



## 网络技术与应用课程实验报告

### 实验三：通过编程获取 IP 地址与 MAC 地址的 对应关系



学 院 密码与网络空间安全学院  
专 业 信息安全、法学双学位班  
学 号 2313815  
姓 名 段俊宇  
班 级 信息安全、法学双学位班

# 一、实验目的

1. 进一步了解 NPcap 相关函数和使用方法。
2. 在 IP 数据报捕获与分析编程实验的基础上，学习 WinPcap 的数据包发送方法。
3. 通过 WinPcap 编程，获取 IP 地址与 MAC 地址的映射关系。

# 二、实验原理

## 1. WinPcap 的数据包发送方法

由于 NPcap 是基于 WinPcap 的扩展和开发，并且截至目前仍在维护中，因此本次实验继续使用 NPcap 库。

NPcap 提供了 `pcap_sendpacket()` 函数来发送数据包，函数参数如下所示

```
PCAP_AVAILABLE_0_8  
PCAP_API int pcap_sendpacket(pcap_t *, const u_char *, int);
```

第一个参数是已经打开的句柄；第二个参数是要发送的数据包缓冲区；第三个参数是数据包大小。在使用这一函数时，需要获取设备列表，然后打开指定的网卡设备，之后要构造好数据包才可以调用该函数进行发送。数据包发送成功后便可以像上次实验一样捕获然后进行分析，最后就可以得到目标 MAC 地址。

## 2. ARP 协议

ARP (Address Resolution Protocol, 地址解析协议) 是一种用于在局域网中将 IP 地址解析为 MAC 地址的网络协议。当设备需要与同一局域网内的另一台设备通信时，它首先通过广播发送 ARP 请求包，询问“谁拥有这个 IP 地址”；所有设备都会收到请求，但只有目标设备会回复 ARP 响应包，告知自己的 MAC 地址。这样，源设备就能建立 IP 地址到 MAC 地址的映射并存入 ARP 缓存表，实现数据链路层的直接通信。ARP 协议工作在 OSI 模型的网络层与数据链路层之间，是 TCP/IP 协议栈中实现局域网通信的基础协议。

ARP 报文头部结构如下：

字段名称	硬件类型	上层协议类型	MAC 地址长度	IP 地址长度	操作码	源 MAC 地址	源 IP 地址	目的 MAC 地址	目的 IP 地址
大小(字节)	2	2	1	1	2	6	4	6	4

在寻找目标 MAC 时，设备首先将 ARP 包中的目的 MAC 地址设置为 FF-FF-FF-FF-FF-FF，这样就涵盖了所有的设备，也就达到了广播的效果；而硬件类型应当为以太网，也就是 0x0001，；操作码应当为 0x0001，表示请求；协议类型应当为 0x0800，表示 IPv4 协议。

### 三、实验过程

在编程的过程中我定义了以太网帧结构体、ARP 报文结构体和一些函数，用来构造 ARP 数据包并对捕获的数据包进行处理等。接下来我将使用注释的形式进行解释。

#### 1. 以太网帧结构体

```
// 以太网帧头部
struct EthernetHeader
{
    byte dstMAC[6];           // 目标 MAC 地址
    byte srcMAC[6];           // 源 MAC 地址
    WORD ethertype;           // 协议类型
};
```

#### 2. ARP 结构体

```
/ ARP 报文
struct ARP
{
    WORD hardware;           // 硬件类型
    WORD protocol;           // 协议类型
    byte MAC_length;          // MAC 地址长度，单位是字节
    byte IPaddress_length;    // 协议长度
    WORD opcode;              // 操作码
    byte srcMAC[6];           // 源 MAC 地址
    byte srcIP[4];             // 源 IP
```

```
    byte dstMAC[6];           // 目标 MAC 地址
    byte dstIP[4];            // 目标 IP
};
```

### 3. 输出 MAC 地址

```
// 打印 MAC 地址
void print_MAC(const byte MAC[6])
{
    for (int i = 0; i < 6; i++)
    {
        printf("%02X", MAC[i]);
        if (i < 5) cout << "-";
    }
    cout << endl;
}
```

### 4. 打印设备信息

```
// 打印设备信息
int PrintDeviceInfo(pcap_if_t* device_list)
{
    pcap_if_t* d;           // 遍历指针
    int count = 0;          // 可用设备数量
    for (d = device_list; d != nullptr; d = d->next)           // 开始
遍历
    {
        cout << (count + 1) << ".Device:" << d->name << endl; // 设备
序号
        if (d->description)
            cout << "Description:" << d->description << endl; // 设备
描述
        else
            cout << "No description available" << endl;
// 打印 IP 地址和子网掩码
for (pcap_addr_t* a = d->addresses; a != nullptr; a = a->next)
{
    if (a->addr->sa_family == AF_INET)
    {
        char ip[INET_ADDRSTRLEN];
        inet_ntop(AF_INET, &((struct
sockaddr_in*)a->addr)->sin_addr, ip, INET_ADDRSTRLEN);
        cout << "IP address: " << ip << endl;
    }
}
```

```

        if (a->netmask)
        {
            char mask[INET_ADDRSTRLEN];
            inet_ntop(AF_INET, &((struct sockaddr_in*)a->netmask)->sin_addr, mask, INET_ADDRSTRLEN);
            cout << "Netmask: " << mask << endl;
        }
    }
    count++;
}
cout << "Total devices:" << count << endl;
return count;
}

```

这里的 INET\_ADDRSTRLEN 表示 32 位 IPv4 地址用十进制+句点表示时，所使用内存长度的最大值。IPv4 地址中每一位十进制数字占一字节，一个句点占一字节，再加上最后的 NULL 结束符，一共是 16 字节；同理 IPv6 地址分为两种情况，一种是不兼容格式，每一位十六进制字符占一字节，一个冒号占一字节，再加上最后的 NULL 结束符，一共是 40 字节；另一种是兼容格式，也就是 IPv6 低 32 位地址是 IPv4 地址，这样一共是 46 字节。但是在定义中发现 INET\_ADDRSTRLEN 是 22，INET6\_ADDRSTRLEN 是 65，这其中包括了出现端口号的情况，对于 IPv6 还包含了出现作用域的情况，这样就可以处理各类情况。如下是 ws2ipdef.h 中的定义

```

// Maximum length of address literals (potentially including a port number)
// generated by any address-to-string conversion routine. This length can
// be used when declaring buffers used with getnameinfo, WSAAddressToString,
// inet_ntoa, etc. We just provide one define, rather than one per api,
// to avoid confusion.
//
// The totals are derived from the following data:
// 15: IPv4 address
// 45: IPv6 address including embedded IPv4 address
// 11: Scope Id
// 2: Brackets around IPv6 address when port is present
// 6: Port (including colon)
// 1: Terminating null byte
//
#define INET_ADDRSTRLEN 22
#define INET6_ADDRSTRLEN 65

```

## 5. 让用户选择需要打开的网卡设备

```
// 让用户选择需要打开的网卡设备
pcap_if_t* select_device(pcap_if_t* alldevices, int count)
{
    pcap_if_t* d; // 遍历指针
    int op; // 设备编号
    cout << "Please enter the number of the Network interface Device you want
to open:";
    cin >> op;

LOOP:
    if (op < 1 || op > count) // 输入的数字无效
    {
        cout << "Invalid number!Out of range!" << endl;
        cout << "continue or quit ([c/q]):"; // 选择继续或者退出
        char op;
        cin >> op;
        if (op == 'c' || op == 'C')
            goto LOOP;
        else
        {
            pcap_freealldevs(alldevices); // 释放设备列表
            exit(0);
        }
    }
    // 寻找指定的设备
    int i;
    for (d = alldevices, i = 0; i < op - 1 && d != nullptr; d = d->next, i++);

    return d;
}
```

这一部分与上次实验的代码几乎一模一样，逻辑也很简单，这里就不再过多解释。

## 6. 获取打开的网卡设备的 MAC

```
// 获取打开的网卡设备的 MAC
void GetLocalIPandMAC(const string device_name, byte mac[6])
{
    IP_ADAPTER_INFO adapterinfo[16]; // 最多 16 个适配器
    DWORD dwBufLen = sizeof(adapterinfo); // 缓冲区大小
    // 调用系统 API 获取网络适配器信息
    DWORD dwStatus = GetAdaptersInfo(adapterinfo, &dwBufLen);
```

```

if (dwStatus != ERROR_SUCCESS) {
    std::cerr << "GetAdaptersInfo failed!" << std::endl;
    return;
}
// 提取设备名称中的 GUID，也就是大括号包裹的部分
string guid;
size_t pos = device_name.find("{}");
if (pos != string::npos) {
    guid = device_name.substr(pos);
}
// 根据 GUID 匹配并找到 MAC 地址
for (PIP_ADAPTER_INFO p = adapterinfo;p != nullptr;p = p->Next)
{
    if (guid.find(p->AdapterName) != string::npos)
    {
        memcpy(mac, p->Address, 6);
        return;
    }
}
}

```

这一部分调用 Windows 系统提供的 API 函数 GetAdaptersInfo() 获取网络设备的信息，之后根据打开的网卡设备名称寻找对应的 MAC 地址，这样就获取了源 MAC 地址。这里需要添加头文件如下

```
#include <iphlpapi.h>
```

## 7. main 函数

```

int main()
{
    pcap_if_t* alldevices;           // 设备列表的存储链表
    pcap_if_t* device;              // 遍历指针
    char errbuf[PCAP_ERRBUF_SIZE]; // 存储错误信息，256 字节
    byte localIP[4];                // 需要打开设备的 IP
    byte targetIP[4];               // 目标 IP
    byte localMAC[6];               // 本机 MAC 地址
    byte packet[42];                // 数据包
    int count;                      // 可用设备数量
    cout << "=====Devices
List=====" << endl;

```

```

// 从本地设备中获取可用设备列表
if (pcap.findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevices,
errbuf) == -1)
{
    // 查找失败，则输出错误信息
    cerr << "ERROR:" << errbuf << endl;
    return 1;
}

// 查找成功的话，设备列表已经存储在 alldevices 里面
count = PrintDeviceInfo(alldevices);
device = select_device(alldevices, count);
if (!device)
{
    return 1;
}

pcap_t* handle;
// 打开设备
handle = pcap_open(device->name, 65536, PCAP_OPENFLAG_PROMISCUOUS,
100, NULL, errbuf);
if (!handle)
{
    cerr << "ERROR:" << errbuf << endl;
    return 1;
}

cout << "Successfully open network device:" << device->name << endl;

// 输入目标地址的 IP
string targetip;
cout << "Please enter the target IP address:";
cin >> targetip;
inet_pton(AF_INET, targetip.c_str(), targetIP);

// 获取该设备的 IP 以及 MAC
struct sockaddr_in* sin = (struct
sockaddr_in*)device->addresses->addr;
DWORD ip = sin->sin_addr.S_un.S_addr; // 网络字节序，转换 IP 的格式
memcpy(localIP, &ip, 4);
GetLocalIPandMAC(device->name, localMAC);
cout << "Source IP and MAC: " << inet_ntoa(sin->sin_addr) << " -> ";
print_MAC(localMAC);

// 构造 ARP 报文
EthernetHeader* eth = (EthernetHeader*)packet;
ARP* arp = (ARP*)(packet + sizeof(EthernetHeader));
// 以太网帧头部
memset(eth->dstMAC, 0xFF, 6); // 广播所有设备

```

```
    memcpy(eth->srcMAC, localMAC, 6);
    eth->ethertype = htons(0x0806);      // ARP 协议
    // ARP 报文
    arp->hardware = htons(0x0001); // 硬件类型为以太网
    arp->protocol = htons(0x0800);
    arp->MAC_length = 6;
    arp->IPaddress_length = 4;
    arp->opcode = htons(0x0001);
    memcpy(arp->srcIP, localIP, 4);
    memcpy(arp->dstIP, targetIP, 4);
    memcpy(arp->srcMAC, localMAC, 6);
    memset(arp->dstMAC, 0, 6);

    // pcap_sendpacket() 函数能够利用网卡设备的句柄发送数据包
    if (pcap_sendpacket(handle, packet, 42) != 0)
    {
        cerr << "Error sending ARP request." << endl;
        return -1;
    }

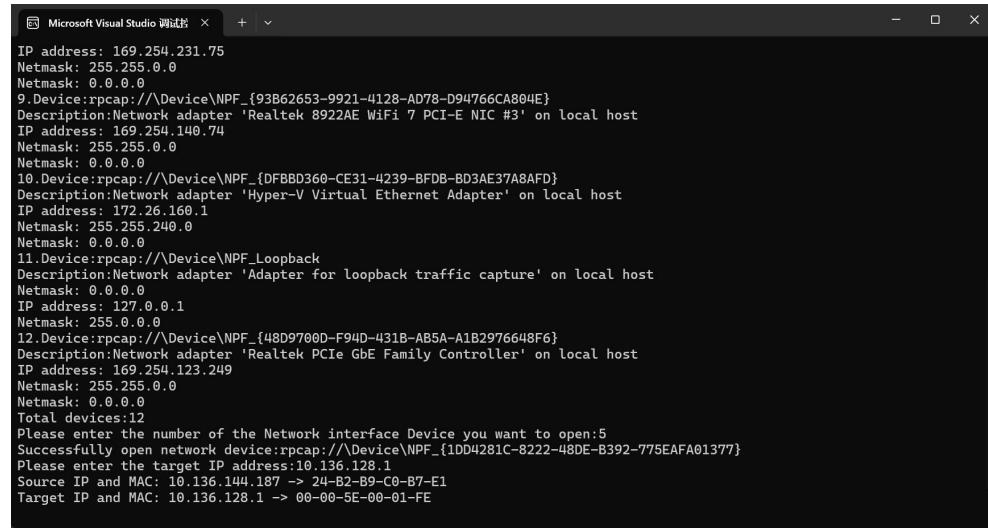
    // 捕获 ARP 数据包
    struct pcap_pkthdr* header;
    const u_char* recv_data;
    int res;
    while ((res = pcap_next_ex(handle, &header, &recv_data)) >= 0)
    {
        if (res == 0) continue;
        // 解析以太网帧
        EthernetHeader* recv_ether = (EthernetHeader*)recv_data;
        if (ntohs(recv_ether->ethertype) == 0x0806)
        {
            // 解析 ARP 数据包
            ARP* recv_arp = (ARP*)(recv_data + sizeof(EthernetHeader));
            if (ntohs(recv_arp->opcode) == 2)
            {
                // 验证 IP，返回的数据包中的 IP 是否和目标 IP 一致
                if (memcmp(recv_arp->srcIP, targetIP, 4) == 0)
                {
                    cout << "Target IP and MAC: " << targetip << " ->";
                    print_MAC(recv_arp->srcMAC);
                    break;
                }
            }
        }
    }
}
```

```
    pcap_close(handle);  
    pcap_freealldevs(alldevices);  
  
    return 0;  
}
```

在 main 函数中，首先从本地设备中获取设备列表，然后让用户选择想要打开的设备，之后让用户输入目标 IP，构造并发送 ARP 数据包，最后捕获回复的 ARP 数据包并分析找到目标 MAC 地址，这样就完成了从输入目标 IP 到获取目标 MAC 的整个过程。

在代码运行时，输入的 IP 地址对应的设备应当与打开的网卡设备处于同一网段中，这样才可以捕获到回复的 ARP 数据包并得到正确的目标 MAC；如果处在不同网段中，最终得到的其实是回环数据，也就是发送出去的数据包被原路返回，得到的目标 MAC 也是源 MAC，无法验证 ARP 协议的功能。

我首先使用 arp -a 命令获取了当前的 arp 项，然后选取了网关设备作为目标，IP 地址为 10.136.128.1。之后我打开了 WiFi 7 PCI-E NIC 网卡，它与网关设备在同一网段中，运行后得到了如下的结果



```
Microsoft Visual Studio 调试器 x + v  
IP address: 169.254.231.75  
Netmask: 255.255.0.0  
Netmask: 0.0.0.0  
9.Device:rpcap://\Device\NPF_{93B62653-9921-4128-AD78-D94766CA804E}  
Description:Network adapter 'Realtek 8922AE WiFi 7 PCI-E NIC #3' on local host  
IP address: 169.254.140.74  
Netmask: 255.255.0.0  
Netmask: 0.0.0.0  
10.Device:rpcap://\Device\NPF_{DFBB0360-CE31-4239-BFDB-BD3AE37A8AFD}  
Description:Network adapter 'Hyper-V Virtual Ethernet Adapter' on local host  
IP address: 172.26.160.1  
Netmask: 255.255.240.0  
Netmask: 0.0.0.0  
11.Device:rpcap://\Device\NPF_Loopback  
Description:Network adapter 'Adapter for loopback traffic capture' on local host  
Netmask: 0.0.0.0  
IP address: 127.0.0.1  
Netmask: 255.0.0.0  
12.Device:rpcap://\Device\NPF_{48D9700D-F94D-431B-AB5A-A1B2976648F6}  
Description:Network adapter 'Realtek PCIe GbE Family Controller' on local host  
IP address: 169.254.123.249  
Netmask: 255.255.0.0  
Netmask: 0.0.0.0  
Total devices:12  
Please enter the number of the Network interface Device you want to open:5  
Successfully open network device:rpcap://\Device\NPF_{1DD4281C-8222-48DE-B392-775EAFA01377}  
Please enter the target IP address:10.136.128.1  
Source IP and MAC: 10.136.144.187 -> 24-B2-B9-C0-B7-E1  
Target IP and MAC: 10.136.128.1 -> 00-00-5E-00-01-FE
```

10.136.144.187 是我电脑的 IP 地址，也是 Wifi 网卡的 IP 地址；目标 IP 地址是网关设备的 IP 地址，通过 ARP 找到了 MAC 地址。

C:\Users\LIANXIANG>arp -a		
接口： 10.136.144.187 --- 0x4		
Internet 地址	物理地址	类型
10.136.128.1	00-00-5e-00-01-fe	动态
10.136.170.198	00-00-5e-00-01-fe	动态
10.136.255.255	ff-ff-ff-ff-ff-ff	静态
224.0.0.2	01-00-5e-00-00-02	静态
224.0.0.22	01-00-5e-00-00-16	静态
224.0.0.251	01-00-5e-00-00-fb	静态
224.0.0.252	01-00-5e-00-00-fc	静态
239.192.152.143	01-00-5e-40-98-8f	静态
239.255.255.250	01-00-5e-7f-ff-fa	静态
255.255.255.255	ff-ff-ff-ff-ff-ff	静态

上面的图片是 arp -a 命令执行的结果，下面的图片是使用 Wireshark 捕获数据包的结果，这些和代码运行得到的 MAC 地址结果都相同，说明成功通过 IP 地址获取了 MAC 地址，代码没有问题！

No.	arp	time	Source	Destination	Protocol	Length	Info
139	16.319529		LiteonTechno_c0:b7:...	Broadcast	ARP	42	Who has 10.136.128.1? Tell 10.136.144.187
140	16.325253		IETF-VRRP-VRID_fe	LiteonTechno_c0:b7:...	ARP	56	10.136.128.1 is at 00:00:5e:00:01:fe

## 四、实验结论及心得体会

本次实验让我对于 ARP 请求-回应机制有了新的认识，通过实验证明了 ARP 获取 MAC 地址的完整过程。在实验中，很多函数都是上一次实验使用过的，对于这些函数的用法又进行了一次巩固，对于网络编程也有了更深刻的认识。