网络技术与应用第三次实验

- 实验名称：通过编程获取IP地址与MAC地址的对应关系
- 专业：物联网工程
- 姓名：秦泽斌
- 学号：2212005

一、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。

二、实验原理

1. NPcap的数据包发送方法

- 设备列表获取方法：

  NpCap 提供了pcap_findalldevs 函数来获取计算机上的网络接口设备的列表；此函数会为传入的 pcap_if_t 赋值（该类型是一个表示了设备列表的链表头；每一个这样的节点都包含了 name 和 description 域来描述设备）。

```
int pcap_findalldevs_ex(
    char * source;      //指定从哪儿获取网络接口列表
    struct pcap_rmauth auth;    //用于验证，由于是本机，置为NULL
    pcap_if_t ** alldevs;       //当该函数成功返回时，alldevs指向获取的列表数组的第一个
                                //列表中每一个元素都是一个pcap_if_t结构
    char * errbuf               //错误信息缓冲区
)
```

- 打开网络接口方法：

  NpCap 提供了pcap_open 函数于获取数据包捕获句柄以查看网络上的数据包。

```
pcap_t * pcap_open(
    const char *source;             //要打开的网卡的名字
    int snaplen,
    int flags,                      //指定以何种方式打开网卡，常用的有混杂模式
    int read_timeout,               //数据包捕获函数等待一个数据包的最大时间，超时则返回0
    struct pcap_rmauth *auth,
    char *errbuf
)
```

- 数据报捕获方法：

Npcap提供了 pcap_next_ex 函数。

```
int pcap_next_ex(
    pcap_t *p,//当为调用pcap_open()成功之后返回的值，它指定了捕获哪块网卡上的数据包
    struct pcap_pkthdr ** pkt_header,//捕获该数据包的时间戳、数据包的长度等等信息
    u_char ** pkt_data//捕获到的网络数据包
)
```

- 数据报发送方法：

  Npcap提供了pcap_sendpacket函数。

```
int pcap_sendpacket(
    pcap_t *p,// 一个已经打开的 pcap 句柄，表示一个网络适配器
    const u_char *buf,// 一个指向待发送数据包的缓冲区的指针
    int size// 待发送数据包的长度
);
```

2. ARP协议以及ARP包

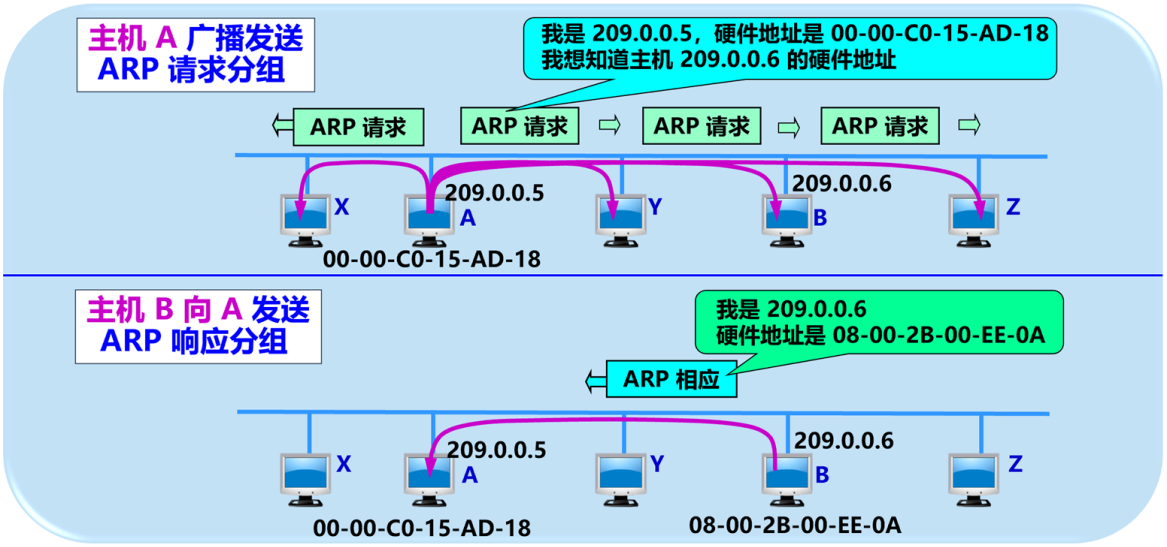
（1）ARP协议

ARP（Address Resolution Protocol）是一种用于解析网络层地址和链路层地址之间关系的协议。它的基本思想是将网络层地址（通常是IP地址）映射到链路层地址（通常是MAC地址）。

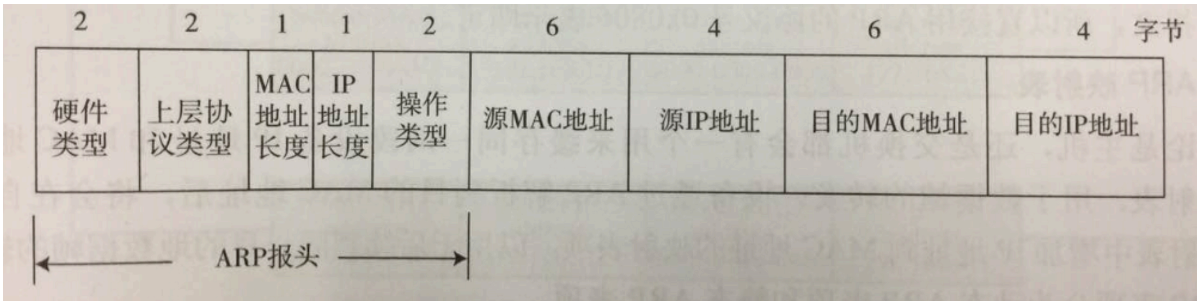
ARP根据IP地址获取物理地址的一个TCP/IP协 议。主机发送信息时将包含目标IP地址的ARP请求广播到局域网络上的所有主机，并接收返回消息，以 此确定目标的物理地址；收到返回消息后将该IP地址和物理地址存入本机ARP缓存中并保留一定时间， 下次请求时直接查询ARP缓存以节约资源。地址解析协议是建立在网络中各个主机互相信任的基础上 的，局域网络上的主机可以自主发送ARP应答消息，其他主机收到应答报文时不会检测该报文的真实性 就会将其记入本机ARP缓存。

ARP的基本工作原理如下：

> - **ARP请求：** 当设备A需要知道设备B的MAC地址时，设备A会在本地网络广播一个ARP请求，询问“谁拥有这个IP地址？”。
> - **ARP应答：** 设备B收到ARP请求后，会向设备A直接发送ARP应答，包含自己的MAC地址。
> - **ARP缓存：** 设备A收到ARP应答后，会将设备B的IP地址和MAC地址的映射关系存储在本地的ARP缓存中，以便将来的通信。
> - **ARP缓存定时过期：** ARP缓存中的映射关系是有时效性的，因为设备的网络连接可能会变化。因此，ARP缓存中的条目会有一个定时器，当超过一定时间后，会自动删除



(2) ARP包报文格式：



- 硬件类型：占两字节，表示ARP报文可以在哪种类型的网络上传输，值为1时表示为以太网地址。
- 上层协议类型：占两字节，表示硬件地址要映射的协议地址类型，映射IP地址时的值为0x0800。
- MAC地址长度：占一字节，标识MAC地址长度，以字节为单位，此处为6。
- IP协议地址长度：占一字节，标识IP得知长度，以字节为单位，此处为4。
- 操作类型：占2字节，指定本次ARP报文类型。1标识ARP请求报文，2标识ARP应答报文。
- 源MAC地址：占6字节，标识发送设备的硬件地址。
- 源IP地址：占4字节，标识发送方设备的IP地址。
- 目的MAC地址：占6字节，表示接收方设备的硬件地址，在请求报文中该字段值全为0，即00-00-00-00-00-00，表示任意地址，因为现在不知道这个MAC地址。
- 目的IP地址：占4字节，表示接受方的IP地址。

三、实验内容

1. 工具函数：MAC 和 IP 打印

```
void printMAC(BYTE MAC[6])
{
    for (int i = 0; i < 6; i++)
    {
        if (i < 5)
            printf("%02x:", MAC[i]);
        else
            printf("%02x", MAC[i]);
    }
};
```

```
void printIP(DWORD IP)
{
    BYTE* p = (BYTE*)&IP;
    for (int i = 0; i < 3; i++)
    {
        cout << dec << (int)*p << ".";
        p++;
    }
    cout << dec << (int)*p;
};
```

- `printMAC`：以 `XX:XX:XX:XX:XX:XX` 的格式打印 MAC 地址。
- `printIP`：以 `X.X.X.X` 的格式打印 IPv4 地址。

---

2. 数据结构定义

```
#pragma pack(1)

struct FrameHeader_t //帧首部
{
    BYTE DesMAC[6];  //目的地址
    BYTE SrcMAC[6];  //源地址
    WORD FrameType;  //帧类型
};

struct ARPFrame_t               //ARP帧
{
    FrameHeader_t FrameHeader;
    WORD HardwareType;
    WORD ProtocolType;
    BYTE HLen;
    BYTE PLen;
    WORD Operation;
    BYTE SendHa[6];
    DWORD SendIP;
    BYTE RecvHa[6];
    DWORD RecvIP;
};
```

- 使用 `#pragma pack(1)` 保证结构体对齐为 1 字节，避免额外填充字节影响数据包的封装和解析。
- `FrameHeader_t`：封装以太网帧首部，包含目的 MAC、源 MAC 和帧类型字段。
- `ARPFrame_t`：封装完整的 ARP 数据帧，包含硬件类型、协议类型、发送方和接收方的 IP 与 MAC 地址等信息。

---

3. 组装 ARP 请求帧

```
ARPFrame_t packetARP(const std::string& srcIP, const BYTE srcMAC[6], const
std::string& targetIP, BYTE targetMAC[6]) {
    ARPFrame_t ARPRequest;
    for (int i = 0; i < 6; i++) {
```

```
        ARPRequest.FrameHeader.DesMAC[i] = 0xFF;  // 广播地址
        ARPRequest.FrameHeader.SrcMAC[i] = srcMAC[i];
        ARPRequest.RecvHa[i] = 0;  // 目标MAC（尚未知）
        ARPRequest.SendHa[i] = srcMAC[i];
    }
    ARPRequest.FrameHeader.FrameType = htons(0x0806);  // ARP帧
    ARPRequest.HardwareType = htons(0x0001);  // 以太网
    ARPRequest.ProtocolType = htons(0x0800);  // IP
    ARPRequest.HLen = 6;  // MAC 长度
    ARPRequest.PLen = 4;  // IP 长度
    ARPRequest.Operation = htons(0x0001);  // ARP 请求
    ARPRequest.SendIP = inet_addr(srcIP.c_str());  // 源 IP
    ARPRequest.RecvIP = inet_addr(targetIP.c_str());  // 目标 IP
    return ARPRequest;
}
```

- 创建一个 ARP 请求帧，设置目标 MAC 地址为广播地址 `FF:FF:FF:FF:FF:FF`。
- 源 MAC 地址和 IP 地址使用用户输入的本地信息。
- **关键字段**：
  - `FrameType` 设置为 `0x0806` 表示 ARP。
  - `Operation` 设置为 `0x0001` 表示 ARP 请求。
  - `RecvIP`：目标 IP 地址。
  - `RecvHa`：目标 MAC 地址未知，置 0。

---

4. 发送与接收 ARP 包

```
void sendARPPacket(pcap_t* pcap_handle, const std::string& srcIP, const BYTE
srcMAC[6], const std::string& targetIP, BYTE targetMAC[6]) {
    ARPFrame_t ARPRequest;
    ARPRequest = packetARP(srcIP, srcMAC, targetIP, targetMAC);

    if (pcap_sendpacket(pcap_handle, (u_char*)&ARPRequest, sizeof(ARPFrame_t)) !=
-1) {
        cout << "ARP请求发送成功" << endl;
    }

    time_t lastPrintTime = time(nullptr);
    int printCount = 0;

    while (true) {
        time_t currentTime = time(nullptr);
        if (difftime(currentTime, lastPrintTime) >= 3) {
            cout << "正在接收ARP中......." << endl;
            lastPrintTime = currentTime;
            printCount++;
        }

        if (printCount >= 10) {
            cout << "接收失败！" << endl;
            return;
        }
```

```
        struct pcap_pkthdr* pkt_header;
        const u_char* pkt_data;
        int result = pcap_next_ex(pcap_handle, &pkt_header, &pkt_data);

        if (result == 1) {
            ARPFrame_t* ARPReply = (ARPFrame_t*)pkt_data;
            if (ARPReply->Operation == htons(0x0002)) {  // ARP 回复
                cout << "捕获到ARP回复包: " << endl;
                printIP(ARPReply->SendIP);
                cout << " <--> ";
                printMAC(ARPReply->SendHa);
                memcpy(targetMAC, ARPReply->SendHa, 6);  // 保存目标 MAC 地址
                return;
            }
        }
    }
}
```

- 使用 `pcap_sendpacket` 发送组装好的 ARP 包。
- 通过循环捕获 ARP 回复包，并验证操作码是否为 `0x0002`（ARP 回复）。
- 若未在 10 次打印周期内收到响应，退出程序。

---

6. 主函数工作流程

主函数是整个程序的核心，负责控制 **网络设备选择、适配器打开、ARP 包发送和捕获** 的流程。

6.1 获取本机网络设备列表

```
pcap_if_t* alldevs; // 存储设备列表的指针
char errbuf[PCAP_ERRBUF_SIZE]; // 错误信息缓冲区

if (pcap_findalldevs(&alldevs, errbuf) == -1)
{
    cerr << "获取网络接口时发生错误: " << errbuf << endl;
    return 1;
}
else { cout << "-------------以下为可用的网络适配器--------------" << endl << endl; }
```

- **作用**：通过 `pcap_findalldevs` 获取所有可用的网络适配器列表，并存储到 `alldevs`。
- 如果获取失败，程序会打印错误信息并退出。
- 如果成功，程序继续显示适配器信息。

6.2 显示网络设备并选择适配器

```
for (pcap_if_t* ptr = alldevs; ptr != nullptr; ptr = ptr->next)
{
    for (pcap_addr_t* a = ptr->addresses; a != nullptr; a = a->next)
    {
```

```
        if (a->addr->sa_family == AF_INET) // 只显示支持 IPv4 的适配器
        {
            interfaces[index] = ptr;
            cout << index + 1 << ". " << ptr->description << endl;
            cout << "IP地址: " << inet_ntoa(((struct sockaddr_in*)(a->addr))-
>sin_addr) << endl << endl;
            index++;
            break;
        }
    }
}
```

- **作用**：遍历获取到的网络设备列表，仅展示支持 IPv4 协议的网络接口。
- 每个设备的描述信息和其对应的 IP 地址会被打印到终端。
- 设备信息存储在数组 `interfaces` 中，便于后续访问。

6.3 设置过滤器（仅捕获 ARP 包）

```
u_int netmask;
netmask = ((sockaddr_in*)(interfaces[num - 1]->addresses->netmask))-
>sin_addr.S_un.S_addr;

bpf_program fcode;
char packet_filter[] = "ether proto \\arp";
if (pcap_compile(pcap_handle, &fcode, packet_filter, 1, netmask) < 0)
{
    cout << "无法编译数据包过滤器。检查语法";
    pcap_freealldevs(alldevs);
    return 0;
}
if (pcap_setfilter(pcap_handle, &fcode) < 0)
{
    cout << "过滤器设置错误";
    pcap_freealldevs(alldevs);
    return 0;
}
```

- **作用**：
    1. **获取子网掩码**：为过滤器设置提供网络掩码。
    2. **编译过滤规则**：
        - `packet_filter`：过滤规则，指定只捕获以太网协议类型为 ARP 的数据包。
        - `pcap_compile`：编译过滤规则。
    3. **设置过滤规则**：将编译后的规则应用到当前适配器上。
- 如果过滤器编译或设置失败，释放设备列表资源并退出程序。

6.4 发送伪造的 ARP 包并捕获响应

```
sendARPPacket(pcap_handle, CheatIP, cheatMAC, LocalIP, LocalMAC);
```

调用 `sendARPPacket` 函数，发送 ARP 包并捕获本地设备的 ARP 响应，获取 `LocalMAC`。

6.5 循环发送用户指定的 ARP 请求

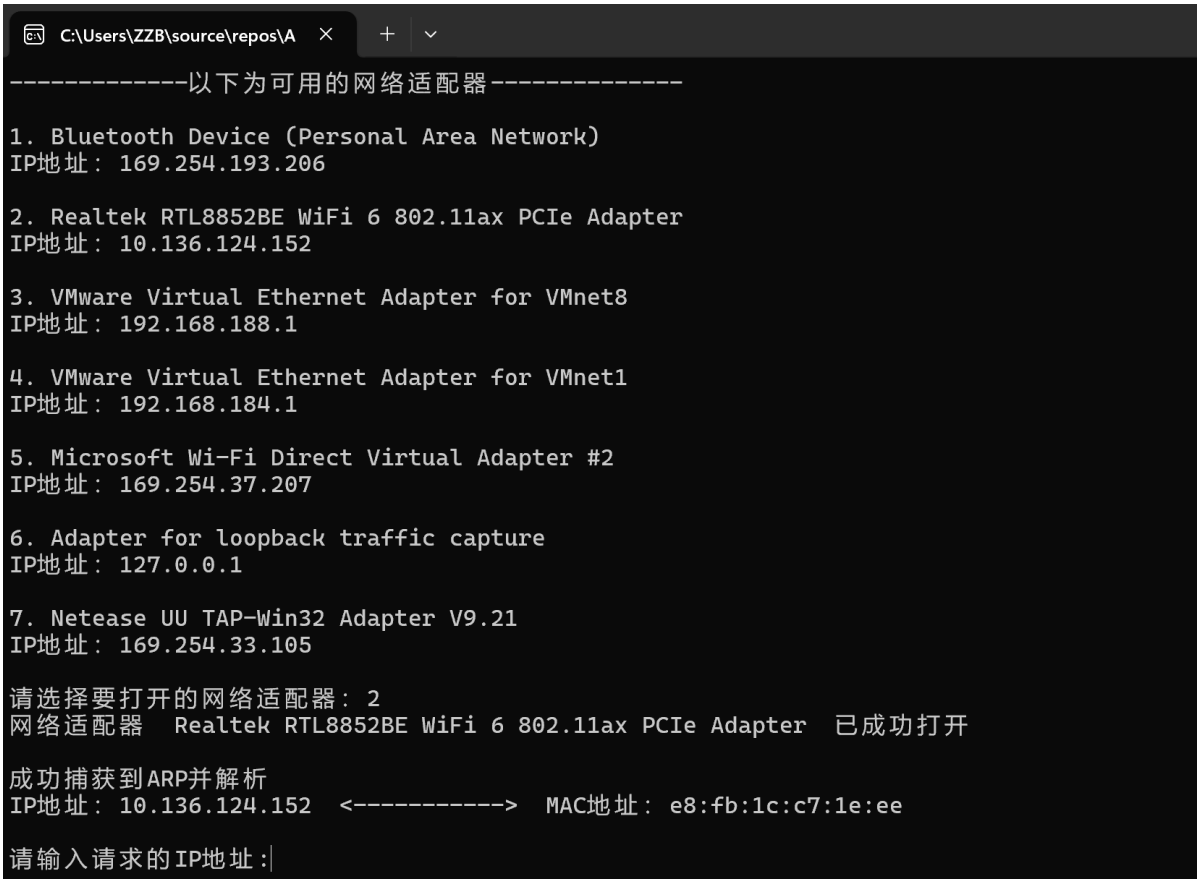
```
while (true) {
    cout << endl << "请输入请求的IP地址:";
    cin >> TargetIP;
    cout << endl;
    sendARPPacket(pcap_handle, LocalIP, LocalMAC, TargetIP, TargetMAC);
}
```

在获取到本机的IP和MAC后，继续调用`sendARPPacket`来广播发送ARP包，并且将该ARP包的源IP和源MAC设置为本机IP和本机MAC，同时由用户输入目标IP，这样在后续的捕获过程中就可以通过解析目标IP返回的ARP包解析出目标的MAC地址，达到实验目的。

四、实验结果

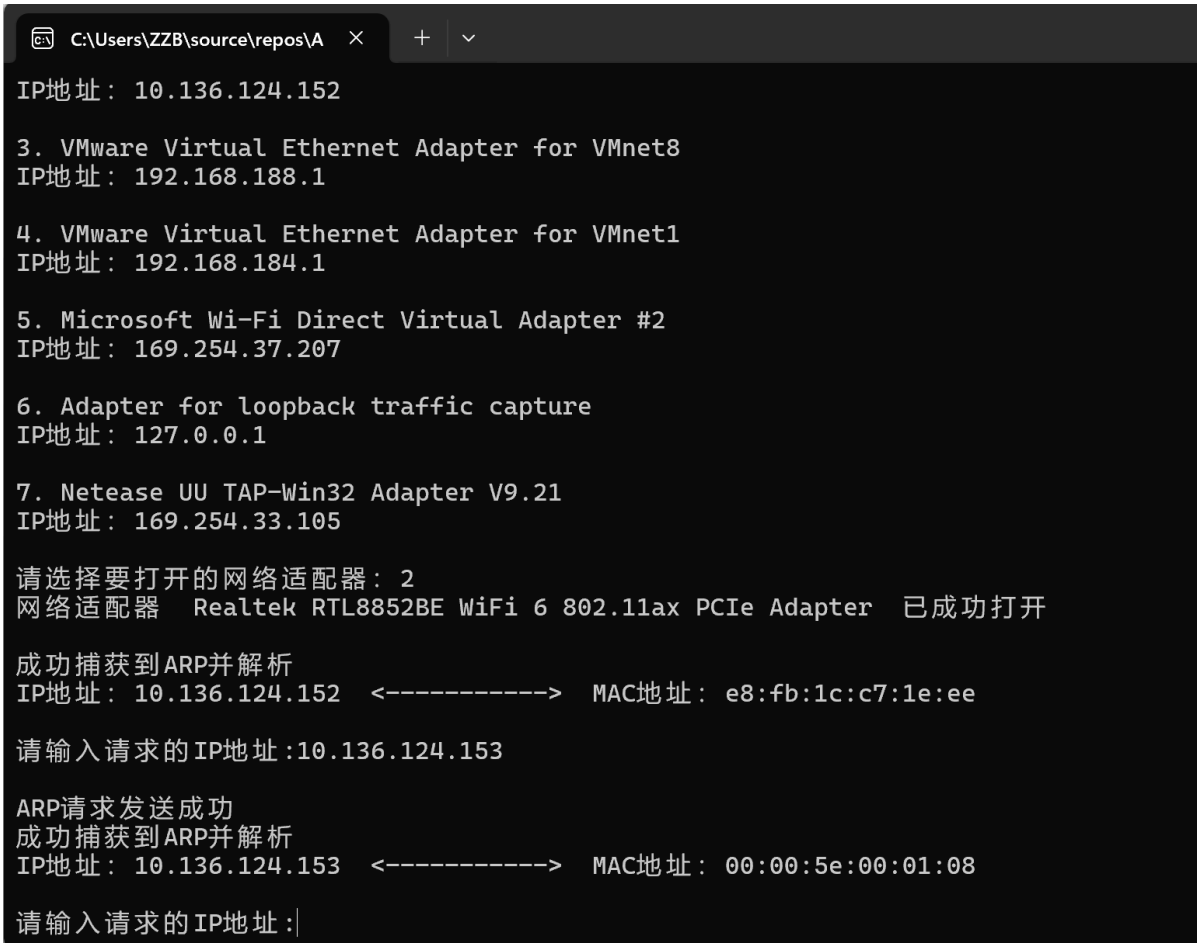
1. 显示可用的网络适配器并由用户选择

2. 选择任意网卡并获取本机IP地址与MAC地址的映射关系



在这里我们选择了WiFi网络适配器，IP地址为10.136.124.152，通过ARP协议解析其对应的MAC地址为e8:fb:1c:c7:1e:ee

3. 输入请求的IP地址，获取目标IP地址与MAC地址的映射关系



这里我们选择同一局域网下的10.136.124.153为解析目标，可以得到其对应的MAC地址为：00:00:5e:00:01:08