南 开 大 学

网络空间安全学院学院

网络技术与应用课程报告

第 3 次实验报告

学号：2011428
姓名：王天行
年级：2020 级
专业：密码科学与技术

2022 年 10 月 13 日

第 1 节 实验内容说明

通过编程获取 IP 地址与 MAC 地址的对应关系实验，要求如下：
（1）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。
（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。

第 2 节 实验准备

学习 Npcap 发送数据包的方法

pcap_sendpacket()

用 pcap_sendpacket()来发送一个数据包，这个函数需要参数：

- 一个装有要发送数据的缓冲区；
- 要发送的长度；
- 一个适配器；

注意：缓冲区中的数据将不被内核协议处理，只是作为最原始的数据流被发送，所以我们必须填充好正确的协议头以便正确的将数据发送

pcap_sendqueue_alloc()

pcap_sendpacket()只是提供了一个简单的直接的发送数据的方法，而发送队列提供了一个高级的、强大的、和最优的机制来发送一组数据包，队列实际上是一个装有要发送数据的一个容器，他又一个最大值来表明它能够容纳的最大比特数。

pcap_sendqueue_alloc()用来创建爱你一个队列，并指定该队列的大小。

一旦队列被创建就可以调用 pcap_sendqueue_queue()来将数据存储到队列中，这个函数接受一个带有时间戳和长度的 pcap_pkthdr 结构和一个装有数据报的缓冲区。这些参数同样也应用于 pcap_next_ex()和 pcap_handler()中，所以给要捕获的数据包或要从文件读取的数据包排队就是 pcap_sendqueue_queue()的事情了。

WinPcap 调用 pcap_sendqueue_transmit()来发送数据包，注意，第三个参数如果非零，那么发送将是同步的，这将站用很大的 CPU 资源，因为发生在内核驱动的同步发送是通过"brute force"loops 的，但是一般情况下能够精确到微秒。

需要指出的是用 pcap_sendqueue_transmit()来发送比用 pcap_sendpacket()来发送一系列的数据要高效的多，因为他的数据是在内核级上被缓冲。

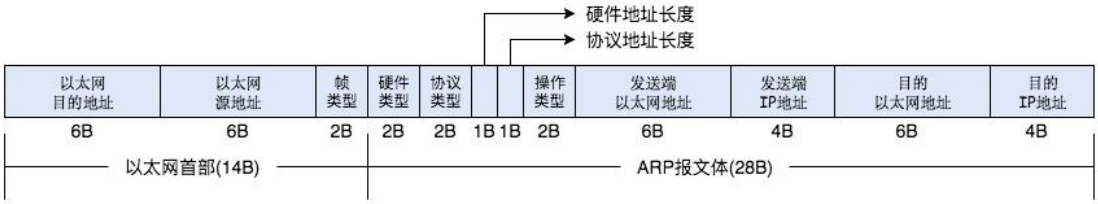
当不再需要队列时可以用 pcap_sendqueue_destroy()来释放掉所有的队列资源。

了解 ARP 协议与 ARP 数据包结构

ARP 请求过程

1、主机 A 先在 ARP 缓存表中查找主机 B 的 MAC 地址
2、如果在 A 的 ARP 缓存表中找到了，那么就继续数据封装通信。如果没有找到主机 A 会先发送 ARP 的广播包（里面包括了 A 的 IP 地址和 MAC 地址、主机 B 的 IP 地址）
3、既然是广播帧那么内网中的所有存活主机都会收到该 ARP 的广播包
4、存活主机会进行检查自身 IP 地址是否与广播包中的目的 IP 地址一致，如果不一致的话进行丢弃。如果一致那么就会将主机 A 的 IP 和 MAC 地址添加到自己的 ARP 缓存表里面，然后再将自己的 MAC 地址和 ARP 响应包通过单播方式发送给主机 A
5、然后主机 A 就可以给主机 B 发送消息

ARP 数据包结构



从而得到 ARP 报文结构：

```
#pragma pack(1)
typedef struct Ethernet_head {  //帧首部
    BYTE    DesMAC[6];  // 目的地址+
    BYTE    SrcMAC[6];  // 源地址+
    WORD    EthType;    // 帧类型
};
typedef struct ArpPacket {  //包含帧首部和ARP首部的数据包
    Ethernet_head   ed;
    WORD HardwareType; //硬件类型
    WORD ProtocolType; //协议类型
    BYTE HardwareAddLen; //硬件地址长度
    BYTE ProtocolAddLen; //协议地址长度
    WORD OperationField; //操作类型，ARP请求（1），ARP应答（2），RARP请求（3），RARP应答（4）。
    BYTE SourceMacAdd[6]; //源mac地址
    DWORD SourceIpAdd; //源ip地址
    BYTE DestMacAdd[6]; //目的mac地址
    DWORD DestIpAdd; //目的ip地址
};
#pragma pack()
```

第 3 节 实验过程

整体代码思路

首先与之前的实验相同，获得本机设备列表和接口列表，跳转到指定的网卡并打开网卡。（因为与之前部分代码一致且代码较长，此处就不放代码截图了）之后需要获取到本机的 MAC 地址和目的 IP 地址，并根据已知数据填充 ARP 数据包，使用 sendpacket()函数将数据包发送到网络中去。之后开始读取接收到的 ARP 数据包，并分析出目的 IP 地址和 MAC 地址。

```
printf(_Format: "\nlistening on %s...\n", d->description);
m_MAC = (unsigned char*)GetSelfMac();

//HANDLE sendthread;      //发送ARP包线程
//HANDLE recvthread;       //接受ARP包线程

//sendthread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)SendArpPacket, adhandle, 0, NULL);
//recvthread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)GetLivePC, adhandle, 0, NULL);

SendArpPacket(adhandle);
GetLivePC(adhandle);

pcap_freealldevs(alldevs);
//CloseHandle(sendthread);
//CloseHandle(recvthread);
return 0;
```

涉及函数及其功能

获取本机 MAC 地址（GetSelfMac()）

```
char* GetSelfMac() {
    ULONG MacAddr[2] = { 0 };    // Mac地址长度6字节
    ULONG uMacSize = 6;
    DWORD dwRet = SendARP(inet_addr(m_IP), 0, &MacAddr, &uMacSize);
    if (dwRet == NO_ERROR)
    {
        BYTE* bPhyAddr = (BYTE*)MacAddr;

        if (uMacSize)
        {
            char* sMac = (char*)malloc(sizeof(char) * 18);
            int n = 0;

            memset(sMac, 0, 18);
            sprintf_s(sMac, (size_t)18, "%.2X-%.2X-%.2X-%.2X-%.2X-%.2X", (int)bPhyAddr[0], (int)bPhyAddr[1], (int)bPhyAddr[2], (int)bPhyAddr[3], (int)bPhyAddr[4], (int)bPhyAddr[5]);
            return sMac;
        }
        else
        {
            printf("Mac地址获取失败！\n");
        }
    }
    else
    {
        printf("ARP报文发送失败:%d\n", dwRet);
    }
    return NULL;
```

发送数据包（SendArpPacket(adhandle)）

```
void SendArpPacket(pcap_t* adhandle) {
    char* ip = m_IP;
    unsigned char* mac = m_MAC;
    char* netmask = m_mask;
    printf("ip_mac:%02x-%02x-%02x-%02x-%02x-%02x\n", mac[0], mac[1], mac[2],
        mac[3], mac[4], mac[5]);
    printf("自身的IP地址为:%s\n", ip);
    printf("地址掩码NETMASK为:%s\n", netmask);
    printf("\n");
    unsigned char* sendbuf; //arp包结构大小
    ArpPacket arp;

    //赋值MAC地址
    memset(arp.ed.DesMAC, 0xff, 6);         //目的地址为全为广播地址
    printf("Des_MAC:%02x-%02x-%02x-%02x-%02x-%02x\n", arp.ed.DesMAC[0], arp.ed.DesMAC[1], arp.ed.DesMAC[2],
        arp.ed.DesMAC[3], arp.ed.DesMAC[4], arp.ed.DesMAC[5]);
    memcpy(arp.ed.SrcMAC, mac, 6);
    printf("Src_MAC:%02x-%02x-%02x-%02x-%02x-%02x\n", arp.ed.SrcMAC[0], arp.ed.SrcMAC[1], arp.ed.SrcMAC[2],
        arp.ed.SrcMAC[3], arp.ed.SrcMAC[4], arp.ed.SrcMAC[5]);
```

```
memcpy(arp.SourceMacAdd, mac, 6);
printf("SourceMacAdd:%02x-%02x-%02x-%02x-%02x-%02x\n", arp.SourceMacAdd[0], arp.SourceMacAdd[1], arp.SourceMacAdd[2],
    arp.SourceMacAdd[3], arp.SourceMacAdd[4], arp.SourceMacAdd[5]);
memset(arp.DestMacAdd, 0x00, 6);
printf("DestMacAdd:%02x-%02x-%02x-%02x-%02x-%02x\n", arp.DestMacAdd[0], arp.DestMacAdd[1], arp.DestMacAdd[2],
    arp.DestMacAdd[3], arp.DestMacAdd[4], arp.DestMacAdd[5]);
arp.ed.EthType = htons(0x0806);//帧类型为ARP3
cout << "EthTyte:" << arp.ed.EthType << endl;
arp.HardwareType = htons(0x0001);
cout << "HardwareType:" << arp.HardwareType << endl;
arp.ProtocolType = htons(0x0800);
cout << "ProtocolType:" << arp.ProtocolType << endl;
arp.HardwareAddLen = 6;
arp.ProtocolAddLen = 4;
arp.SourceIpAdd = inet_addr(ip); //请求方的IP地址为自身的IP地址
cout << "SourceIpAss:" << arp.SourceIpAdd << endl;
arp.OperationField = htons(0x0001);
cout << "OperationField:" << arp.OperationField << endl;
arp.DestIpAdd = inet_addr(d_IP);
```

```
    arp.OperationField = htons(0x0001);
    cout << "OperationField:" << arp.OperationField << endl;
    arp.DestIpAdd = inet_addr(d_IP);
    cout << "DestIpAdd:" << arp.DestIpAdd << endl;
    u_char* a = (u_char*)&arp;
    //如果发送成功
    cout << "a:";
    for (int i = 0; i < 48; i++) {
        printf("% 02x ", a[i]);
    }
    cout << endl;
    if (pcap_sendpacket(adhandle, (u_char*)&arp, sizeof(ArpPacket)) == 0) {
        printf("\nPacketSend succeed\n");
    }
    else {
        printf("PacketSendPacket in getmine Error: %d\n", GetLastError());
    }
    flag = TRUE;
    return;
}
```

接收数据包（GetLivePC(adhandle)）

```
//(pcap_t *adhandle)
int GetLivePC(pcap_t* adhandle) {
    //gparam *gpara = (gparam *)lpParameter;
    //pcap_t *adhandle = gpara->adhandle;
    int res;
    unsigned char Mac[6];
    struct pcap_pkthdr* pkt_header;
    const u_char* pkt_data;
    while ((res = pcap_next_ex(adhandle, &pkt_header, &pkt_data)) >= 0) {
        //cout << "ETH_ARP:" << *(unsigned short*)(pkt_data + 12) << "        " << htons(0x0806) << endl;
        if (*(unsigned short*)(pkt_data + 12) == htons(0x0806)) {
            ArpPacket* recv = (ArpPacket*)pkt_data;
            //cout << "ARP_REPLY:" << *(unsigned short*)(pkt_data + 20) << "        " << htons(ARP_REPLY) << endl;
            if (*(unsigned short*)(pkt_data + 20) == htons(ARP_REPLY)) {
                printf("sourceIP地址:%d.%d.%d.%d    MAC地址:",
                    *(unsigned char*)(pkt_data + 28) & 255,
                    *(unsigned char*)(pkt_data + 28 + 1) & 255,
                    *(unsigned char*)(pkt_data + 28 + 2) & 255,
                    *(unsigned char*)(pkt_data + 28 + 3) & 255);
                for (int i = 0; i < 6; i++) {
                    Mac[i] = *(unsigned char*)(pkt_data + 22 + i);
                    printf("%02x ", Mac[i]);
                }
                printf("\n");
                printf("destinationIP地址:%d.%d.%d.%d    MAC地址:",
                    *(unsigned char*)(pkt_data + 38) & 255,
                    *(unsigned char*)(pkt_data + 38 + 1) & 255,
                    *(unsigned char*)(pkt_data + 38 + 2) & 255,
                    *(unsigned char*)(pkt_data + 38 + 3) & 255);
                for (int i = 0; i < 6; i++) {
                    Mac[i] = *(unsigned char*)(pkt_data + 32 + i);
                    printf("%02x ", Mac[i]);
                }
                printf("\n");
                return 0;
            }
        }
        Sleep(10);
    }
    return 0;
```

实验结果

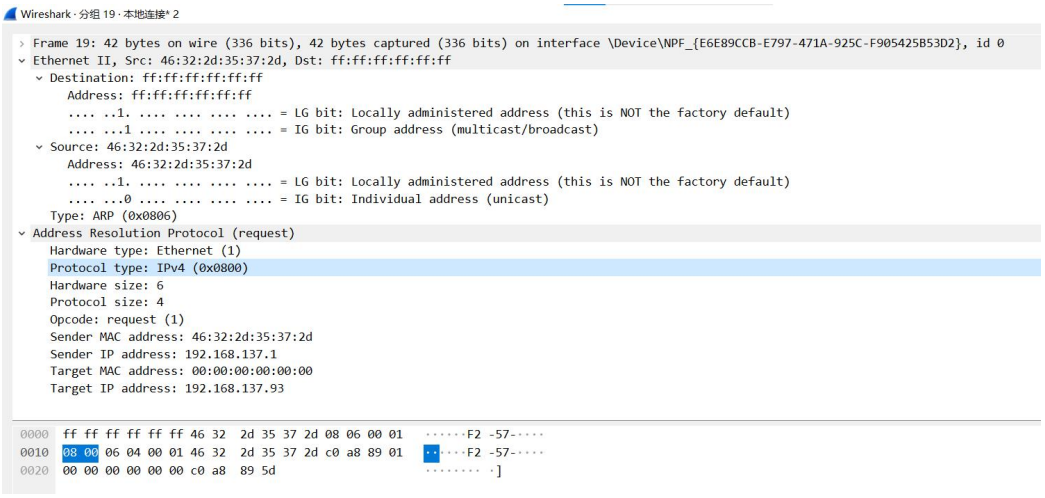
代码输出结果：

```
Microsoft Visual Studio 调试控制台
输入目标IP:192.168.137.93
ip_mac:46-32-2d-35-37-2d
自身的IP地址为:192.168.137.1
地址掩码NETMASK为:255.255.255.0

Des_MAC:ff-ff-ff-ff-ff-ff
Src_MAC:46-32-2d-35-37-2d
SourceMacAdd:46-32-2d-35-37-2d
DestMacAdd:00-00-00-00-00-00
EthTyte:1544
HardwareType:256
ProtocolType:8
SourceIpAss:25798848
OperationField:256
DestIpAdd:1569302720
a:ff ff ff ff ff ff 46 32 2d 35 37 2d 08 06 00 01 08 00 06 04 00 01 46 32 2d 35 37 2d c0 a8 89 01 00 00 00 00 00 00 c0 a8 89 5d cc cc cc cc cc cc

PacketSend succeed
sourceIP地址:192.168.137.93    MAC地址:92 e2 b4 52 c9 3e
destinationIP地址:192.168.137.1    MAC地址:46 32 2d 35 37 2d

D:\wtx\computer-network\计算机网络\Lab\网技-EX3\ARP\x64\Debug\ARP.exe (进程 18932)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

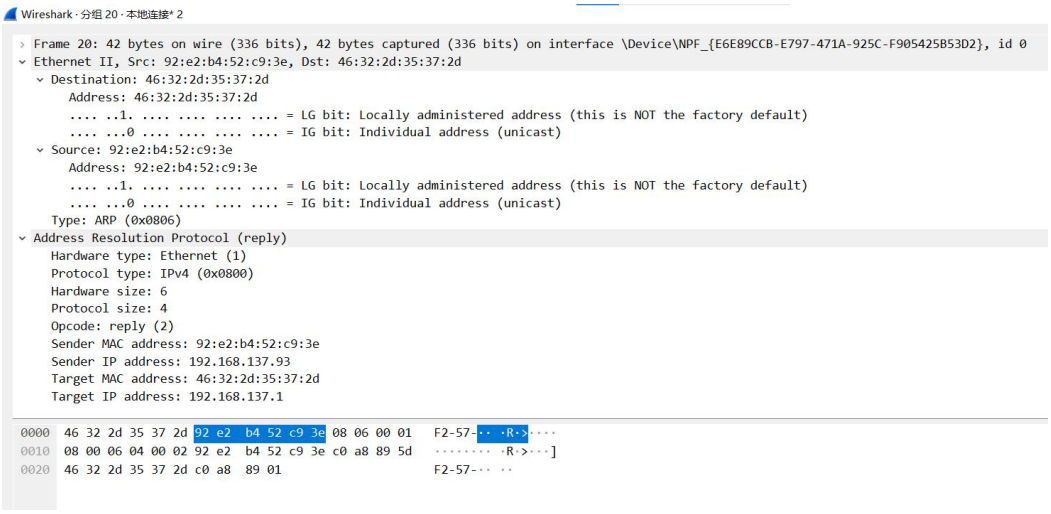
Wireshark 抓包结果：

19	12.525207	46:32:2d:35:37:2d	ff:ff:ff:ff:ff:ff	ARP	42	Who has 192.168.137.93? Tell 192.168.137.1
20	12.822539	92:e2:b4:52:c9:3e	46:32:2d:35:37:2d	ARP	42	192.168.137.93 is at 92:e2:b4:52:c9:3e

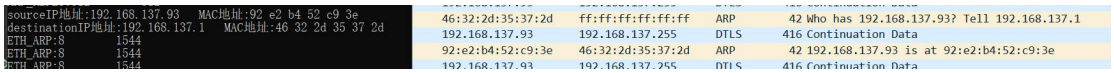
第一个数据包：


Wireshark · 分组 19 · 本地连接* 2
Frame 19: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{E6E89CCB-E797-471A-925C-F905425B53D2}, id 0
Ethernet II, Src: 46:32:2d:35:37:2d, Dst: ff:ff:ff:ff:ff:ff
Destination: ff:ff:ff:ff:ff:ff
Address: ff:ff:ff:ff:ff:ff
.... ..1. .... = LG bit: Locally administered address (this is NOT the factory default)
....1 .... = IG bit: Group address (multicast/broadcast)
Source: 46:32:2d:35:37:2d
Address: 46:32:2d:35:37:2d
.... ..1. .... = LG bit: Locally administered address (this is NOT the factory default)
....0 .... = IG bit: Individual address (unicast)
Type: ARP (0x0806)
Address Resolution Protocol (request)
Hardware type: Ethernet (1)
Protocol type: IPv4 (0x0800)
Hardware size: 6
Protocol size: 4
Opcode: request (1)
Sender MAC address: 46:32:2d:35:37:2d
Sender IP address: 192.168.137.1
Target MAC address: 00:00:00:00:00:00
Target IP address: 192.168.137.93
0000 ff ff ff ff ff ff 46 32 2d 35 37 2d 08 06 00 01
0010 08 00 06 04 00 01 46 32 2d 35 37 2d c0 a8 89 01
0020 00 00 00 00 00 00 c0 a8 89 5d

第二个数据包：


Wireshark · 分组 20 · 本地连接* 2
Frame 20: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{E6E89CCB-E797-471A-925C-F905425B53D2}, id 0
Ethernet II, Src: 92:e2:b4:52:c9:3e, Dst: 46:32:2d:35:37:2d
Destination: 46:32:2d:35:37:2d
Address: 46:32:2d:35:37:2d
.... ..1. .... = LG bit: Locally administered address (this is NOT the factory default)
....0 .... = IG bit: Individual address (unicast)
Source: 92:e2:b4:52:c9:3e
Address: 92:e2:b4:52:c9:3e
.... ..1. .... = LG bit: Locally administered address (this is NOT the factory default)
....0 .... = IG bit: Individual address (unicast)
Type: ARP (0x0806)
Address Resolution Protocol (reply)
Hardware type: Ethernet (1)
Protocol type: IPv4 (0x0800)
Hardware size: 6
Protocol size: 4
Opcode: reply (2)
Sender MAC address: 92:e2:b4:52:c9:3e
Sender IP address: 192.168.137.93
Target MAC address: 46:32:2d:35:37:2d
Target IP address: 192.168.137.1
0000 46 32 2d 35 37 2d 92 e2 b4 52 c9 3e 08 06 00 01
0010 08 00 06 04 00 02 92 e2 b4 52 c9 3e c0 a8 89 5d
0020 46 32 2d 35 37 2d c0 a8 89 01

二者对比：


sourceIP地址:192.168.137.93 MAC地址:92 e2 b4 52 c9 3e
destinationIP地址:192.168.137.1 MAC地址:46 32 2d 35 37 2d
ETH_ARP:8 1544
ETH_ARP:8 1544
46:32:2d:35:37:2d ff:ff:ff:ff:ff:ff ARP 42 Who has 192.168.137.93? Tell 192.168.137.1
192.168.137.93 192.168.137.255 DTLS 416 Continuation Data
92:e2:b4:52:c9:3e 46:32:2d:35:37:2d ARP 42 192.168.137.93 is at 92:e2:b4:52:c9:3e
192.168.137.93 192.168.137.255 DTLS 416 Continuation Data