网络技术与应用实验报告

物联网工程 2211999 邢清画

一、实验名称

实验3——通过编程获取IP地址与MAC地址的对应关系

二、实验要求

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。

三、实验内容

1. 环境配置

1.1 NPcap 安装

在官网（https://npcap.com/#download）下载Npcap和Npcap-SDK

Downloading and Installing Npcap Free Edition

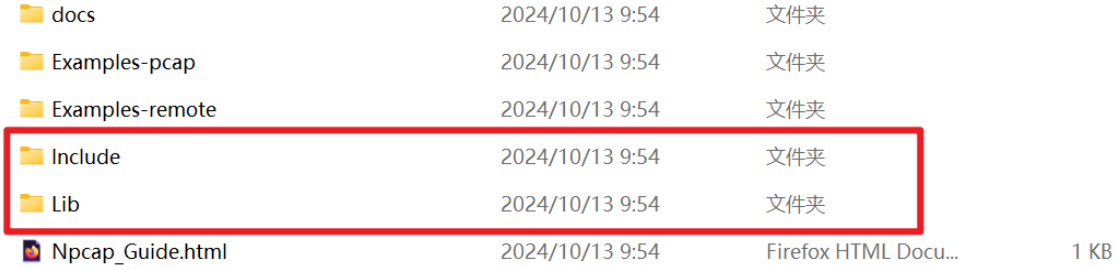
The free version of Npcap may be used (but not externally redistributed) on up to 5 systems (free license details). It may also be used on unlimited systems where it is only used with Nmap, Wireshark, and/or Microsoft Defender for Identity. Simply run the executable installer. The full source code for each release is available, and developers can build their apps against the SDK. The improvements for each release are documented in the Npcap Changelog.

- Npcap 1.80 installer for Windows 7/2008R2, 8/2012, 8.1/2012R2, 10/2016, 2019, 11 (x86, x64, and ARM64).
- Npcap SDK 1.13 (ZIP).
- Npcap 1.80 debug symbols (ZIP).
- Npcap 1.80 source code (ZIP).

The latest development source is in our Github source repository. Windows XP and earlier are not supported; you can use WinPcap for these versions.

第一步：安装npcap-1.80.exe，双击运行npcap-1.80.exe，一直点击Next直到完成安装。

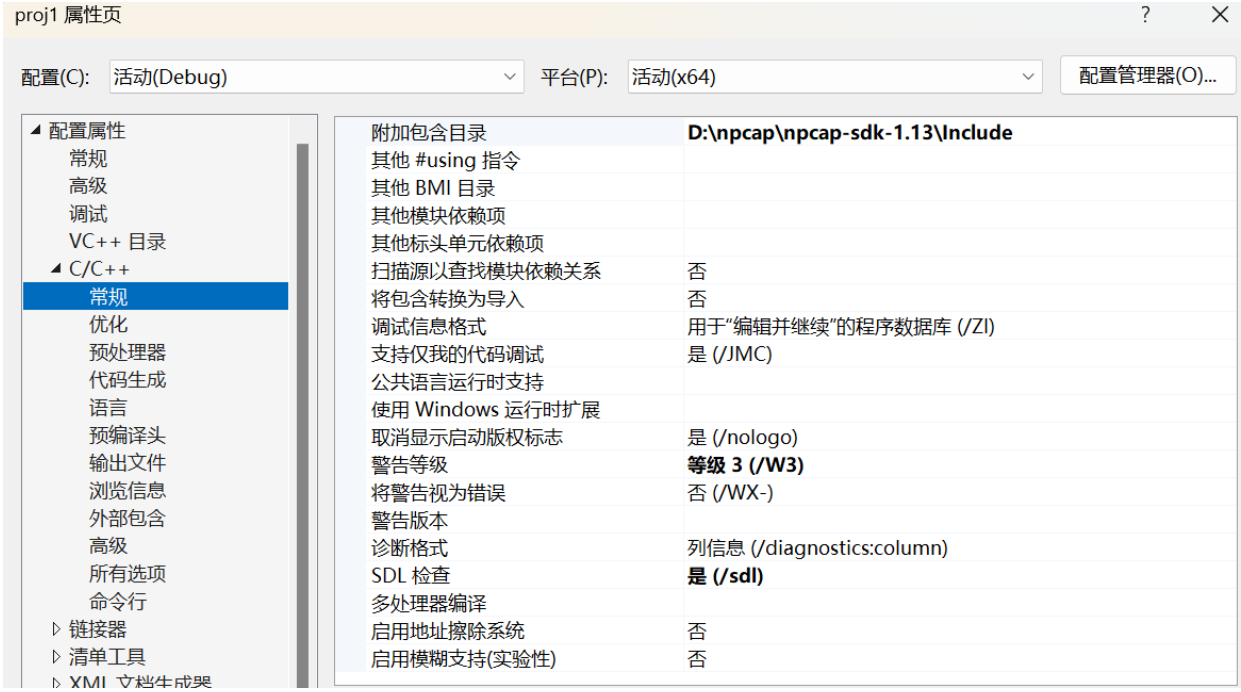
第二步：把Npcap SDK 1.13.zip解压，解压后如下图所示。



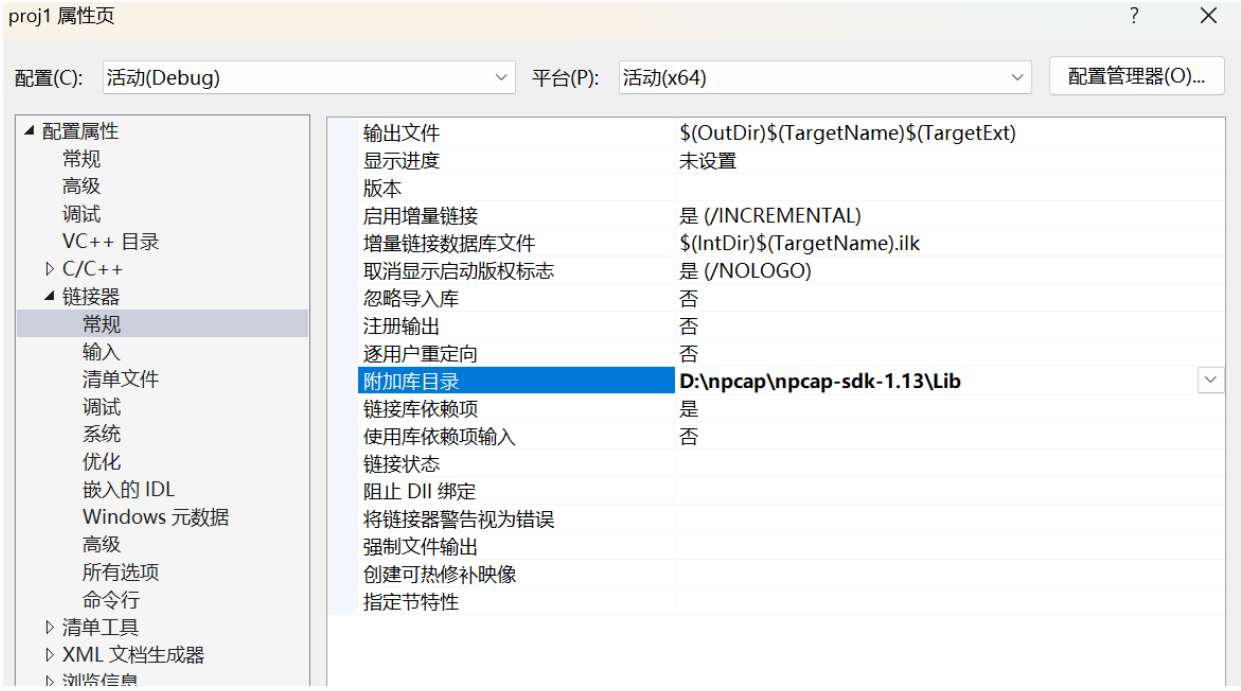
只需要上图所示的两个文件夹，拷贝至VS code工程路径下即可

1.2 项目设置

（1）项目→属性→C/C++→常规→附加包含目录：添加sdk中的Include目录



（2）项目→属性→链接器→常规→附加库目录：添加sdk中的Lib目录

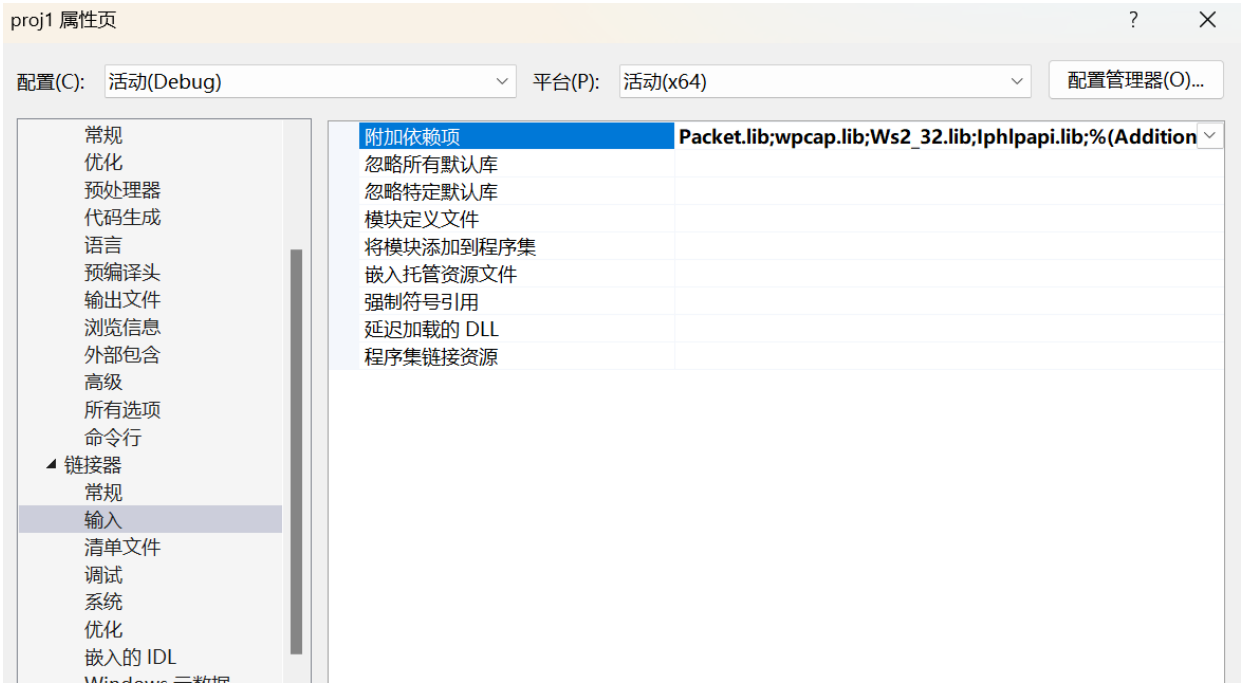


（3）项目→属性→链接器→输入→附加依赖项：添加
Packet.lib;wpcap.lib;Ws2_32.lib;Iphlpapi.lib;

`Packet.lib` 和 `wpcap.lib` 是 Npcap SDK 提供的库文件。

`Ws2_32.lib` 是 Windows 套接字库，提供网络相关的基本功能。

`Iphlpapi.lib` 用于访问 IP 帮助程序 API。

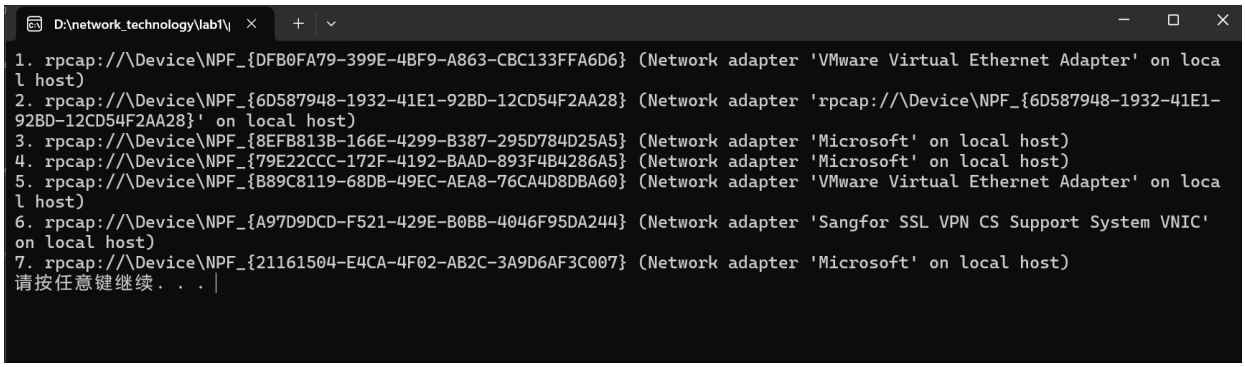


（4）输入官网的测试代码，若成功检测则环境配置成功

```
#include "pcap.h"

void main()
{
    pcap_if_t *alldevs;
    pcap_if_t *d;
    int i = 0;
    char errbuf[PCAP_ERRBUF_SIZE];
    /* Retrieve the device list from the local machine */
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL /* auth is not needed */,
&alldevs, errbuf) == -1)
    {
        fprintf(stderr, "Error in pcap_findalldevs_ex: %s\n", errbuf);
        exit(1);
    }
    /* Print the list */
    for (d = alldevs; d != NULL; d = d->next)
    {
        printf("%d. %s", ++i, d->name);
        if (d->description)
            printf(" (%s)\n", d->description);
        else
            printf(" (No description available)\n");
    }
    if (i == 0)
    {
        printf("\nNo interfaces found! Make sure Npcap is installed.\n");
        return;
    }
    /* We don't need any more the device list. Free it */
    pcap_freealldevs(alldevs);
```

```
    system("pause");
}
```



2. NPcap的数据包发送方法

NPcap 是一个 Windows 平台上用于数据包捕获和发送的网络驱动程序，通常用于网络分析工具（如 Wireshark）的底层支持。相比传统的 WinPcap，NPcap 提供了更高的性能、兼容性，并且支持 Windows 10 和更高版本。下面是 NPcap 数据包发送的方法以及涉及到的相关知识：

2.1 初始化：打开适配器

在使用 NPcap 发送数据包之前，首先需要初始化网络适配器。这通常是通过 `pcap_open()` 函数完成的，该函数可以打开指定的网络适配器并返回一个句柄，之后的操作都将基于这个句柄进行。

```
pcap_t *handle;
handle = pcap_open_live("eth0", 65536, 1, 1000, errbuf);
```

- `"eth0"` 是网卡接口名，可以替换成系统中实际的适配器名称。
- `65536` 是捕获的数据包的最大长度。
- `1` 表示是否设置为混杂模式。
- `1000` 是超时时间，单位为毫秒。

2.2 构建和组装数据包

NPcap 的数据包发送要求用户自己构建数据包，包括协议头部和有效载荷数据。构建数据包涉及到多种网络协议知识，常见的协议头部包括以太网头、IP 头、UDP/TCP 头等。

- 以太网头部:通常包含目标 MAC 地址、源 MAC 地址以及以太网类型（表示上层协议）。
- IP 头部:包括源 IP 地址、目标 IP 地址、协议字段（UDP、TCP、ICMP 等）以及总长度等信息。
- UDP/TCP 头部:传输层的 UDP 和 TCP 协议头部中包含源端口、目的端口等信息。

通常，数据包的构建可以通过指定相应的头部结构，逐一填充字段来完成。

```
unsigned char packet[42]; // 创建一个42字节的包
// 构造以太网头
struct ethhdr *eth = (struct ethhdr *)packet;
// 填充 MAC 地址等字段

// 构造 IP 头
```

```
struct iphdr *iph = (struct iphdr *)(packet + sizeof(struct ethhdr));
// 填充源IP、目标IP等字段

// 构造 UDP 头
struct udphdr *udph = (struct udphdr *)(packet + sizeof(struct ethhdr) +
sizeof(struct iphdr));
// 填充源端口、目标端口等字段
unsigned char packet[42]; // 创建一个42字节的包
// 构造以太网头
struct ethhdr *eth = (struct ethhdr *)packet;
// 填充 MAC 地址等字段

// 构造 IP 头
struct iphdr *iph = (struct iphdr *)(packet + sizeof(struct ethhdr));
// 填充源IP、目标IP等字段

// 构造 UDP 头
struct udphdr *udph = (struct udphdr *)(packet + sizeof(struct ethhdr) +
sizeof(struct iphdr));
// 填充源端口、目标端口等字段
```

2.3 发送数据包

在完成数据包的构建后，可以使用 `pcap_sendpacket()` 函数将数据包发送出去。该函数会将整个数据包按二进制数据发送到指定的网络适配器。

```
if (pcap_sendpacket(handle, packet, sizeof(packet)) != 0) {
    fprintf(stderr, "\nError sending the packet: %s\n", pcap_geterr(handle));
}
```

- `handle` 是之前使用 `pcap_open()` 打开的网络适配器句柄。
- `packet` 是要发送的数据包。
- `sizeof(packet)` 是数据包的长度。

2.4 关闭适配器和释放资源

完成数据包发送后，应当使用 `pcap_close()` 函数关闭适配器，释放资源。

```
pcap_close(handle);
```

2.5 相关知识点

- **混杂模式（Promiscuous Mode）**：在网络适配器上启用混杂模式允许捕获所有流经的网络数据包，而不仅限于发给本机的数据包。NPcap 默认支持混杂模式。
- **协议栈**：在 OSI 七层模型中，以太网、IP、UDP/TCP 分别位于不同的协议层。手动构造数据包时，需要注意不同协议层之间的结构依赖性，确保协议头部正确填充并按顺序排列。
- **检查和校验和（Checksum）**：为了保证数据包的完整性，通常需要计算 IP 和 UDP/TCP 头部的校验和。IP 头部校验和基于头部字段，而 UDP/TCP 校验和则包括源 IP、目标 IP、协议号、UDP/TCP 头部以及数据部分。

数据包发送的常见问题

1. **适配器权限**：在 Windows 上，部分适配器需要管理员权限才能发送数据包。
2. **防火墙**：Windows 防火墙可能会阻止某些数据包的发送，尤其是低级别的构造数据包。
3. **数据包格式**：错误的头部字段（如 MAC 地址、IP 地址、端口等）可能会导致数据包被丢弃或目标主机不响应。

在了解了基本方法和实现流程之后，接下来利用 NPcap 在 Windows 平台上实现**自定义数据包的发送**功能。

3. 数据包捕获与分析

3.1 获取设备列表

之前实验1的代码仅打印网卡信息，为了方便之后输入和检查网卡的IP地址等，完善之前的代码，补充打印**IP地址、子网掩码、广播地址**等。

利用 pcap 包的指针和 pcap_findalldevs 函数可以查找当前所有网卡设备，并输出查找到的所有设备信息。

```
// 获取IP地址的辅助函数
void* getaddress(struct sockaddr* sa) {
    if (sa->sa_family == AF_INET) {
        return &(((struct sockaddr_in*)sa)->sin_addr); // IPv4地址
    }
    return &(((struct sockaddr_in6*)sa)->sin6_addr); // IPv6地址
}
```

```
// 查找所有网络接口
    if (pcap_findalldevs(&alldevs, errbuf) == -1) {
        cerr << "查找设备失败: " << errbuf << endl;
        return -1;
    }

    // 遍历并打印所有网络接口的信息
    for (dev = alldevs; dev; dev = dev->next) {
        cout << ++i << ". 设备名称: " << dev->name << endl;
        if (dev->description) {
            cout << "   描述: " << dev->description << endl;
        }
        else {
            cout << "   描述: 无" << endl;
        }

        // 遍历该接口的所有地址
        for (addr = dev->addresses; addr != NULL; addr = addr->next) {
            if (addr->addr && addr->addr->sa_family == AF_INET) { // IPv4 地址
                char ip[INET_ADDRSTRLEN];
                char netmask[INET_ADDRSTRLEN];
                char broadaddr[INET_ADDRSTRLEN];
```

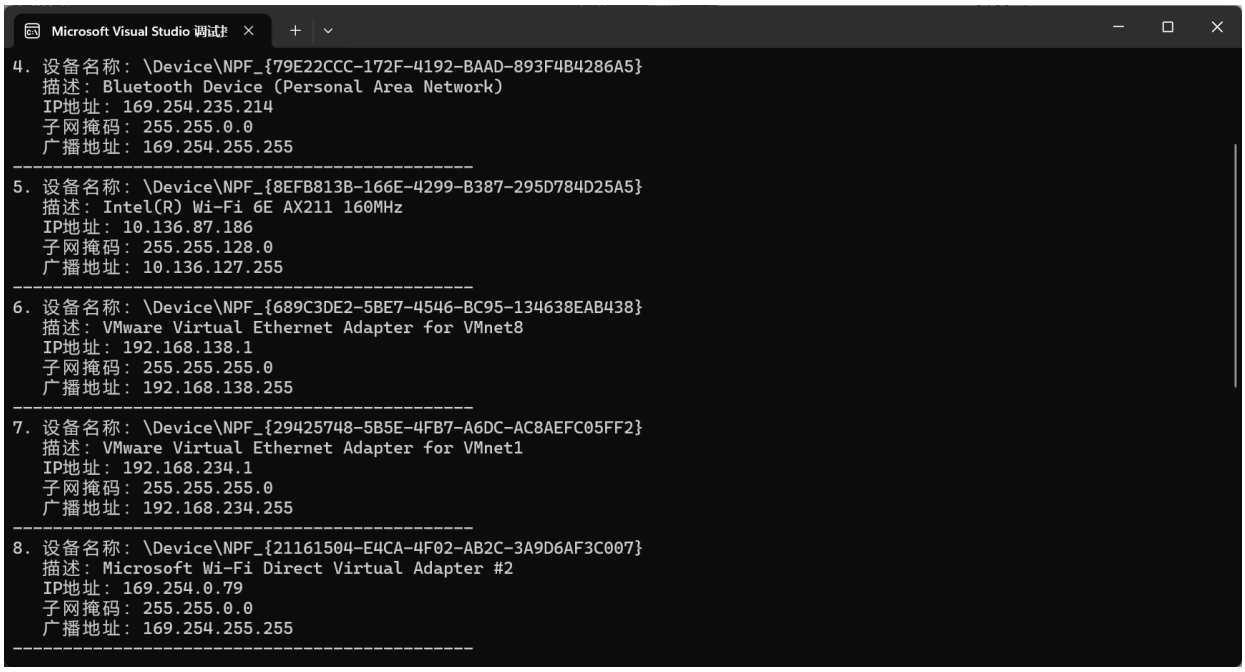
```
                // 获取并显示IP地址
                inet_ntop(AF_INET, getaddress(addr->addr), ip, sizeof(ip));
                cout << "   IP地址：" << ip << endl;

                // 获取并显示子网掩码
                if (addr->netmask) {
                    inet_ntop(AF_INET, getaddress(addr->netmask), netmask,
sizeof(netmask));
                    cout << "   子网掩码：" << netmask << endl;
                }
                else {
                    cout << "   子网掩码：无" << endl;
                }

                // 获取并显示广播地址
                if (addr->broadaddr) {
                    inet_ntop(AF_INET, getaddress(addr->broadaddr), broadaddr,
sizeof(broadaddr));
                    cout << "   广播地址：" << broadaddr << endl;
                }
                else {
                    cout << "   广播地址：无" << endl;
                }
            }
        }
        cout << "-----------------------------------------------" << endl;
    }

    // 检查是否找到任何设备
    if (i == 0) {
        cout << "未找到任何网络接口！" << endl;
    }
```

终端输出情况如下，显示所有网卡信息，便于后面实验对照检查。



同时在控制台输入ipconfig/all可以查看目前所有网卡的信息，与程序终端对比，结果保持一致。

```
C:\Users\lenovo>ipconfig/all

Windows IP 配置

   主机名  . . . . . . . . . . . . . : Huahua-LEGION
   主 DNS 后缀 . . . . . . . . . . . :
   节点类型  . . . . . . . . . . . . : 混合
   IP 路由已启用 . . . . . . . . . . : 否
   WINS 代理已启用 . . . . . . . . . : 否

无线局域网适配器 本地连接* 1:

   媒体状态  . . . . . . . . . . . . : 媒体已断开连接
   连接特定的 DNS 后缀 . . . . . . . :
   描述. . . . . . . . . . . . . . . : Microsoft Wi-Fi Direct Virtual Adapter
   物理地址. . . . . . . . . . . . . : 70-A8-D3-B0-77-A1
   DHCP 已启用 . . . . . . . . . . . : 是
   自动配置已启用. . . . . . . . . . : 是

无线局域网适配器 本地连接* 2:

   媒体状态  . . . . . . . . . . . . : 媒体已断开连接
   连接特定的 DNS 后缀 . . . . . . . :
   描述. . . . . . . . . . . . . . . : Microsoft Wi-Fi Direct Virtual Adapter #2
   物理地址. . . . . . . . . . . . . : 72-A8-D3-B0-77-A0
   DHCP 已启用 . . . . . . . . . . . : 否
   自动配置已启用. . . . . . . . . . : 是

以太网适配器 VMware Network Adapter VMnet1:

   连接特定的 DNS 后缀 . . . . . . . :
   描述. . . . . . . . . . . . . . . : VMware Virtual Ethernet Adapter for VMnet1
   物理地址. . . . . . . . . . . . . : 00-50-56-C0-00-01
   DHCP 已启用 . . . . . . . . . . . : 是
   自动配置已启用. . . . . . . . . . : 是
   本地链接 IPv6 地址. . . . . . . . : fe80::fc10:d70a:7adb:f098%6(首选)
   IPv4 地址 . . . . . . . . . . . . : 192.168.234.1(首选)
   子网掩码  . . . . . . . . . . . . : 255.255.255.0
   获得租约的时间  . . . . . . . . . : 2024年11月17日 14:59:19
   租约过期的时间  . . . . . . . . . : 2024年11月17日 20:59:19
   默认网关. . . . . . . . . . . . . :
   DHCP 服务器 . . . . . . . . . . . : 192.168.234.254
   DHCPv6 IAID . . . . . . . . . . . : 318787670
   DHCPv6 客户端 DUID  . . . . . . . : 00-01-00-01-2A-6F-A7-87-9C-2D-CD-06-C7-05
   TCPIP 上的 NetBIOS  . . . . . . . : 已启用

以太网适配器 VMware Network Adapter VMnet8:

   连接特定的 DNS 后缀 . . . . . . . :
   描述. . . . . . . . . . . . . . . : VMware Virtual Ethernet Adapter for VMnet8
   物理地址. . . . . . . . . . . . . : 00-50-56-C0-00-08
   DHCP 已启用 . . . . . . . . . . . : 是
   自动配置已启用. . . . . . . . . . : 是
```

3.2 网卡连接

枚举本机的网络设备（网卡）并选择目标设备：

- 通过遍历设备列表，允许用户选择一个网络接口用于后续的数据包捕获和发送。
- 用户输入一个序号，程序根据序号选定目标设备。

获取并显示目标设备的基本信息：

- 例如设备的 IP 地址和设备名称。

打开目标网络接口：

- 使用选定的设备初始化一个捕获会话（`pcap_t*`），以便进行数据包捕获和发送。

```
/*选择设备及打开网卡*/
    pcap_if_t* count2; //遍历用的指针2
    int num = 0;
    cout << "输入当前要连接的网卡序号：";
    cin >> num;
    while (num < 1 || num>11) {
        cout << "请检查网卡序号输入是否正确！" << endl;
        cout << "重新输入当前要连接的网卡序号：";
        cin >> num;
    }
    count2 = devices;
    for (int i = 1; i < num; i++) {//循环遍历指针选择第几个网卡
        count2 = count2->next;
    }
    inet_ntop(AF_INET, getaddress((struct sockaddr*)count2->addresses->addr), srcip,
sizeof(srcip));
    //将 a->addr 强制转换为 struct sockaddr_in 类型的指针，并访问 sin_addr 成员，其中包含了
IPv4 地址。
    cout << "当前网络设备接口卡IP为: " << srcip << endl << "当前网络设备接口卡名字为: " <<
count2->name << endl;
    //打开网络接口
    //指定获取数据包最大长度为65536,可以确保程序可以抓到整个数据包
    //指定时间范围为200ms
    pcap_t* point = pcap_open(count2->name, 65536, PCAP_OPENFLAG_PROMISCUOUS, 200,
NULL, errbuf);
    if (point == NULL) {
        cout << "打开当前网络接口失败" << endl;  //打开当前网络接口失败
        pcap_freealldevs(devices);
        return 0;
    }
    else {
        cout << "打开当前网络接口成功！！" << endl;
    }
```

实现方法：

用户选择网络接口：

- 利用 `pcap_findalldevs()` 获取所有设备的列表。用户通过输入网卡序号选择目标设备。

遍历设备列表：

- 遍历设备列表（`pcap_if_t`）直到找到与用户输入序号匹配的设备。

获取并打印设备信息：

- 通过访问设备结构体的字段，提取其 IP 地址（`a->addr->sa_family == AF_INET`）和设备名称（`count2->name`）。
- 使用 `inet_ntop` 将设备的 IP 地址从二进制格式转换为可读的字符串格式。

打开网络接口：

- 使用 `pcap_open()` 函数打开设备接口，设置捕获数据包的最大长度（65536 字节）和超时时间（200 毫秒）。
- 判断接口是否成功打开，如果失败，释放设备资源并提示用户。

在这一部分，由于我们要连接目标网卡，这里设置了一个**虚拟地址**来向网卡发送ARP数据包。

关键内容：

1. 设置虚拟地址信息

```
// 定义虚拟源IP和源MAC地址
    const char* virtualSrcIP = "192.168.1.1"; // 虚拟源IP地址
    const unsigned char virtualSrcMAC[6] = { 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF };
// 虚拟源MAC地址
```

2. 构造 ARP 请求数据包

- 将目标 MAC 地址设置为广播地址 `FF:FF:FF:FF:FF:FF`，表示所有设备都可以接收此请求。
- 源 MAC 地址设置为虚拟地址 `virtualSrcMAC`，用作请求的发送者地址。
- ARP 帧头中的类型字段设置为 `0x0806`，表示该帧是 ARP 数据包。
- 填充 ARP 请求的其他字段，如硬件类型（以太网）、协议类型（IPv4）、硬件地址长度（6 字节）、协议地址长度（4 字节）等。
- 设置操作类型为请求（`0x0001`），指定虚拟源 IP 地址为 `virtualSrcIP`，目标 IP 地址为 `192.168.138.1`。

此时源地址为我们设定的虚拟地址，目标地址为选中的网卡。

```
ARP_Frame send_ARPFrame;
    //获取本机的MAC地址
    // 组装报文
    // 构造ARP请求
    for (int i = 0; i < 6; i++) {
        send_ARPFrame.FrameHeader.DesMAC[i] = 0xFF; // 目标MAC设置为广播地址
        send_ARPFrame.FrameHeader.SrcMAC[i] = virtualSrcMAC[i]; // 虚拟源MAC地址
```

```
        send_ARPFrame.SrcMAC[i] = virtualSrcMAC[i]; // 虚拟源MAC地址
        send_ARPFrame.DesMAC[i] = 0x00; // 目标MAC地址未知
    }
    send_ARPFrame.FrameHeader.FrameType = htons(0x0806); // ARP帧类型
    send_ARPFrame.HardwareType = htons(0x0001); // 硬件类型为以太网
    send_ARPFrame.ProtocolType = htons(0x0800); // 协议类型为IPv4
    send_ARPFrame.HLen = 6; // 硬件地址长度
    send_ARPFrame.PLen = 4; // 协议地址长度
    send_ARPFrame.op = htons(0x0001); // 操作为ARP请求
    send_ARPFrame.SrcIP = inet_addr(virtualSrcIP); // 虚拟源IP地址
    send_ARPFrame.DesIP = inet_addr(srcip);// 动态设置目标 IP 地址为用户选择的网卡 IP 地址
```

使用 `pcap_sendpacket` 函数发送构造的 ARP 请求：

```
// 发送 ARP 请求数据包
cout << "加载中...";
pcap_sendpacket(point, (u_char*)&send_ARPFrame, sizeof(ARP_Frame));
```

3. 捕获数据包并解析

- 使用 `pcap_next_ex` 函数捕获网络中的数据包。
- 判断捕获到的数据包是否为 ARP 数据包（帧类型字段为 `0x0806`）。
- 进一步检查 ARP 操作字段是否为响应类型（`0x0002`）。
- 验证响应包中目标 IP 地址字段是否与请求中的目标 IP 地址一致。
- 如果匹配成功，解析并提取响应包中的源 MAC 地址（即目标设备的 MAC 地址）。

```
struct pcap_pkthdr* pkt_header; // 捕获的数据包头部
const u_char* packetData;       // 捕获的数据包内容

bool responseReceived = false;  // 标志是否接收到 ARP 响应
while (true) {
    int ret = pcap_next_ex(point, &pkt_header, &packetData);
    if (ret > 0) { // 捕获到数据包
        // 判断是否为 ARP 响应包
        if (*(unsigned short*)(packetData + 12) == htons(0x0806) && // 帧类型为 ARP
            *(unsigned short*)(packetData + 20) == htons(0x0002)) { // 操作类型为响应
            cout << endl;
            cout << "-----------------------------------------------" << endl;
            cout << "ARP 数据包内容: " << endl;

            // 打印虚拟源 IP 地址
            cout << "源 IP 地址:\t " << virtualSrcIP << endl;

            // 打印虚拟源 MAC 地址
            cout << "源 MAC 地址:\t ";
            for (int i = 0; i < 6; ++i) {
                printf("%02X", virtualSrcMAC[i]);
                if (i < 5) cout << "-";
            }
```

```
            cout << endl;

            // 打印目标 IP 地址
            cout << "目标 IP 地址:\t ";
            for (int i = 28; i < 32; ++i) {
                printf("%d", packetData[i]);
                if (i < 31) cout << ".";
            }
            cout << endl;

            // 提取目标 MAC 地址
            cout << "目标 MAC 地址:\t ";
            for (int i = 6; i < 12; ++i) {
                printf("%02X", packetData[i]);
                if (i < 11) cout << "-";
            }
            cout << endl;

            cout << "------------------------------------------------" << endl;
            responseReceived = true; // 设置响应标志为 true
            break;
        }
    }
    // 等待短时间，避免高 CPU 占用
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
}
```

4. 超时和循环控制

- 使用 `std::chrono` 获取当前时间，与开始等待时的时间进行对比。
- 如果捕获数据包的时间超过设定的最大等待时间（5 秒），则退出循环。
- 在循环中加入短暂延时 `std::this_thread::sleep_for`，降低 CPU 占用。

```
// 定义最大等待时间（单位：毫秒）
const int MAX_WAIT_TIME_MS = 5000; // 等待时间 5 秒
auto startTime = std::chrono::steady_clock::now(); // 获取当前时间作为起始时间

while (true) {
    // 检查是否超过最大等待时间
    auto currentTime = std::chrono::steady_clock::now();
    if (std::chrono::duration_cast<std::chrono::milliseconds>(currentTime -
startTime).count() > MAX_WAIT_TIME_MS) {
        break; // 超时退出循环
    }
    // 捕获数据逻辑在这里
}
```

如果未收到响应，提示用户检查网络配置。如果成功捕获响应包，输出结果并解析目标设备的 MAC 地址。

```
if (!responseReceived) {
    cout << endl;
    cout << "未能接收到 ARP 响应包，请检查目标 IP 或网络配置。" << endl;
} else {
    cout << "ARP 响应解析成功! 目标 MAC 地址已提取。" << endl;
}
```

具体关于ARP数据包的构造和发送在下面3.3部分陈述。

3.3 ARP数据包的构造和发送

1. ARP数据结构体和数据帧头部定义

```
typedef struct Frame_Header//帧首部
{
    BYTE DesMAC[6];  //目的地址
    BYTE SrcMAC[6];  //源地址
    WORD FrameType;  //帧类型
};
typedef struct ARP_Frame//ARP数据
{
    Frame_Header FrameHeader;
    WORD HardwareType; //硬件类型
    WORD ProtocolType; //协议类型
    BYTE HLen; //硬件长度
    BYTE PLen; //协议长度
    WORD op; //操作类型
    BYTE SrcMAC[6]; //源MAC地址
    DWORD SrcIP; //源IP地址
    BYTE DesMAC[6]; //目的MAC地址
    DWORD DesIP; //目的IP地址
};
```

2. 组装报文

利用ARP_Frame结构体定义ARP数据报内部的所有信息，例如目标IP地址和源IP地址等信息。

- 设置目标 MAC 地址为广播地址（`FF:FF:FF:FF:FF:FF`），表示请求会被发送到局域网内的所有设备。
- 使用本机（其实是刚刚选择的网卡）的 MAC 地址（通过 `mac` 数组）和 IP 地址（`srcip`）作为源信息。
- 设置目标 IP 地址为用户输入的 IP 地址。
- 填充 ARP 协议相关字段，包括硬件类型（以太网）、协议类型（IPv4）、地址长度、操作类型（请求）。

```
for (int i = 0; i < 6; i++) {
    rev_ARPFrame.FrameHeader.DesMAC[i] = 0xff; // 广播地址
    rev_ARPFrame.FrameHeader.SrcMAC[i] = mac[i]; // 本机 MAC 地址
    rev_ARPFrame.DesMAC[i] = 0x00; // 设置为 0
    rev_ARPFrame.SrcMAC[i] = mac[i]; // 本机 MAC 地址
}
```

```
rev_ARPFrame.FrameHeader.FrameType = htons(0x0806); // ARP 帧类型
rev_ARPFrame.HardwareType = htons(0x0001);          // 硬件类型为以太网
rev_ARPFrame.ProtocolType = htons(0x0800);          // 协议类型为 IPv4
rev_ARPFrame.HLen = 6;                              // 硬件地址长度
rev_ARPFrame.PLen = 4;                              // 协议地址长度
rev_ARPFrame.op = htons(0x0001);                    // 操作为 ARP 请求
rev_ARPFrame.SrcIP = inet_addr(srcip);              // 设置本机的源 IP 地址
cout << "请输入目的IP地址：";
char ip[INET_ADDRSTRLEN];
cin >> ip;
rev_ARPFrame.DesIP = inet_addr(ip);                 // 设置用户输入的目标 IP 地址
```

3. 发送 ARP 请求并设置重试机制

通过调用 `pcap_sendpacket` 发送 ARP 请求数据包。

设置重试机制，允许最多发送 `MAX_RETRIES` 次请求。每次发送请求后，重试计数器 `retryCount` 自增，最多允许重试 3 次。（避免等待时间不够）

```
int retryCount = 0; // 重试计数器
const int MAX_RETRIES = 3; // 最大重试次数

while (retryCount < MAX_RETRIES) {
    // 发送 ARP 请求
    pcap_sendpacket(point, (u_char*)&rev_ARPFrame, sizeof(ARP_Frame));
    cout << "已发送第 " << (retryCount + 1) << " 次 ARP 请求..." << endl;
```

```
--------------------------------------------
请输入目的IP地址：192.168.138.128
已发送第 1 次 ARP 请求...

--------------------------------------------
ARP数据包内容：
源IP地址：        192.168.138.1
源MAC地址：       00-50-56-C0-00-08
目的IP地址：      192.168.138.128
目的MAC地址：     00-0C-29-9B-04-F6
获取MAC地址成功，MAC地址为：00-0C-29-9B-04-F6
--------------------------------------------
已发送第 2 次 ARP 请求...

--------------------------------------------
ARP数据包内容：
源IP地址：        192.168.138.1
源MAC地址：       00-50-56-C0-00-08
目的IP地址：      192.168.138.128
目的MAC地址：     00-0C-29-9B-04-F6
获取MAC地址成功，MAC地址为：00-0C-29-9B-04-F6
--------------------------------------------
已发送第 3 次 ARP 请求...

--------------------------------------------
ARP数据包内容：
源IP地址：        192.168.138.1
源MAC地址：       00-50-56-C0-00-08
目的IP地址：      192.168.138.128
目的MAC地址：     00-0C-29-9B-04-F6
获取MAC地址成功，MAC地址为：00-0C-29-9B-04-F6
--------------------------------------------
```

4. 捕获 ARP 响应数据包

使用 `pcap_next_ex` 捕获网络中的数据包。解析捕获的数据包，检查是否符合 ARP 响应的特征：

1. 帧类型字段是否为 ARP（`0x0806`）。
2. 操作类型字段是否为响应（`0x0002`）。
3. 数据包中目标 IP 地址是否与请求中的目标 IP 地址一致。

```
    auto startTime = std::chrono::steady_clock::now();
    bool packetCaptured = false;

    while (true) {
        ret = pcap_next_ex(point, &pkt_header, &packetData);
        if (ret > 0) {
            // 判断是否为目标的 ARP 响应包
            if (*(unsigned short*)(packetData + 12) == htons(0x0806) &&
                *(unsigned short*)(packetData + 20) == htons(0x0002) &&
                *(unsigned long*)(packetData + 28) == rev_ARPFrame.DesIP) {
                cout << endl;
                cout << "-----------------------------------------------" << endl;
                cout << "ARP数据包内容：" << endl;
```

判断条件严格匹配目标的 ARP 响应包，确保捕获到的数据包是所需的响应。只有捕获到符合条件的包时，进入后续解析逻辑。

5. 解析并打印 ARP 数据包内容

捕获到的 ARP 响应中的目标 IP 地址位于数据包的第 28-31 字节。

使用条件 `*(unsigned long*)(packetData + 28) == rev_ARPFrame.DesIP` 确保解析到的响应对应用户输入的目标地址。

```
//打印数据包
                cout << "源IP地址:\t ";
                for (int i = 38; i < 42; ++i) {
                    printf("%d", packetData[i]);
                    if (i < 41) cout << ".";
                }
                cout << endl;
                // 提取MAC地址（0-6字节）
                cout << "源MAC地址:\t ";
                for (int i = 0; i < 6; ++i) {
                    printf("%02X", packetData[i]);
                    if (i < 5) cout << "-";
                }
                cout << endl;
                cout << "目的IP地址:\t ";
                for (int i = 28; i < 32; ++i) {
                    printf("%d", packetData[i]);
                    if (i < 31) cout << ".";
                }
```

```
                cout << endl;
                // 提取目的MAC地址（后6字节）
                cout << "目的MAC地址:\t ";

                for (int i = 6; i < 12; ++i) {
                    printf("%02X", packetData[i]);
                    if (i < 11) cout << "-";
                }
                cout << endl;
                cout << "获取MAC地址成功，MAC地址为：";
                for (int i = 6; i < 12; ++i) {
                    printf("%02X", packetData[i]);
                    if (i < 11) cout << "-";
                }
                cout << endl;
                cout << "----------------------------------------------" << endl;
                break;
```

四、结果分析

测试连接 VMware Virtual Ethernet Adapter for VMnet8

```
----------------------------------------------
输入当前要连接的网卡序号: 6
当前网络设备接口卡IP为: 192.168.138.1
当前网络设备接口卡名字为: \Device\NPF_{689C3DE2-5BE7-4546-BC95-134638EAB438}
打开当前网络接口成功！！
加载中...
```

输入网卡的序号，这里输入vmware虚拟机的网卡序号6，由虚拟地址向对应网卡发送信息，显示相对应网卡的信息以及IP地址和Mac地址。

```
输入当前要连接的网卡序号: 6
当前网络设备接口卡IP为: 192.168.138.1
当前网络设备接口卡名字为: \Device\NPF_{689C3DE2-5BE7-4546-BC95-134638EAB438}
打开当前网络接口成功！！
加载中...
----------------------------------------------
ARP数据包内容:
源IP地址:       192.168.1.1
源MAC地址:      AA-BB-CC-DD-EE-FF
目的IP地址:     192.168.138.1
目的MAC地址:    00-50-56-C0-00-08
获取MAC地址成功，MAC地址为：00-50-56-C0-00-08
----------------------------------------------
```

用windows的命令行ipconfig/all查询自己的IP地址和mac地址发现与抓捕到的mac地址相同

```
以太网适配器 VMware Network Adapter VMnet8:

   连接特定的 DNS 后缀 . . . . . :
   描述. . . . . . . . . . . . . . : VMware Virtual Ethernet Adapter for VMnet8
   物理地址. . . . . . . . . . . . : 00-50-56-C0-00-08
   DHCP 已启用 . . . . . . . . . . : 是
   自动配置已启用. . . . . . . . . : 是
   本地链接 IPv6 地址. . . . . . . . : fe80::8949:c14c:cd1a:b7a3%10(首选)
   IPv4 地址 . . . . . . . . . . . . : 192.168.138.1(首选)
   子网掩码 . . . . . . . . . . . . : 255.255.255.0
   获得租约的时间  . . . . . . . . . : 2024年11月17日 14:59:19
   租约过期的时间  . . . . . . . . . : 2024年11月17日 20:59:19
   默认网关. . . . . . . . . . . . . :
   DHCP 服务器 . . . . . . . . . . . : 192.168.138.254
   DHCPv6 IAID . . . . . . . . . . . : 419450966
   DHCPv6 客户端 DUID  . . . . . . . : 00-01-00-01-2A-6F-A7-87-9C-2D-CD-06-C7-05
   主 WINS 服务器  . . . . . . . . . : 192.168.138.2
   TCPIP 上的 NetBIOS  . . . . . . . : 已启用
```

在wireshark中抓包观察这一过程：

arp

No.	Time	Source	Destination	Protocol	Lengtl	Info
2	3.613156	aa:bb:cc:dd:ee:ff	Broadcast	ARP	42	Who has 192.168.138.1? Tell 192.168.1.1
3	3.613247	VMware_c0:00:08	aa:bb:cc:dd:ee:ff	ARP	42	192.168.138.1 is at 00:50:56:c0:00:08

由虚拟地址 192.168.1.1 发送数据包给 192.168.138.1，并显示对应的MAC地址，与终端信息一致。

输入目的地址 192.168.138.128

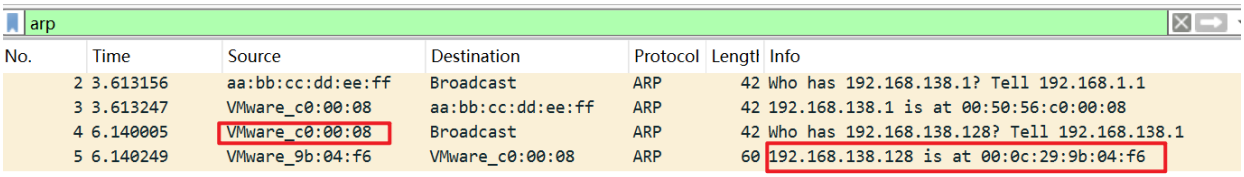
输入目的IP地址为 192.168.138.128，这是我在虚拟机中服务器的IP地址，可以用来检测是否能正确获取到mac地址，对应的MAC应当是 00-0C-29-9B-04-F6：

```
fzy@ubuntu: ~/Desktop
fzy@ubuntu:~/Desktop$ ip add show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gro
up default qlen 1000
    link/ether 00:0c:29:9b:04:f6 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.138.128/24 brd 192.168.138.255 scope global dynamic noprefixrou
te ens33
       valid_lft 1192sec preferred_lft 1192sec
    inet6 fe80::683e:e706:c51d:2ceb/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
fzy@ubuntu:~/Desktop$
```

观察终端输出结果：

```
------------------------------------------
ARP数据包内容：
源IP地址：          192.168.138.1
源MAC地址：         00-50-56-C0-00-08
目的IP地址：        192.168.138.128
目的MAC地址：       00-0C-29-9B-04-F6
获取MAC地址成功，MAC地址为：00-0C-29-9B-04-F6
------------------------------------------
```

观察wireshark中的对应数据包发送接收记录：



arp

No.	Time	Source	Destination	Protocol	Lengtl	Info
2	3.613156	aa:bb:cc:dd:ee:ff	Broadcast	ARP	42	Who has 192.168.138.1? Tell 192.168.1.1
3	3.613247	VMware_c0:00:08	aa:bb:cc:dd:ee:ff	ARP	42	192.168.138.1 is at 00:50:56:c0:00:08
4	6.140005	VMware_c0:00:08	Broadcast	ARP	42	Who has 192.168.138.128? Tell 192.168.138.1
5	6.140249	VMware_9b:04:f6	VMware_c0:00:08	ARP	60	192.168.138.128 is at 00:0c:29:9b:04:f6

测试通过。

五、实验总结

在本次实验中，我学习并掌握了以下知识点：

1. **ARP 协议的原理和实现：**
   - 了解了 ARP 协议的工作机制，包括通过广播请求获取目标设备 MAC 地址的流程。
   - 学会了如何构造符合 ARP 协议的请求数据包以及解析响应数据包。
2. **网络编程基础：**
   - 学习了如何使用 `Npcap` 库进行网络设备的枚举、数据包的发送与接收。
   - 通过 `pcap_sendpacket` 和 `pcap_next_ex` 实现了数据包的实时捕获与分析。
3. **数据解析与动态输入：**
   - 掌握了如何从捕获的数据包中提取关键信息（如 IP 和 MAC 地址）。
   - 学会了如何通过用户输入动态设置目标设备的 IP 地址，并将其嵌入到数据包中。

遇到的困难与解决办法

1. **ARP 响应匹配问题：**
   - **问题：**在捕获的数据包中，无法正确筛选出目标设备的 ARP 响应。
   - **解决办法：**通过严格判断帧类型、操作类型以及目标 IP 地址字段，确保捕获到的包是目标设备的响应包。
2. **重复发送导致的高频请求：**
   - **问题：**ARP 请求在未接收到响应时频繁发送，导致过多广播。
   - **解决办法：**引入延时机制与最大重试次数限制，避免频繁发送，并提高网络的容错性。
3. **用户输入的目标 IP 动态处理：**
   - **问题：**最初目标 IP 是硬编码的，缺乏灵活性。

- **解决办法**：通过用户输入动态绑定目标 IP，并在请求和响应处理中清晰标注目标地址。

自己的思考

本次实验让我深刻理解了网络通信的底层逻辑，尤其是局域网中设备间的地址解析过程。此外，通过使用 `Npcap` 进行编程，我意识到工具的选择和网络环境的配置对实验成功的重要性。在实验中，数据包解析的精确性尤为关键，一些细节如字节偏移和字段匹配需要反复调试，才能确保结果准确。

未来，我希望能将这次实验的经验应用到其他网络协议的实现与分析中，同时进一步研究网络安全方向，例如 ARP 欺骗攻击的检测与防御。