],RecFrame-
>sender_mac[2],RecFrame->sender_mac[3],RecFrame-
>sender_mac[4],RecFrame->sender_mac[5]);
                system("pause");
                return 0;
            }
        }
    }
```

实验结果

- 将设备连接到局域网中，如下图尝试获取网关192.168.201.236的MAC地址，得到的响应包中源MAC地址字段为`7e:cd:1c:60:8f:da`



- 同时使用Wireshark抓包得到的对应arp包详情如下，通过对比可以看出程序运行结果正确无误



仓库链接

代码连接：Network-Technology-and-Application/lab3 at main · shockstove/Network-Technology-and-Application (github.com)