Lab3: 编程获取IP地址与MAC地址的对应关系

2110651周涛

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WinPcap（NPcap）的数据包发送方法。

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

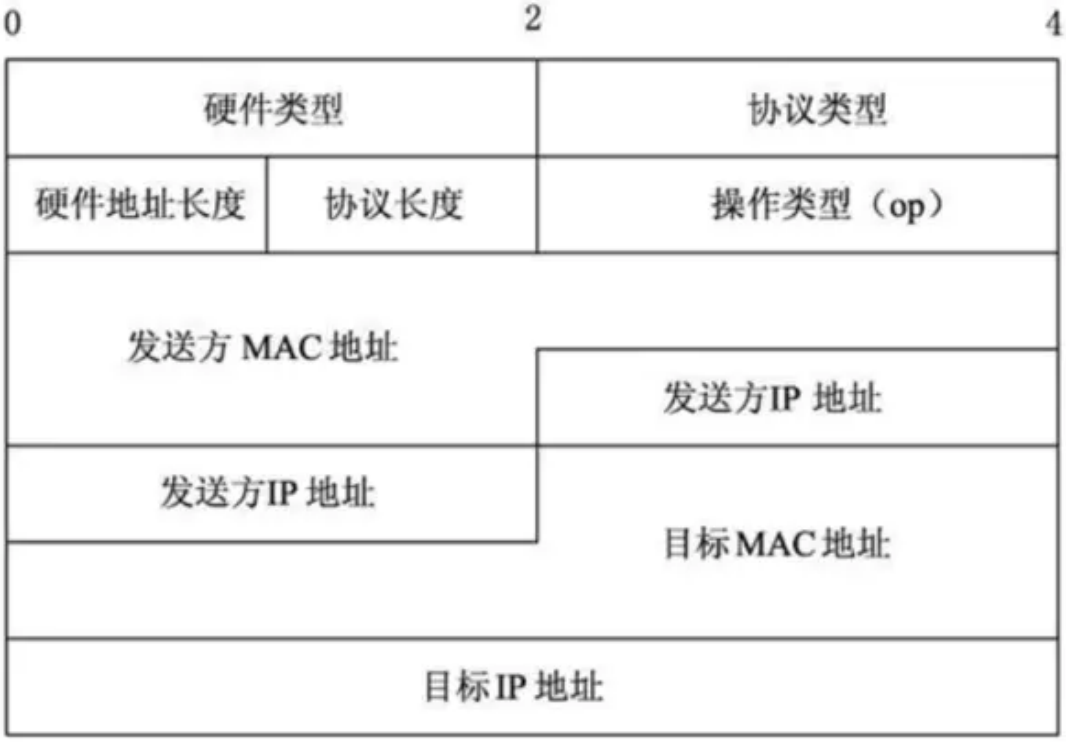
（3）程序要具有输入IP地址，显示输入IP地址与获取的MAC地址对应关系界面。界面可以是命令行界面，也可以是图形界面，但应以简单明了的方式在屏幕上显示。

（4）编写的程序应结构清晰，具有较好的可读性。

实验思路

前置知识

ARP报文格式：



ARP 报文总长度为 28 字节，MAC 地址长度为 6 字节，IP 地址长度为 4 字节。每个字段的含义：

- 硬件类型：指明了发送方想知道的硬件接口类型，以太网的值为 1。
- 协议类型：表示要映射的协议地址类型。它的值为 0x0800，表示 IP 地址。
- 硬件地址长度和协议长度：分别指出硬件地址和协议的长度，以字节为单位。对于以太网上 IP 地址的ARP请求或应答来说，它们的值分别为 6 和 4。
- 操作类型：用来表示这个报文的类型，ARP 请求为 1，ARP 响应为 2，RARP 请求为 3，RARP 响应为 4。
- 发送方 MAC 地址：发送方设备的硬件地址。
- 发送方 IP 地址：发送方设备的 IP 地址。
- 目标 MAC 地址：接收方设备的硬件地址。
- 目标 IP 地址：接收方设备的IP地址。

注：ARP数据包分为请求包和响应包，对应报文中有些字段值上的区别：

ARP 请求包报文的操作类型（op）字段的值为 request(1)，目标 MAC 地址字段的值为 Target 00：00：00：00：00：00广播地址。

ARP 响应包报文中操作类型（op）字段的值为 reply(2)，目标 MAC 地址字段的值为目标主机的硬件地址。

而由于广播发送只能在相同网段的网卡内进行发送，而主机上各个网卡的IP可能并不在同一网段，不能接收到对方的广播消息。所以本次实验必须使用相同网卡发出和响应ARP报文

设计思路

1. 获取网络接口卡的列表，选择需要捕获MAC地址的网卡
2. 构造ARP请求报文的内容：

ARP请求
广播
伪造源MAC地址和源IP地址
目的IP地址（所选网卡的IP地址）

3. 用选择的网卡进行报文发送
4. 对所选网卡进行流量监听，筛选其中的ARP报文（类型为0x806），捕获该网卡的ARP响应报文，在响应报文的**帧首部源MAC地址**部分可以看到发送该ARP响应的网卡对应的MAC地址。

关键代码分析

ARP报文格式

```
#pragma pack(1)//进入字节对齐方式 分配地址时没有空余
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;//帧首部
    WORD HardwareType;//硬件类型
    WORD ProtocolType;//协议类型
    BYTE HLen;//硬件地址
    BYTE PLen;//协议地址
    WORD Operation;//ARP操作码
    BYTE SendHa[6];//发送者硬件地址
    DWORD SendIP;//发送者IP
    BYTE RecvHa[6];//接收者硬件地址
    DWORD RecvIP;//接收者IP
};
```

获取设备列表并打印网卡信息，选择网卡打开

```
// 获得本机的设备列表
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &allDevices, errorBuffer)
== -1) {
        cout << "Error occurred while obtaining the network interface: " <<
errorBuffer << endl;
        return 0;
    }

    //显示接口列表
    for (currentDevice = allDevices; currentDevice != NULL; currentDevice =
currentDevice->next) {
        cout << "NETCard" << cardIndex + 1 << "\t" << currentDevice->description
<< endl;
```

```
        cardIndex++;
    }

    int selectedCard;
    cout << "Enter the number of the network card to be opened: ";
    cin >> selectedCard;
    currentDevice = allDevices;
    for (int i = 1; i < selectedCard; i++) {
        currentDevice = currentDevice->next;
    }

    pcap_t* pcapHandle = pcap_open(currentDevice->name, 1024,
PCAP_OPENFLAG_PROMISCUOUS, 1000, NULL, errorBuffer);
    if (pcapHandle == NULL) {
        cout << "An error occurred while opening the network card: " <<
errorBuffer << endl;
        return 0;
    } else {
        cout << "Successfully opened" << endl;
    }
```

设置过滤器，只要ARP包

```
// 设置过滤器
bool setFilter(pcap_t* pcap_handle, pcap_if_t* ptr) {
    u_int netmask;
    netmask = ((sockaddr_in*)(ptr->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 << "Unable to compile packet filter." << endl;
        return false;
    }

    if (pcap_setfilter(pcap_handle, &fcode) < 0) {
        cout << "Filter setting error." << endl;
        return false;
    }
    return true;
}

//调用过滤器函数
if (!setFilter(pcapHandle, currentDevice)) {
    pcap_freealldevs(allDevices);
    return 0;
}
```

组装报文

```
//组装报文
for (int i = 0; i < 6; i++)
{
ARPFrame.FrameHeader.DesMAC[i] = 0xFF;//设置为本机广播地址255.255.255.255.255.255
```

```
ARPFrame.FrameHeader.SrcMAC[i] = 0x88;//设置为虚拟的MAC地址6
ARPFrame.RecvHa[i] = 0;//设置为0
ARPFrame.SendHa[i] = 0x88;
}
ARPFrame.FrameHeader.FrameType = htons(0x0806);//帧类型为ARP
ARPFrame.HardwareType = htons(0x0001);//硬件类型为以太网
ARPFrame.ProtocolType = htons(0x0800);//协议类型为IP
ARPFrame.HLen = 6;//硬件地址长度为6
ARPFrame.PLen = 4; // 协议地址长为4
ARPFrame.Operation = htons(0x0001);//操作为ARP请求
SendIP = ARPFrame.SendIP = htonl(0x70707070);//源IP地址设置为虚拟的IP地址
```

发送报文，捕获流量

用选中的网卡发送报文，报文长度为**sizeof (ARPFrame_t)**，如果发送成功，返回0

```
//将所选择的网卡的IP设置为请求的IP地址
for (address = currentDevice->addresses; address != NULL; address = address-
>next) {
    if (address->addr->sa_family == AF_INET) {
        receiverIP = arpFrame.RecvIP = inet_addr(inet_ntoa(((struct
sockaddr_in*)(address->addr))->sin_addr));
    }
}

//发送报文
pcap_sendpacket(pcapHandle, (u_char*)&arpFrame, sizeof(ARPFrame_t));
cout << "ARP request sent successfully" << endl;

while (true) {
    int result = pcap_next_ex(pcapHandle, &packetHeader, &packetData);
    if (result == -1) {
        cout << "An error occurred while capturing the packet: " << errorBuffer
<< endl;
        return 0;
    } else {
        if (result == 0) {
            cout << "No datagrams captured" << endl;
        } else {
            ipPacket = (ARPFrame_t*)packetData;
            //筛选出本机的返回ARP数据包
            if (ipPacket->RecvIP == senderIP && ipPacket->SendIP == receiverIP)
{
                printIP(ipPacket->SendIP);
                cout << "->";
                printMAC(ipPacket->SendHa);
                cout << endl;
                break;
            }
        }
    }
}
```

获取远程网卡MAC地址

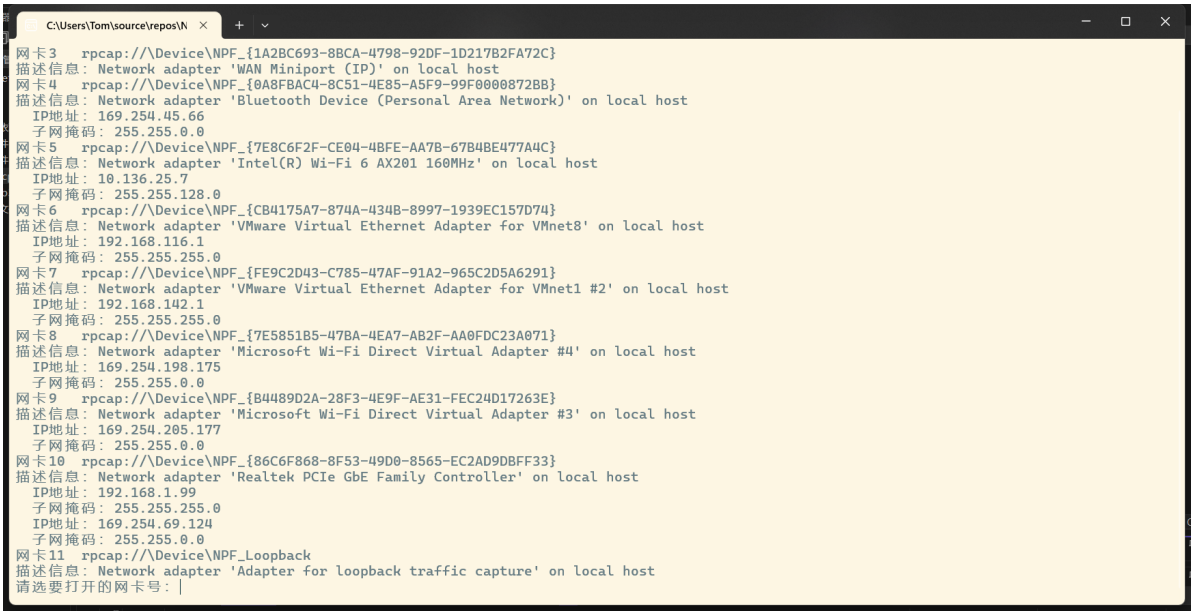
```
// 向网络发送数据包
```

```
cout << "Send a packet to the network, enter the requested IP address: ";
char ipAddress[32];
cin >> ipAddress;
receiverIP = arpFrame.RecvIP = inet_addr(ipAddress);
senderIP = arpFrame.SendIP = ipPacket->SendIP;

for (int i = 0; i < 6; i++) {
    arpFrame.SendHa[i] = arpFrame.FrameHeader.SrcMAC[i] = ipPacket->SendHa[i];
}

if (pcap_sendpacket(pcapHandle, (u_char*)&arpFrame, sizeof(ARPFrame_t)) != 0) {
    cout << "ARP request sent failure" << endl;
} else {
    cout << "ARP request sent successfully" << endl;

    while (true) {
        int result = pcap_next_ex(pcapHandle, &packetHeader, &packetData);
        if (result == -1) {
            cout << "An error occurred while capturing the packet: " <<
errorBuffer << endl;
            return 0;
        } else {
            if (result == 0) {
                cout << "No datagrams captured" << endl;
            } else {
                ipPacket = (ARPFrame_t*)packetData;
                if (ipPacket->RecvIP == senderIP && ipPacket->SendIP ==
receiverIP) {
                    cout << "The requested IP corresponds to its MAC address as
follows:" << endl;
                    printIP(ipPacket->SendIP);
                    cout << "->";
                    printMAC(ipPacket->SendHa);
                    cout << endl;
                    break;
                }
            }
        }
    }
}
pcap_freealldevs(allDevices);
```

封装ARP请求时使用本机网卡的IP和MAC地址，将本机IP和MAC填入报文，重新发送ARP请求，这样就会返回网关和远程主机的MAC地址。

实验结果

网卡列表获取



选择相应的网卡发送报文

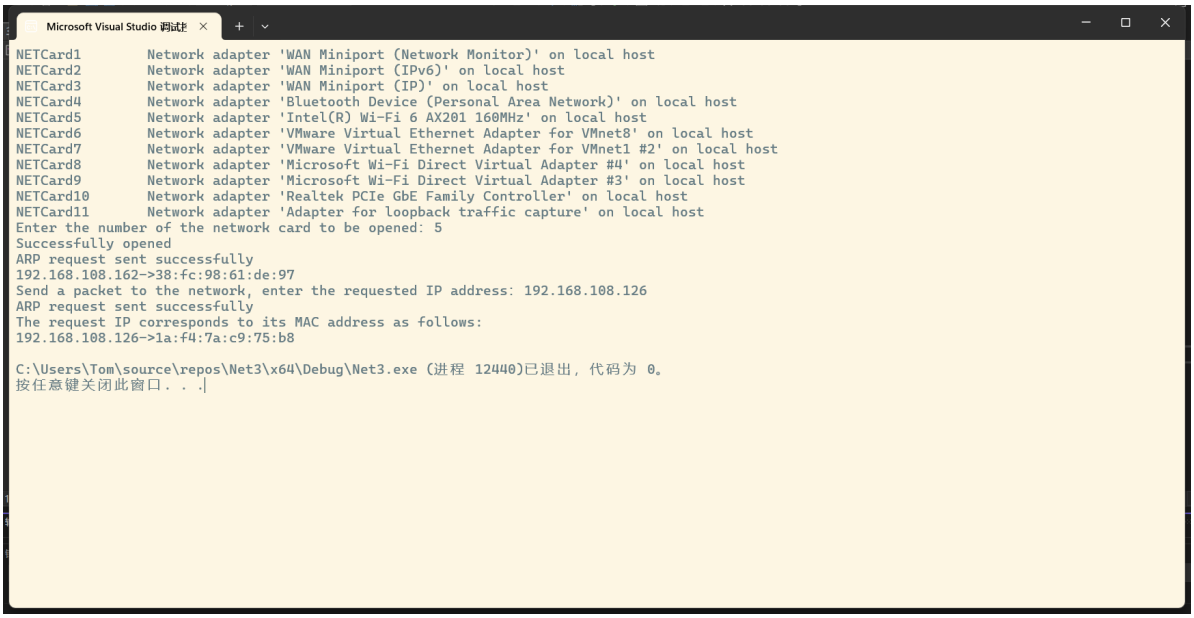
捕获到数据包请求IP与MAC地址的对应关系

与使用命令行**ipconfig/all**查询结果一致

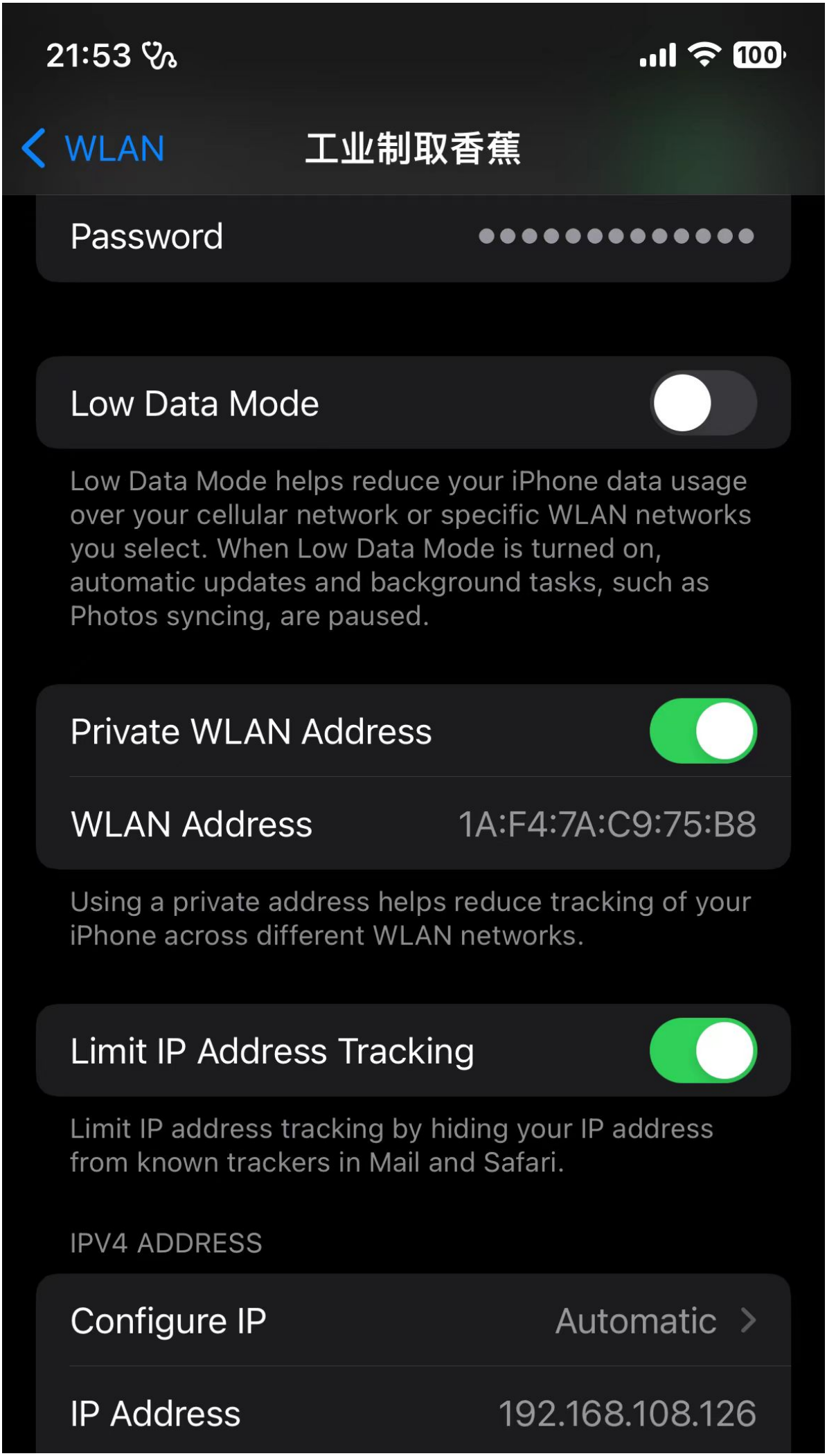
```
无线局域网适配器 WLAN:

   连接特定的 DNS 后缀 . . . . . :
   描述. . . . . . . . . . . . . . : Intel(R) Wi-Fi 6 AX201 160MHz
   物理地址. . . . . . . . . . . . : 38-FC-98-61-DE-97
   DHCP 已启用 . . . . . . . . . . : 是
   自动配置已启用. . . . . . . . . : 是
   IPv4 地址 . . . . . . . . . . . : 10.136.25.7(首选)
   子网掩码 . . . . . . . . . . . . : 255.255.128.0
   获得租约的时间 . . . . . . . . . : 2023年11月13日 18:47:53
   租约过期的时间 . . . . . . . . . : 2023年11月14日 2:29:35
   默认网关. . . . . . . . . . . . : 10.136.0.1
   DHCP 服务器 . . . . . . . . . . : 10.136.0.1
   DNS 服务器 . . . . . . . . . . . : 222.30.45.41
                                      202.113.16.41
   TCPIP 上的 NetBIOS . . . . . . . : 已启用
```

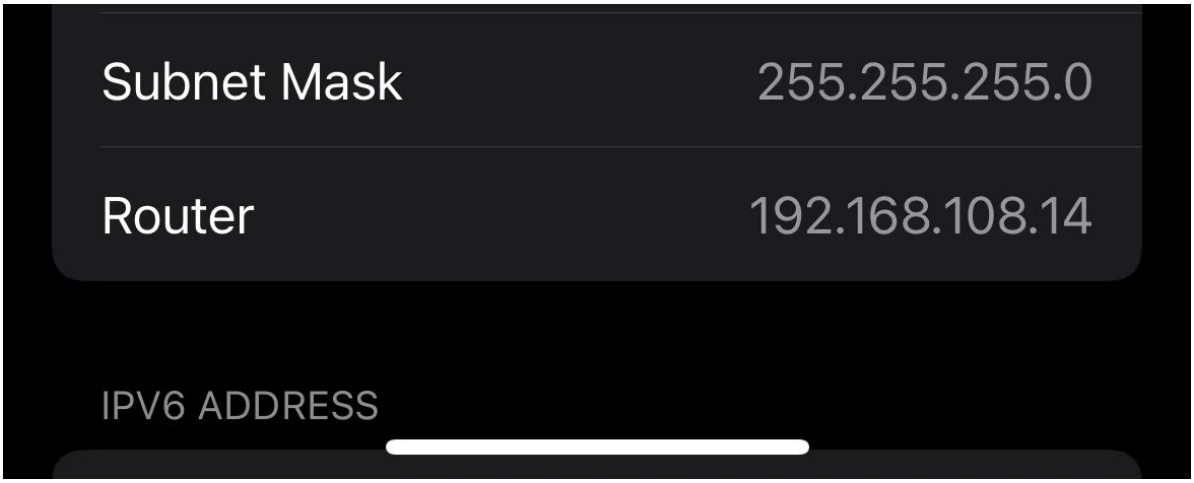
同一局域网下的其他设备

向连接于同一局域网的设备发送ARP数据包，获得其IP地址与MAC地址的关系



与设备mac地址一致


21:53
100
WLAN
工业制取香蕉
Password
Low Data Mode
Low Data Mode helps reduce your iPhone data usage over your cellular network or specific WLAN networks you select. When Low Data Mode is turned on, automatic updates and background tasks, such as Photos syncing, are paused.
Private WLAN Address
WLAN Address
1A:F4:7A:C9:75:B8
Using a private address helps reduce tracking of your iPhone across different WLAN networks.
Limit IP Address Tracking
Limit IP address tracking by hiding your IP address from known trackers in Mail and Safari.
IPV4 ADDRESS
Configure IP
Automatic
IP Address
192.168.108.126


Subnet Mask
255.255.255.0
Router
192.168.108.14
IPV6 ADDRESS