网络技术与应用课程实验报告

姓名：孟启轩

学号：2212452

专业：计算机科学与技术

实验 3：通过编程获取 IP 地址与 MAC 地址的对应关系

一、实验内容：

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 NPcap 的数据包发送方法。

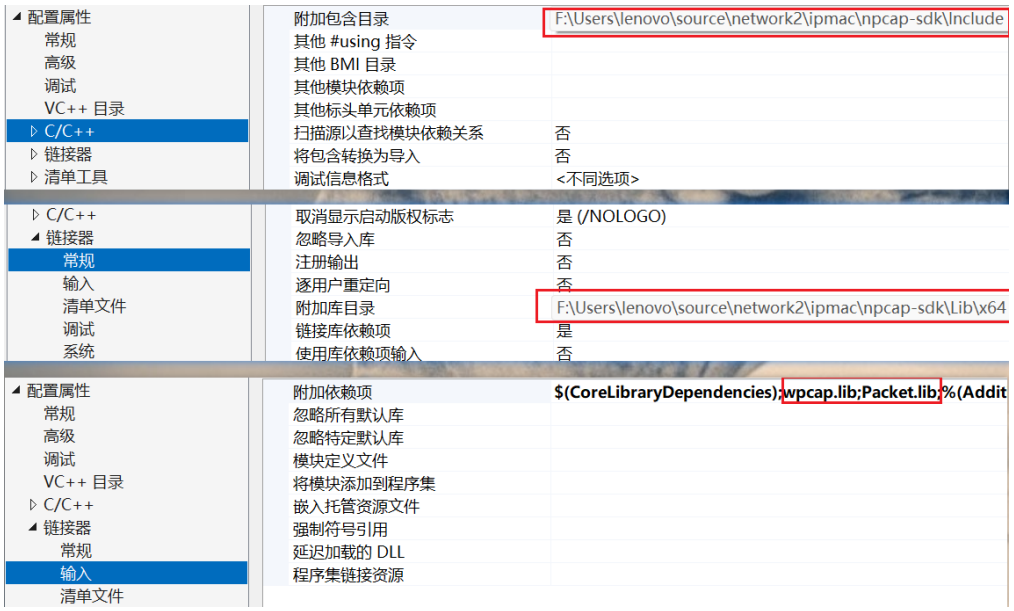
（2）通过 NPcap 编程，获取 IP 地址与 MAC 地址的映射关系。

（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。

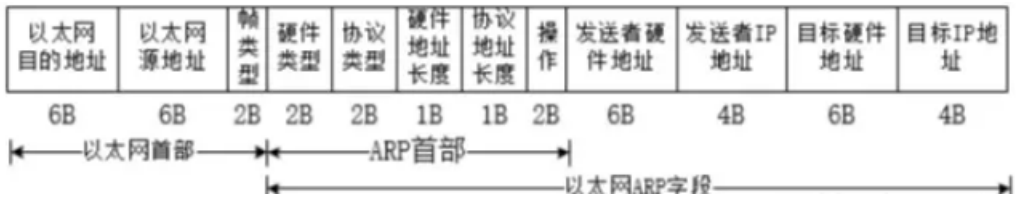
二、前期准备：

1. **安装 WinPcap 驱动程序和 DLL。**
2. **创建基于 WinPcap 的应用程序，并配置项目属性：**
   - 添加 `pcap.h` 头文件。
   - 在预处理器定义中增加与 WinPcap 相关的标识：添加 `WPCAP` 和 `HAVE_REMOTE`。
   - 配置包含文件目录：将 `pcap.h` 所在的 `Include` 文件夹路径添加到 IDE 中。
   - 添加 `wpcap.lib`和`Packet.lib` 库文件：将库文件所在的 `Lib` 文件夹路径添加到 IDE 中。



3. **了解 arp 帧**

ARP（Address Resolution Protocol，地址解析协议）是一种用于 IPv4 网络中的协议，它负责将网络层的 IP 地址解析为数据链路层的 MAC 地址。通过 ARP 协议，设备可以在局域网中通信前获取目标设备的 MAC 地址，从而在以太网中进行有效的数据帧传输。



ARP 报文主要有两种类型：ARP 请求和 ARP 响应。请求报文用于查询 MAC 地址，响应报文用于返回 MAC 地址。ARP 报文在以太网中传输，其格式如下：

字段	长度（字节）	含义
硬件类型（Hardware Type）	2	指明硬件地址类型，Ethernet的值为1
协议类型（Protocol Type）	2	指明协议地址类型，IPv4的值为0x0800
硬件地址长度（Hardware Size）	1	硬件地址的长度，以字节为单位。以太网MAC地址的长度为6字节
协议地址长度（Protocol Size）	1	协议地址的长度，以字节为单位。IPv4地址的长度为4字节
操作码（Opcode）	2	操作类型：1表示ARP请求，2表示ARP响应
发送方MAC地址（Sender MAC Address）	6	发送方的MAC地址
发送方IP地址（Sender IP Address）	4	发送方的IP地址
目标MAC地址（Target MAC Address）	6	目标设备的MAC地址。ARP请求时填充0，ARP响应时填充目标的MAC地址
目标IP地址（Target IP Address）	4	目标设备的IP地址

4.arp 帧运行流程

(1)发送 ARP 请求

• 当主机 A 需要与主机 B 通信时，会检查自己的 ARP 缓存表（ARP Cache），以查看是否已有目标 IP 地址对应的 MAC 地址。

• 如果缓存表中没有找到主机 B 的 MAC 地址，主机 A 会生成一个 ARP 请求帧，请求帧包含主机 A 的 IP 地址和 MAC 地址、目标主机 B 的 IP 地址，以及一个空的目标 MAC 地址字段。

• 主机 A 将 ARP 请求帧封装为以太网帧，并向局域网内的所有设备广播（广播 MAC 地址：FF:FF:FF:FF:FF:FF）。

(2)接收 ARP 请求

• 网络中的所有设备都会收到此广播帧，并检查帧中的目标 IP 地址是否与自己的 IP 地址匹配。

• 如果某设备（假设为主机 B）发现帧中的目标 IP 地址与自己的 IP 地址匹配，则识别该帧是发给它的 ARP 请求帧；其他设备则丢弃该帧。

(3)发送 ARP 响应

• 主机 B 会生成一个 ARP 响应帧，其中包含自己的 MAC 地址及 IP 地址，并将帧的目标 MAC 地址设置为主机 A 的 MAC 地址。

• ARP 响应是一个单播帧，因此主机 B 只将该帧发送给主机 A，而不会再广播。

(4)接收 ARP 响应

• 主机 A 收到主机 B 的 ARP 响应帧后，从中提取到主机 B 的 MAC 地址，并将 IP 地址与 MAC 地址的对应关系缓存到自己的 ARP 表中。

• 主机 A 和主机 B 已具备相互通信的 MAC 地址信息，从而可以在局域网内正常发送数据帧。

(5)ARP 缓存更新

• ARP 表中的条目通常有一定的生存时间（TTL），一段时间后会失效并删除。

• 当条目过期时，若主机仍需与该目标通信，就会重新发送 ARP 请求。

三、实验过程：

（1）程序主要部分详解

```
// 将地址转换为16进制字符串类型
string* Byte2Hex(unsigned char bArray[], int bArray_len)
{
    // 初始化一个新的字符串指针，用于存储最终的十六进制字符串
    string* strHex = new string();

    for (int i = 0; i < bArray_len; i++)
    {
        char hex1; // 存储高四位的十六进制字符
        char hex2; // 存储低四位的十六进制字符
        int value = bArray[i]; // 获取当前字节的值

        int S = value / 16; // 高四位
        int Y = value % 16; // 低四位

        // 将高四位转换为对应的十六进制字符
        if (S >= 0 && S <= 9)
            hex1 = (char)(48 + S); // 0-9转换为字符'0'-'9'
        else
            hex1 = (char)(55 + S); // 10-15转换为字符'A'-'F'

        // 将低四位转换为对应的十六进制字符
        if (Y >= 0 && Y <= 9)
            hex2 = (char)(48 + Y);
        else
            hex2 = (char)(55 + Y);

        // 拼接当前字节的十六进制字符串
        // 若不是最后一个字节，则在后面添加分隔符'-'
        if (i != bArray_len - 1) {
            *strHex = *strHex + hex1 + hex2 + "-";
        }
        else
            *strHex = *strHex + hex1 + hex2;
    }
    return strHex; // 返回最终的十六进制字符串指针
}
```

计算当前字节的高四位和低四位，将每个部分分别转换为十六进制字符，再拼接字符串，并在每个字节间添加分隔符。这段程序的主要作用是将 MAC 地址字节数组转换为十六进制字符串格式，其中每个字节转换为两位十六进制字符。

```cpp
#pragma pack(1) // 设定字节对齐为1字节，确保结构体按精确字节存储
#define BYTE unsigned char // 定义 BYTE 类型，等价于 unsigned char

// 帧首部
typedef struct FrameHeader_t {
    BYTE DesMAC[6];  // 目的 MAC 地址（6 字节）
    BYTE SrcMAC[6];  // 源 MAC 地址（6 字节）
    WORD FrameType;  // 帧类型（2 字节），例如表示 ARP 类型
} FrameHeader_t;

// ARP 帧
typedef struct ARPFrame_t {
    FrameHeader_t FrameHeader; // 帧首部，包含源 MAC、目的 MAC 和帧类型
    WORD HardwareType;         // 硬件类型（2 字节），表示硬件地址类型
    WORD ProtocolType;         // 协议类型（2 字节），表示协议地址类型
    BYTE HLen;                 // 硬件地址长度（1 字节），通常为 6（MAC 地址长度）
    BYTE PLen;                 // 协议地址长度（1 字节），通常为 4（IPv4 地址长度）
    WORD Operation;            // 操作类型（2 字节），1 为请求，2 为响应
    BYTE SendHa[6];            // 源 MAC 地址（6 字节）
    DWORD SendIP;              // 源 IP 地址（4 字节）
    BYTE RecvHa[6];            // 目的 MAC 地址（6 字节）
    DWORD RecvIP;              // 目的 IP 地址（4 字节）
} ARPFrame_t;
```

这段程序定义了以太网帧和 ARP 帧的结构体，用于表示和处理 ARP 协议的帧格式。通过定义结构体来组织帧头部和各字段，方便在程序中操作 ARP 帧的数据。定义了 ARP 请求或响应帧的结构，包括以太网帧头部、硬件类型、协议类型、操作类型、源和目的的 MAC/IP 地址等字段。该结构体用于构建、解析和发送 ARP 帧。

FrameHeader_t：表示以太网帧的头部，包含目的 MAC、源 MAC 和帧类型。

ARPFrame_t：表示 ARP 帧，包含以太网帧头部和 ARP 报文的主要字段，如硬件类型、协议类型、操作类型、源和目的的 MAC 和 IP 地址等，用于表示完整的 ARP 请求或响应帧。

一些函数：

```cpp
#pragma pack()
ARPFrame_t ARPFrame;//要发送的APR数据包(其他主机)
ARPFrame_t ARPF_Send;//要发送的APR数据包（本机）
unsigned char mac[44], desmac[44];//目的主机和其他主机的mac


void ARP_show(struct pcap_pkthdr* header, const u_char* pkt_data)
{
    struct ARPFrame_t* arp;
    arp = (struct ARPFrame_t*)(pkt_data);
    in_addr source, aim;
    memcpy(&source, &arp->SendIP, 4);
    memcpy(&aim, &arp->RecvIP, sizeof(in_addr));
    cout << "源MAC地址:   " << *(Byte2Hex(arp->FrameHeader.SrcMAC, 6)) << endl;
    cout << "源IP地址:    " << inet_ntoa(source) << endl;
    cout << "目的MAC地址: " << *(Byte2Hex(arp->FrameHeader.DesMAC, 6)) << endl;
    cout << "目的IP地址   " << inet_ntoa(aim) << endl;
    cout << endl;
}

//获取本机网络接口的MAC地址和IP地址
void printAddressInfo(const pcap_addr_t* a) {
    char str[INET_ADDRSTRLEN];

    if (a->addr->sa_family == AF_INET) {
        inet_ntop(AF_INET, &((struct sockaddr_in*)a->addr)->sin_addr, str, sizeof(str));
        std::cout << "IP地址: " << str << std::endl;

        inet_ntop(AF_INET, &((struct sockaddr_in*)a->netmask)->sin_addr, str, sizeof(str));
        std::cout << "网络掩码: " << str << std::endl;

        inet_ntop(AF_INET, &((struct sockaddr_in*)a->broadaddr)->sin_addr, str, sizeof(str));
        std::cout << "广播地址: " << str << std::endl;
    }
}
```

```cpp
//循环打印设备信息
void printInterfaceList(pcap_if_t* alldevs) {
    int n = 1;

    for (pcap_if_t* d = alldevs; d != NULL; d = d->next) {
        std::cout << n++ << "." << std::endl;

        if (d->description)
            std::cout << "(" << d->description << ")" << std::endl << std::endl;
        else
            std::cout << "(No description)\n";

        for (pcap_addr_t* a = d->addresses; a != NULL; a = a->next) {
            printAddressInfo(a);
        }
    }

    if (n == 1) {
        std::cout << "\nNo interfaces found!\n";
    }
}

pcap_if_t* getInterfaceList() {
    pcap_if_t* alldevs = nullptr;
    char errbuf[PCAP_ERRBUF_SIZE];

    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, nullptr, &alldevs, errbuf) == -1) {
        std::cerr << "Error in pcap_findalldevs_ex: " << errbuf << std::endl;
        return nullptr;
    }

    return alldevs;
}
```

```cpp
void initializeMACAddress(unsigned char* address, unsigned char value) {
    for (int i = 0; i < 6; i++) {
        address[i] = value;
    }
}
```

①void ARP_show(struct pcap_pkthdr* header, const u_char* pkt_data);

作用：显示捕获的 ARP 数据包的详细信息。

功能细节：

- 将 pkt_data 转换为 ARPFrame_t 结构指针。
- 从 ARP 数据包中提取源 IP、目的 IP、源 MAC 地址和目的 MAC 地址。
- 调用 Byte2Hex 将 MAC 地址转换为十六进制字符串并输出，调用 inet_ntoa 将 IP 地址转换为可读的字符串格式并显示。

②void printAddressInfo(const pcap_addr_t* a);

作用：显示网络接口的 IP 地址、子网掩码和广播地址。

功能细节：

- 检查地址族是否为 IPv4。
- 使用 inet_ntop 将 IP 地址、子网掩码和广播地址转换为字符串，并输出。

③void printInterfaceList(pcap_if_t* alldevs);

作用：循环输出所有网络接口的信息，包括接口的描述、IP 地址、子网掩码和广播地址。

功能细节：

- 遍历 alldevs 链表（存放所有接口信息）。

• 输出每个接口的描述和地址信息，通过调用 printAddressInfo 函数打印每个接口的 IP 地址等信息。

• 如果没有找到接口，则输出提示。

④pcap_if_t* getInterfaceList();

作用：获取所有可用网络接口的列表。

功能细节：

• 使用 pcap_findalldevs_ex 函数查找所有可用接口，返回接口信息链表指针 alldevs。

• 如果获取接口失败，则输出错误信息并返回 nullptr。

⑤void initializeMACAddress(unsigned char* address, unsigned char value);

作用：初始化 MAC 地址，将其设置为指定值。

功能细节：

• 将给定地址 address 的前 6 字节都设为 value，用于初始化或重置 MAC 地址。

```
154  void SET_ARP_HOST(ARPFrame_t& ARPFrame1, const char* ip) {
155      // 初始化帧头和ARP数据帧的MAC地址
156      initializeMACAddress(ARPFrame1.FrameHeader.DesMAC, 0x00); // 目的 MAC 地址全 0
157      initializeMACAddress(ARPFrame1.FrameHeader.SrcMAC, 0x1f); // 本机 MAC 地址设为 0x1f
158      initializeMACAddress(ARPFrame1.SendHa, 0x1f);             // 源 MAC 地址设为 0x1f
159      initializeMACAddress(ARPFrame1.RecvHa, 0x00);             // 目标 MAC 地址全 0
160
161      // 设置以太网和 ARP 协议的基本字段
162      ARPFrame1.FrameHeader.FrameType = htons(0x0806);          // 帧类型设为 ARP（0x0806）
163      ARPFrame1.HardwareType = htons(0x0001);                   // 硬件类型设为以太网（0x0001）
164      ARPFrame1.ProtocolType = htons(0x0800);                   // 协议类型设为 IPv4（0x0800）
165      ARPFrame1.HLen = 6;                                       // 硬件地址长度为 6 字节
166      ARPFrame1.PLen = 4;                                       // 协议地址长度为 4 字节
167      ARPFrame1.Operation = htons(0x0001);                      // 操作类型设为请求（0x0001）
168      ARPFrame1.SendIP = inet_addr("192.168.120.110");          // 设置源 IP 地址（本地 IP）
169      ARPFrame1.RecvIP = inet_addr(ip);                         // 设置目标 IP 地址（参数传入的IP）
170  }
171
172  void SET_ARP_DEST(ARPFrame_t& ARPFrame, const char* ip, const unsigned char* mac) {
173      // 初始化目标帧的 MAC 地址
174      initializeMACAddress(ARPFrame.FrameHeader.DesMAC, 0xff); // 广播 MAC 地址（目的 MAC）
175      initializeMACAddress(ARPFrame.RecvHa, 0x00);              // 接收方 MAC 地址设为 0
176      memcpy(ARPFrame.FrameHeader.SrcMAC, mac, 6);              // 使用本地网卡 MAC 作为源 MAC
177      memcpy(ARPFrame.SendHa, mac, 6);                          // 源 MAC 地址也设为本地网卡 MAC
178
179      // 设置以太网帧和 ARP 协议的基本字段
180      ARPFrame.FrameHeader.FrameType = htons(0x0806);           // 帧类型设为 ARP
181      ARPFrame.HardwareType = htons(0x0001);                    // 硬件类型设为以太网
182      ARPFrame.ProtocolType = htons(0x0800);                    // 协议类型设为 IPv4
183      ARPFrame.HLen = 6;                                        // 硬件地址长度为 6 字节
184      ARPFrame.PLen = 4;                                        // 协议地址长度为 4 字节
185      ARPFrame.Operation = htons(0x0001);                       // 操作类型设为请求
186      ARPFrame.SendIP = inet_addr(ip);                          // 设置源 IP 地址（传入的参数IP）
187  }
```

这段程序定义了两个函数 SET_ARP_HOST 和 SET_ARP_DEST，用于初始化和设置 ARP 请求帧的字段，以便发送 ARP 请求数据包。SET_ARP_HOST 函数配置本地主机发送的 ARP 请求数据包，而 SET_ARP_DEST 函数配置目标主机的 ARP 数据包，包含本地和目标的 MAC 地址、IP 地址等信息。

SET_ARP_HOST：设置 ARP 请求的发送方信息，包括源和目的的 MAC 地址、IP 地址、硬件类型、协议类型等，以便模拟本机的 ARP 请求。

SET_ARP_DEST：设置目标主机的 ARP 请求信息，包含目标和源的 MAC 地址、IP 地址及 ARP 操作类型等，用于构建 ARP 请求帧，通常在发送 ARP 请求时设置目标主机信息。

```cpp
while ((k = pcap_next_ex(dev, &pkt_header, &pkt_data)) >= 0) { // 循环获取数据包
    // 发送ARP请求数据包
    pcap_sendpacket(dev, (u_char*)&ARPF_Send, sizeof(ARPFrame_t));

    struct ARPFrame_t* arp_message;
    arp_message = (struct ARPFrame_t*)(pkt_data); // 将接收到的数据包转为ARP帧结构
    if (k == 1) { // 如果成功读取到数据包

        // 检查是否为ARP响应包（以太网帧类型0x0806表示ARP，操作类型0x0002表示ARP响应）
        if (arp_message->FrameHeader.FrameType == htons(0x0806) && arp_message->Operation == htons(0x0002)) {
            cout << "ARP数据包：\n";
            ARP_show(header, pkt_data); // 打印ARP包的源和目标信息
            memcpy(mac, &(pkt_data[22]), 6); // 从数据包中提取目标主机的MAC地址
            cout << "本机MAC：" << *(Byte2Hex(mac, 6)) << endl; // 打印本机MAC地址
            break; // 接收到ARP响应后退出循环
        }
    }
}
```

这段代码实现了通过 ARP 协议发送和接收数据包的循环操作，直到成功接收到 ARP 响应包。主要流程包括发送 ARP 请求、判断接收到的是否为 ARP 响应数据包，以及在接收到目标主机的 MAC 地址时停止循环。这段代码用于发送 ARP 请求数据包并监听响应，当接收到目标设备的 ARP 响应包时，打印该 ARP 包的相关信息（如源 MAC 和 IP 地址、目的 MAC 和 IP 地址）并提取目标设备的 MAC 地址。

```cpp
while ((k = pcap_next_ex(dev, &pkt_header, &pkt_data)) >= 0) { // 不断捕获数据包

    pcap_sendpacket(dev, (u_char*)&ARPFrame, sizeof(ARPFrame_t)); // 发送ARP请求包
    struct ARPFrame_t* arp_message;
    arp_message = (struct ARPFrame_t*)(pkt_data); // 将接收到的数据包转换为ARP帧结构
    if (k == 0) continue; // 若无数据包则继续循环

    else // 若捕获到数据包则检查是否符合ARP响应包格式

        // 验证包类型是否为ARP响应包，且接收IP地址是否与目标一致
        if (arp_message->FrameHeader.FrameType == htons(0x0806)
            && arp_message->Operation == htons(0x0002)
            && *(unsigned long*)(pkt_data + 28) == ARPFrame.RecvIP) {

            cout << "ARP数据包：\n";
            ARP_show(header, pkt_data); // 打印ARP包中的源和目的信息
            memcpy(desmac, &(pkt_data[22]), 6); // 提取目标主机的MAC地址
            cout << "目的主机的MAC：" << *(Byte2Hex(desmac, 6)) << endl; // 打印目标主机的MAC地址
            break; // 成功接收响应包后退出循环
        }
}
```

这段代码通过 ARP 协议来发送请求并循环接收响应包。它不断发送 ARP 请求，直到收到包含目标主机 MAC 地址的 ARP 响应包。收到正确的 ARP 响应后，程序会打印响应包的详细信息，并提取和显示目标主机的 MAC 地址。

（2）运行结果及分析

```
F:\Users\lenovo\source\netwo
网络掩码: 255.255.0.0
广播地址: 169.254.255.255
12.
(Network adapter 'TAP-Windows Adapter V9' on local host)

IP地址: 169.254.113.85
网络掩码: 255.255.0.0
广播地址: 169.254.255.255


请选择发送数据包的网卡: 7
10.130.92.26
Network adapter 'MediaTek Wi-Fi 6 MT7921 Wireless LAN Card' on local host
ARP数据包:
源MAC地址:    3C-55-76-20-A6-71
源IP地址:     10.130.92.26
目的MAC地址: 00-00-5E-00-01-0D
目的IP地址   10.130.0.1

本机MAC: 3C-55-76-20-A6-71

请输入目的主机的IP地址: 10.130.54.167
ARP数据包:
源MAC地址:    AE-41-46-6B-EF-D3
源IP地址:     10.130.54.167
目的MAC地址: 3C-55-76-20-A6-71
目的IP地址   10.130.92.26

目的主机的MAC: AE-41-46-6B-EF-D3
请按任意键继续. . .
```

第一次发送 ARP 请求：

•源 MAC 地址：3C-55-76-20-A6-71 这是我的设备的 MAC 地址，表明请求是从我的设备（IP 地址：10.130.92.26）发出的。

•源 IP 地址：10.130.92.26 这是我设备的 IP 地址，ARP 请求包的源 IP。

•目的 MAC 地址：00-00-5E-00-01-0D 这是一个典型的 ARP 广播地址（00-00-5E-00-01-0D），表示 ARP 请求包的目标是一个网络上的特定主机，通常它会发送到网络上所有设备。

•目的 IP 地址：10.130.0.1 这是 ARP 请求中我希望查询的 IP 地址，也就是我的设备希望通过 ARP 协议询问“哪个主机拥有这个 IP 地址”。

•本机 MAC 地址：3C-55-76-20-A6-71 这是发送 ARP 请求时，使用的本机网卡的 MAC 地址。

第二次收到 ARP 响应：

•源 MAC 地址：AE-41-46-6B-EF-D3 这是响应 ARP 请求的主机（目的 IP 地址：10.130.54.167）的 MAC 地址。

•源 IP 地址：10.130.54.167 这是响应 ARP 请求的主机的 IP 地址。

•目的 MAC 地址：3C-55-76-20-A6-71 这是我的设备的 MAC 地址，表明 ARP 响应包是发给我的设备的。

•目的 IP 地址：10.130.92.26 这是我的设备的 IP 地址，表明 ARP 响应包的目标是我的设备。

•目的主机的 MAC 地址：AE-41-46-6B-EF-D3 这是目标主机（10.130.54.167）的 MAC 地址，响应包中返回了这个 MAC 地址，表明这是该 IP 地址对应的 MAC。

在发送的第一个 ARP 请求中，我的设备询问了 10.130.0.1 这个 IP 地址对应的 MAC 地址。

第二个 ARP 响应包中返回了目标 IP 地址 10.130.54.167 的 MAC 地址 AE-41-46-6B-EF-D3，这表明这个 MAC 地址属于 10.130.54.167 这个 IP 地址。我的设备成功地收到了 ARP 响应，并且能够获得目标 IP 地址 10.130.54.167 的 MAC 地址。

整体来说，我的 ARP 请求和响应流程是正常的，程序成功地获取了目标主机的 MAC 地址。

四、实验中遇到的问题以及感悟：

在完成实验的过程中，遇到的主要问题包括：首先，需要熟悉 NPcap 的数据包发送方法，这是在 IP 数据报捕获和分析实验的基础上进一步深入的内容。其次，在获取 IP 地址与 MAC 地址映射关系时，处理 ARP 请求和响应的过程复杂，需要准确地构建和解析 ARP 数据包。

通过本实验，我加深了对网络协议、ARP 机制和 NPcap 编程的理解，提升了对数据包捕获和分析的能力。实验的最大收获在于掌握了 NPcap 的使用方法和 ARP 协议的实现细节，为后续网络编程和协议分析打下了坚实的基础。