

# 网络技术与应用第三次实验实验报告

姓名：阿斯雅 学号：2210737

## 一、前期准备

### ARP 协议简介

ARP（地址解析协议，Address Resolution Protocol）是一个网络协议，用于将网络层地址（通常是IP地址）映射到链路层地址（通常是MAC地址）。ARP协议主要用于IPv4网络中，帮助设备在本地网络内通过IP地址找到目标设备的MAC地址，从而实现数据链路层的通信。

### ARP 工作原理

- ARP 请求

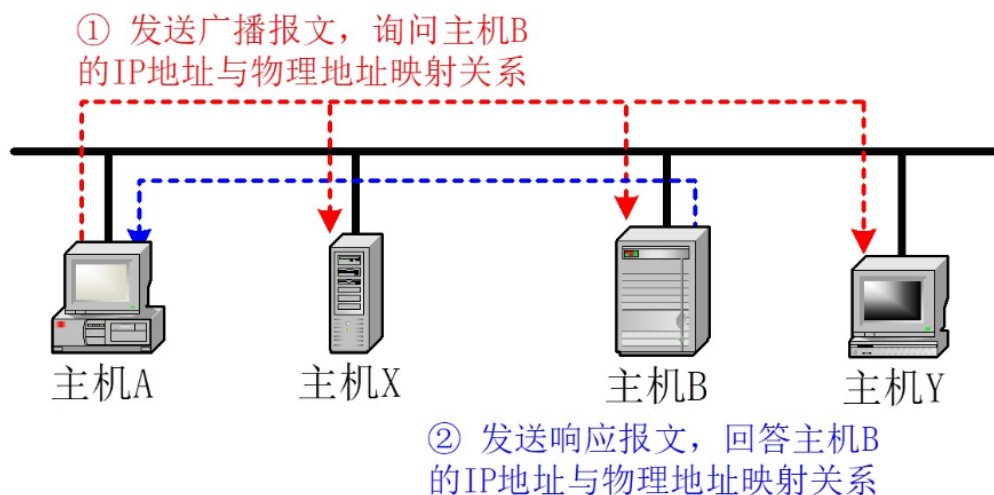
当一个设备（如计算机、路由器等）需要发送数据包到一个目标IP地址，但不知道目标的MAC地址时，它会广播一个ARP请求包。这个包包含源设备的IP地址和MAC地址，以及目标设备的IP地址。请求格式为：

- 目标IP地址：需要解析MAC地址的IP地址。
- 源IP地址：请求设备的IP地址。
- 源MAC地址：请求设备的MAC地址。
- 目标MAC地址：设为0，表示目标设备的MAC地址未知。

- ARP 响应

当目标设备接收到ARP请求后，它会检查请求中的目标IP地址是否与自己的IP地址匹配。如果匹配，目标设备会发送一个ARP响应包，回应请求设备自己的MAC地址。这个响应通常是单播的，即直接发送给请求者。响应格式为：

- 目标IP地址：与请求中的目标IP地址相同。
- 源IP地址：响应设备的IP地址。
- 源MAC地址：响应设备的MAC地址。
- 目标MAC地址：请求设备的MAC地址。



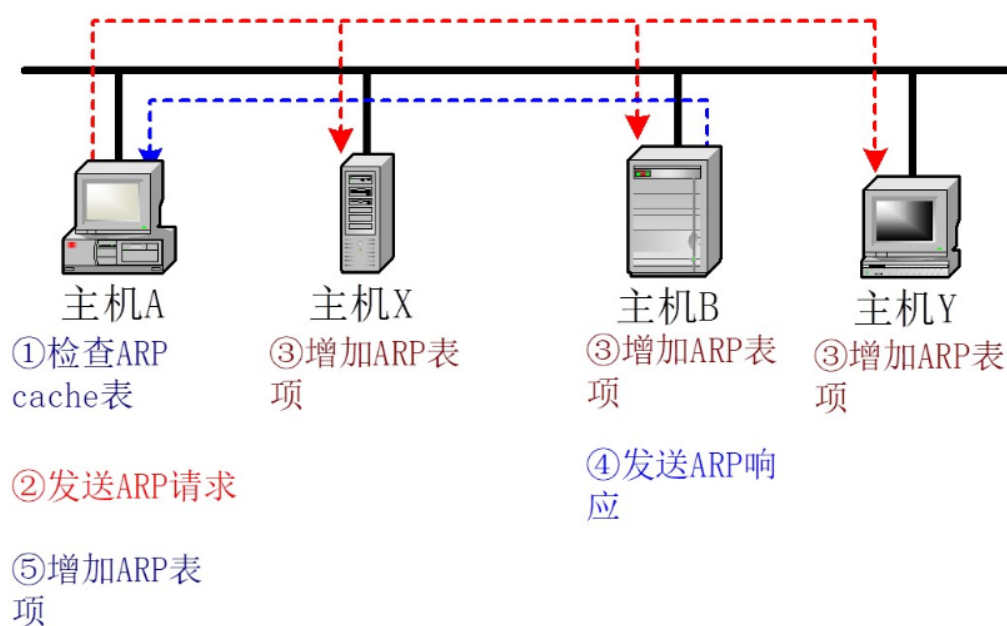
## ARP 报文格式

- 硬件类型：以太网接口类型为1
- 协议类型:IP协议类型为080016
- 操作:ARP请求为1,ARP应答为2
- 硬件地址长度:MAC地址长度为6B
- 协议地址长度:IP地址长度为4B
- 源MAC地址:发送方的MAC地址
- 源IP地址:发送方的IP地址
- 目的MAC地址:ARP请求中该字段没有意义;ARP响应中为接收方的MAC地址
- 目的IP地址:ARP请求中为请求解析的IP地址;ARP响应中为接收方的IP地址

0		15	16	31
硬件类型		协议类型		
硬件地址长度	协议地址长度	操作		
源MAC地址（0-3）				
源MAC地址（4-5）		源IP地址（0-1）		
源IP地址（2-3）		目的MAC地址（0-1）		
目的MAC地址（2-5）				
目的IP地址（0-3）				

## ARP 优化技术

- 高速缓存技术
  - 主机使用cache保存已知的ARP表项
  - 主机获得其他IP地址与物理地址映射关系后存入该cache
  - 发送时先检索cache,若找不到再利用ARP解析
  - 利用计时器保证cache中ARP表项的“新鲜性”
- 其他改进技术
  - 收到ARP请求后,目的主机将源主机的IP地址与物理地址的映射关系存入自己cache中
  - 广播发送的ARP请求,所有主机都会收到。这些主机可将该映射关系存入各自的cache
  - 主机启动时可主动广播自己IP地址与物理地址的映射关系



## 二、实验过程

本次实验过程可分为如下几个步骤：

- 打开指定网卡
- 获得本机网卡MAC地址
- 输入在同一局域网下的目标IP地址
- 获得目标MAC地址

接下来我将根据我的代码——解释。

### 打开指定网卡

首先我们可以根据实验一打开一个指定的网卡。

```
// 定义指向网络接口列表的指针
pcap_if_t* alldevs;
// 定义一个缓冲区，用于存储错误信息
char errbuf[PCAP_ERRBUF_SIZE];
```

```

// 获取所有可用的网络接口，如果失败，则打印错误信息并返回1
if (pcap_findalldevs(&alldevs, errbuf) == -1) {
    // 如果获取网络接口列表失败，输出错误信息
    cerr << COLOR_RED << "获取网络接口时发生错误: " << errbuf << COLOR_RESET << endl;
    return 1;
}

// 定义一个大小为100的数组来存储网络接口指针
array<pcap_if_t*, 100> interfaces{};
// 用于记录当前存储接口的索引
int index = 0;

// 遍历获取到的所有网络接口
for (pcap_if_t* ptr = alldevs; ptr; ptr = ptr->next) {
    // 遍历当前网络接口的所有地址
    for (pcap_addr_t* a = ptr->addresses; a; a = a->next) {
        // 检查地址类型是否为IPv4
        if (a->addr->sa_family == AF_INET) {
            // 如果是IPv4地址，保存该网络接口并打印接口信息
            interfaces[index++] = ptr;
            // 输出接口序号和接口描述
            cout << COLOR_MAGENTA << index << ". " << ptr->description <<
COLOR_RESET << endl;
            // 输出接口的IP地址
            cout << "      " << COLOR_CYAN << "IP地址: "
                << inet_ntoa(((struct sockaddr_in*)(a->addr))->sin_addr)
                << COLOR_RESET << endl;
            // 跳出当前地址的循环，只显示一个IP地址
            break;
        }
    }
}

// 提示用户选择一个网络适配器
int num;
cout << COLOR_CYAN << "请选择要打开的网络适配器: " << COLOR_RESET;
cin >> num;

// 判断用户输入的适配器序号是否合法
if (num <= 0 || num > index) {
    cerr << COLOR_RED << "无效序号" << COLOR_RESET << endl;
    return 1;
}

// 打开用户选择的网络适配器
pcap_t* pcap_handle = pcap_open(interfaces[num - 1]->name, 1024,
PCAP_OPENFLAG_PROMISCUOUS, 1000, nullptr, errbuf);

// 如果打开适配器失败，输出错误信息并返回1
if (!pcap_handle) {
    cerr << COLOR_RED << "打开网络适配器时发生错误: " << errbuf << COLOR_RESET <<
endl;
    return 1;
}

// 成功打开适配器，输出成功信息

```

```
cout << COLOR_GREEN << "成功打开" << interfaces[num - 1]->description <<
COLOR_RESET << endl;
```

## 获得网卡MAC地址

因为我们并不知道打开的网卡的实际的MAC地址，但在 ARP 报文中是要求填写源MAC地址的。所以我们得想办法获得源MAC地址。其中一般有两种方法，一种是直接在终端中输入 `ipconfig/all` 指令就可以知道自己电脑上全部网卡的 IP 地址和MAC地址，而第二种则是通过欺骗的方式来获得源MAC地址，也就是我们自己设定一个虚拟的 IP 地址和MAC地址，然后因为又知道打开的网卡的 IP 地址，所以先利用 ARP 协议来获得网卡MAC地址。

```
void sendARPPacket(pcap_t* pcap_handle, const string& srcIP, const array<BYTE,
6>& srcMAC, const string& targetIP, array<BYTE, 6>& targetMAC) {
    // 创建一个ARP请求帧结构
    ARPFrame_t ARPRequest{};

    // 设置ARP帧的目标MAC地址为广播地址
    ARPRequest.FrameHeader.DesMAC.fill(0xFF); // 广播地址

    // 设置源MAC地址
    ARPRequest.FrameHeader.SrcMAC = srcMAC;
    ARPRequest.SendHa = srcMAC;

    // 设置以太网帧类型为ARP协议 (0x0806)
    ARPRequest.FrameHeader.FrameType = htons(0x0806);

    // 设置硬件类型为以太网 (0x0001)，协议类型为IPv4 (0x0800)
    ARPRequest.HardwareType = htons(0x0001);
    ARPRequest.ProtocolType = htons(0x0800);

    // 设置硬件地址长度为6 (以太网MAC地址长度)，协议地址长度为4 (IPv4地址长度)
    ARPRequest.HLen = 6;
    ARPRequest.PLen = 4;

    // 设置ARP操作码为请求 (0x0001)
    ARPRequest.Operation = htons(0x0001);

    // 设置源IP地址
    ARPRequest.SendIP = inet_addr(srcIP.c_str());

    // 设置目标IP地址
    ARPRequest.RecvIP = inet_addr(targetIP.c_str());

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

    // 捕获响应包
    struct pcap_pkthdr* pkt_header;
    const u_char* pkt_data;
```

```

// 持续接收数据包直到捕获到ARP响应
while (pcap_next_ex(pcap_handle, &pkt_header, &pkt_data) != -1) {
    // 解析数据包为ARP帧
    auto ARPReply = reinterpret_cast<const ARPFrame_t*>(pkt_data);

    // 判断收到的包是否是目标IP的ARP响应
    if (ARPReply->RecvIP == ARPRequest.SendIP && ARPReply->SendIP ==
        ARPRequest.RecvIP && ARPReply->Operation == htons(0x0002)) {
        // 捕获到有效的ARP响应
        cout << COLOR_YELLOW << "成功捕获到ARP并解析" << COLOR_RESET << endl;
        cout << COLOR_CYAN;
        // 输出发送方IP地址和MAC地址
        printIP(ARPReply->SendIP);
        cout << " -> ";
        printMAC(ARPReply->SendHa);
        cout << COLOR_RESET;

        // 将目标设备的MAC地址保存到 `targetMAC`
        targetMAC = ARPReply->SendHa;

        // 返回，结束函数
        return;
    }
}

// 如果没有捕获到有效的ARP响应，输出错误信息
cerr << COLOR_RED << "捕获数据包时发生错误" << COLOR_RESET << endl;
}

// 获取用户选择的网络接口的本地IP地址（从接口的第一个地址获取）
string LocalIP = inet_ntoa(((struct sockaddr_in*)(interfaces[num - 1]-
>addresses->addr))->sin_addr);

// 定义一个假的MAC地址，用于ARP欺骗
array<BYTE, 6> cheatMAC = { 0x66, 0x66, 0x66, 0x66, 0x66, 0x66 };

// 定义本地和目标设备的MAC地址
array<BYTE, 6> LocalMAC{};
array<BYTE, 6> TargetMAC{};

// 发送ARP请求包，假设虚拟IP地址为 "112.112.112.112"
sendARPPacket(pcap_handle, "112.112.112.112", cheatMAC, LocalIP, LocalMAC);

```

The screenshot shows a Windows command prompt window with two panes. The left pane displays the output of the 'ipconfig /all' command for the Intel(R) Wi-Fi 6E AX211 160MHz adapter, showing its IP address as 10.130.58.73 and its physical address as 84-7B-57-4C-36-8C. The right pane shows the output of the 'ipconfig /all' command for the same adapter, highlighting the physical address 84-7B-57-4C-36-8C and the DHCP server 10.130.0.1.

可以从实验结果知道欺骗获得的MAC地址是正确的。

## 获得目标MAC地址

在获得了源MAC地址后接下来就是输入一个IP地址，然后获得它的MAC地址。其中要注意的是输入的IP地址必须要和网卡的IP地址是在同一局域网下的，也就是相互能ping通。

```
// 定义一个字符串变量来存储目标IP地址
string TargetIP;

// 无限循环，等待用户输入目标IP地址
while (true) {
    // 提示用户输入目标IP地址
    cout << COLOR_CYAN << "请输入请求的IP地址: " << COLOR_RESET;
    cin >> TargetIP;

    // 调用 sendARPPacket 函数发送ARP请求，目标IP地址为用户输入的地址
    sendARPPacket(pcap_handle, LocalIP, LocalMAC, TargetIP, TargetMAC);
}

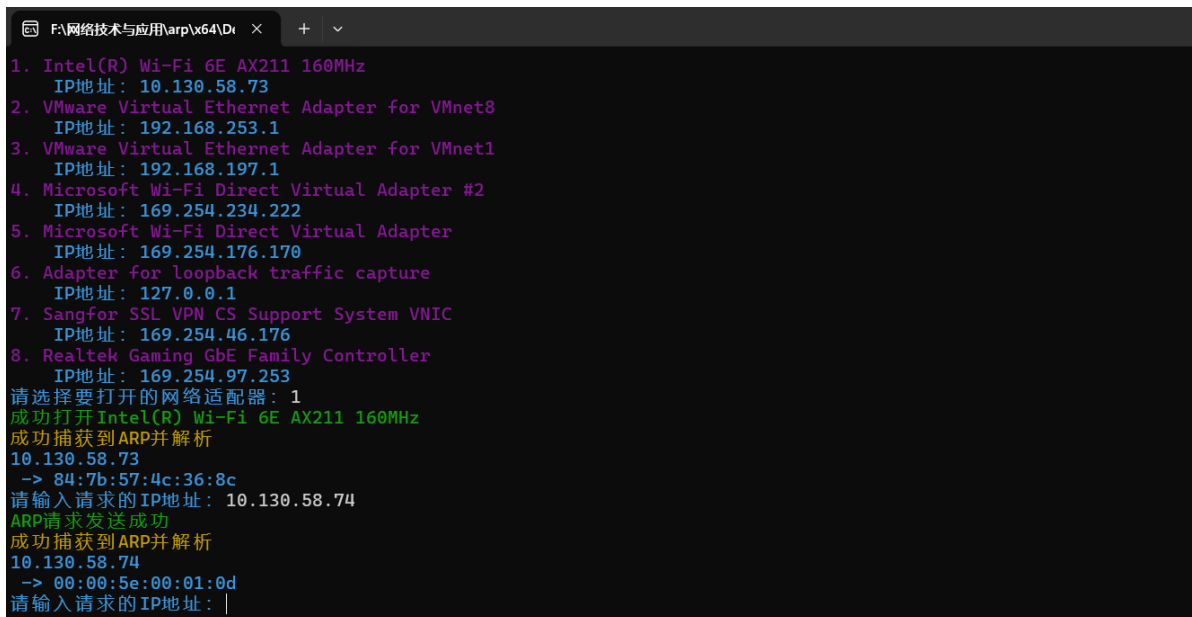
// 释放所有网络接口资源
pcap_freealldevs(alldevs);

// 关闭捕获句柄
pcap_close(pcap_handle);
```

## 验证实验结果

要想验证获得的MAC地址是否正确，我们可以在终端先ping通这个 IP 主机，然后在本地的 ARP 缓存表中就会保存该IP地址的IP地址和MAC地址的映射关系，我们使用 `arp -a` 就可以获得缓存表。然后比较即可。

输入 IP 地址 10.130.58.74 获得MAC地址 00:00:5e:00:01:0d



```
F:\网络技术与应用\arp\vx64\Dir > .\arp.exe
1. Intel(R) Wi-Fi 6E AX211 160MHz
   IP地址: 10.130.58.73
2. VMware Virtual Ethernet Adapter for VMnet8
   IP地址: 192.168.253.1
3. VMware Virtual Ethernet Adapter for VMnet1
   IP地址: 192.168.197.1
4. Microsoft Wi-Fi Direct Virtual Adapter #2
   IP地址: 169.254.234.222
5. Microsoft Wi-Fi Direct Virtual Adapter
   IP地址: 169.254.176.170
6. Adapter for loopback traffic capture
   IP地址: 127.0.0.1
7. Sangfor SSL VPN CS Support System VNIC
   IP地址: 169.254.46.176
8. Realtek Gaming GbE Family Controller
   IP地址: 169.254.97.253
请选择要打开的网络适配器: 1
成功打开Intel(R) Wi-Fi 6E AX211 160MHz
成功捕获到ARP并解析
10.130.58.73
-> 84:7b:57:4c:36:8c
请输入请求的IP地址: 10.130.58.74
ARP请求发送成功
成功捕获到ARP并解析
10.130.58.74
-> 00:00:5e:00:01:0d
请输入请求的IP地址: |
```

接着ping 该主机

```
C:\Users\HP>ping 10.130.58.74

正在 Ping 10.130.58.74 具有 32 字节的数据:
来自 10.130.58.74 的回复: 字节=32 时间=290ms TTL=63
来自 10.130.58.74 的回复: 字节=32 时间=391ms TTL=63
来自 10.130.58.74 的回复: 字节=32 时间=300ms TTL=63
来自 10.130.58.74 的回复: 字节=32 时间=140ms TTL=63

10.130.58.74 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 140ms, 最长 = 391ms, 平均 = 280ms
```

最后查ARP缓存表

```
C:\Users\HP>arp -a

接口: 10.130.58.73 --- 0x6
Internet 地址          物理地址          类型
10.130.0.1             00-00-5e-00-01-0d 动态
10.130.58.74           00-00-5e-00-01-0d 动态
10.130.99.14           00-00-5e-00-01-0d 动态
10.130.127.255         ff-ff-ff-ff-ff-ff 静态
112.112.112.112        66-66-66-66-66-66 动态
224.0.0.22             01-00-5e-00-00-16 静态
224.0.0.251            01-00-5e-00-00-fb 静态
224.0.0.252            01-00-5e-00-00-fc 静态
239.255.255.250       01-00-5e-7f-ff-fa 静态
255.255.255.255       ff-ff-ff-ff-ff-ff 静态
```

可以发现输出的MAC地址是正确的。

### 三、实验总结

#### 实验难点

本次实验的主要难点在于如何获得自己网卡的MAC地址。通常情况下，MAC地址是硬件层的唯一标识，我们可以通过操作系统的网络工具（如 `ipconfig`）直接查看网卡的MAC地址。然而，在实验中，由于需要模拟ARP协议并使用自己的网卡作为源设备，我们面临着无法直接获取自己网卡MAC地址的问题。为了解决这一问题，我通过老师在课程中的讲解，采用了ARP协议的欺骗技术来获得源MAC地址。

具体来说，我们通过构造一个虚拟的ARP请求包，利用已知的网卡IP地址广播ARP请求，并通过欺骗的方式获取目标设备的MAC地址。由于我们能够轻松获得网卡的IP地址，因此ARP协议能够成功地在局域网内请求到自己的MAC地址，从而完成MAC地址的解析。这种技术使得我们可以在没有直接访问系统硬件的情况下获取网卡的MAC地址，解决了实验中的关键难题。

#### 心得体会

本次实验主要通过实现ARP协议的发送与接收，深入理解了ARP协议在局域网中如何通过广播请求来解析IP地址到MAC地址的映射。实验中，我学习了如何使用 `pcap` 库捕获和发送数据包，掌握了ARP请求和ARP响应的报文结构和工作流程。通过实践，我成功实现了目标设备的MAC地址解析，并通过代码调试验证了实验的正确性。实验过程中，我还了解了ARP协议的优化技术，如高速缓存机制，这为网络性能提升提供了重要的思路。



通过本次实验，我对ARP协议有了更加深入的理解，为后续更复杂的网络编程打下了坚实的基础。