# 第四次实验：获取 IP 地址与 MAC 地址的对应关系

1711342 李纪

2019 年 11 月 16 日

**摘要**

这是我的**实验四**的实验报告，请老师查阅，谢谢。

**关键字：**IP、ARP、MAC

# 目录

**1 实验的目的** **3**

**2 实验基本要求** **3**

**3 实验方案和方法的选择** **3**

3.1 利用命令获取 IP 地址与 MAC 地址的对应关系 . . . . . 3

3.2 通过编程获取 IP 地址与 MAC 地址的对应关系 . . . . . 5

3.2.1 获取本机网络接口的 MAC 地址和 IP 地址 . . . . . 5

3.2.2 向网络发送数据包 . . . . . 5

**4 实验过程** **5**

4.1 开发及测试环境 . . . . . 5

**5 关键部分源代码** **6**

5.1 功能：检测出所有的网络接口卡 . . . . . 8

5.2 功能：显示设备详细信息 . . . . . 9

5.3 功能：发送 ARP 请求 . . . . . 11

5.4 功能：捕获数据报 . . . . . 14

5.5 功能：处理数据报 . . . . . 16

**6 程序演示** **18**

6.1 程序界面介绍 . . . . . 18

6.2 程序逻辑介绍 . . . . . 19

**附录 A ARP：地址解析协议** **21**

A.1 基本功能 . . . . . 21

A.2 数据包结构 . . . . . 21

# 1 实验的目的

本实验的目的是获取以太网中主机的 MAC 地址，因此以太网在该实验中必不可少，本实验使用的以太网既可以是共享式以太网，也可以是交换式以太网。

# 2 实验基本要求

实验基本要求参考学院官网（学生端后台）上发布的实验要求以及《计算机网络技术与应用》[1]。

# 3 实验方案和方法的选择

实验方案和方法参考《网络技术与应用》[1]。

## 3.1 利用命令获取 IP 地址与 MAC 地址的对应关系

网络操作系统通常会将从网络中得到的 IP 地址与 MAC 地址的映射关系存放在本地的高速缓冲区中，因此，查看该缓冲区中的表项就可以获得一个 IP 地址与 MAC 地址的对应关系。多数网络操作系统（包括 Windows、UNIX、Linux 等系列操作系统）都内置了一个 arp 命令，用于查看、添加和删除高速缓存区中的 ARP 表项。

在 Windows 操作系统中，高速缓存区中的 ARP 表可以包含动态和静态表项。动态表项随时间推移自动添加和删除。静态表项则一直保留在高速缓存区中，直到人为删除或重新启动计算机为止。

在 ARP 表中，每个动态表项的潜在生命周期都是十分钟。新表项加入时定时器开始计时，如果某个表项添加后两分钟内没有被再次使用，则此表项过期并从 ARP 表中删除。如果某个表项被再次使用，则该表项又收到两分钟的生命周期，如果某个表项始终在使用，则它的最长生命周期为十分钟。

#### 1. 显示高速缓存区中的 ARP 表

显示高速缓存区中的 ARP 表可以使用 arp -a 命令，因为 ARP 表在没有进行手工配置之前，通常为动态 ARP 表项，所以，表项的变动较大，arp -a 命令输出的结果也大不相同。如果高速缓存区中的 ARP 表项为空，则 arpa 命令输出的结果为 No ARP Entries Found：如果 ARP 表中存在 IP 地址与 MAC 地址的映射关系，则 arp -a 命令显示该映射关系。如图 1 所示。

如果希望看到的 IP 地址与 MAC 地址的映射关系没有包含在 ARP 表中，可以利用 ping 命令去 ping 该 IP 地址，一旦 ping 成功该 IP 地址与 MAC 地址的映射关系就会加入

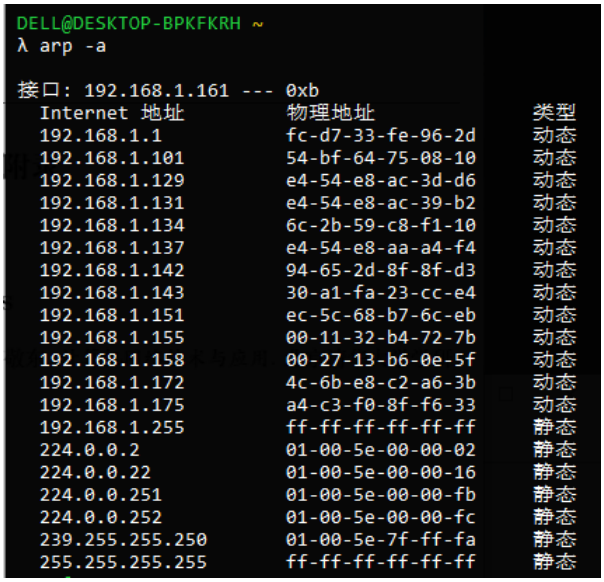


图 1: 利用 arp -a 命令显示高速缓存区中的 ARP 表

ARP 表中。

**2. 添加 ARP 静态表项**

存储在高速缓存区中的 ARP 表既可以有动态表项，也可以有静态表项。通过 arp -s inet_addr eth_addr 命令，可以将 IP 地址与 MAC 地址的映射关系手工加入 ARP 表中。其中，inet_adr 为 IP 地址，eth_addr 为与其对应的 MAC 地址，通过 arp -s 命令加入的表项是静态表项，所以，系统不会自动将它从 ARP 表中删除，直到人为删除或关机。需要注意的是 · 在人为增加 ARP 表项时，一定要确保 IP 地址与 MAC 地址的对应关系是正确的，否则将导致发送失败。

**3. 删除 ARP 表项**

动态表项和静态表项都可以通过 arp -d inet_addr 命令删除，其中 inet_addr 为该表项的 IP 地址。如果要删除 ARP 表中的所有表项，也可以使用 “∗” 代替具体的 IP 地址

## 3.2 通过编程获取 IP 地址与 MAC 地址的对应关系

### 3.2.1 获取本机网络接口的 MAC 地址和 IP 地址

调川 WinPcap 的 pcap_findalldevs_ex() 函数后，参数 alldevs 指向的链表中包含了主机中安装的网络接口设备列表。在 alldevs 链表每个元素保存的网络接口相关信息中，地址信息保存了该网络接口卡上绑定的 IP 地址、网络掩码、广播地址和目的地址等。由于每个网卡接口卡上都可以绑定多个 IP 地址因此每个网络接口卡拥有的地址信息也采用了链表结构。然后即可通过此函数获取本机网卡除 MAC 地址外的大部分信息。

为了形成 ARP 请求数据包，不但需要知道本机网络接口上绑定的 IP 地址，而且必须知道这块网卡的 MAC 地址。获取本机网络接口的 MAC 地址和 IP 地址可以使用不同的方法，常用的方法包括利用 NetBIOS 编程接口及 winsock 提供的 gethostbyname() 函数等。但是，如果希望这样获取的 MAC 地址和 IP 地址与 WinPcap 获取的设备接口名字联系起来，那么，编程过程中需要做进一步的处理。实际上，在理解 ARP 的基本思想后，可以直接通过 WinPcap 获取本机网络接口的 MAC 地址。

按照 ARP，以太网中的主机如果发现一个 ARP 请求的 IP 地址为自己拥有的 IP 地址，那么，它将形成 ARP 响应，并将该 IP 地址与 MAC 地址的对应关系返回给请求主机如果应用程序能够捕获到本机发出的 ARP 响应，那么就能够知道本机网络接口的 MAC 地址。

### 3.2.2 向网络发送数据包

为了获取以太网中其他主机的 IP 地址与 MAC 地址的对应关系，应用程序需要向以太网广播 ARP 请求。向以太网发送数据包可以使用 winPcap 提供的 pcap_sendpacket() 函数。发送成功时，pcap_sendpacket() 函数返回 0，否则返回-1。

# 4 实验过程

先理解实验中涉及到的原理（ARP 协议），然后根据正确配置环境，在上一次实验 [2] 的基础之上继续写入源代码。

ARP 的相关知识点在附录 A 中解释。

## 4.1 开发及测试环境

目前仅能保证程序在开发及测试环境中正常工作，**其他情况均不能保证程序能够稳定运行**。

测试环境如下：

- 操作系统：Microsoft Windows 10 专业版 (x64) (version: 18362)

- IDE：Visual Studio 2019 (version: 16.3.8)
- Windows SDK version: 10.0
- 平台工具集：Visual Studio 2019 (v142)
- C++ 语言标准：std:C++17

# 5 关键部分源代码

**("..." 部分表示省略)**

由于我是借用了实验 2[2] 的代码做的，所以很多代码与实验 2 相比较可能会有重复。

此部分分为 4 个小节，第一个小节展示函数和变量的声明，后三个小节每个小节解释一个具体功能的实现。每小节开头都有**代码摘要**，粗略地展示了功能的实现步骤。

代码解释大部分可以根据代码里的注释来解释说明，请老师查阅。为了方便解释与理解，代码之间的联系与具体实现请直接通过 Visual Studio 查看源代码。

```
    ...
class CCapturePacketDlg : public CDialogEx
{
    ...
public:
  map<CString, CString> devMAC; // 本机网络接口卡 IP 和 MAC 的映射关系
  pcap_if_t* alldevs = nullptr; // 指向设备链表首部的指针
  pcap_if_t* curdev = nullptr; // 一个全局指针，指向当前选中的设备
  pcap_t* adhandle = nullptr; // 一个句柄
  CWinThread* m_Capturer = nullptr; // 启动工作者线程的指针
  char errbuf[PCAP_ERRBUF_SIZE]; // 错误信息缓冲区
  bool m_capStatus = false; // 判断是否处于捕获 IP 数据报状态
  bool m_arpCapStatus = false; // 判断是否处于捕获 ARP 数据报状态
  // bool m_gotARPreply = false; // 判断是否已捕获 ARP 回复
  // 在编辑控件中显示内容
  void displayString(CEdit& editCtrl, CString& str);
  afx_msg void OnBnClickedButtoncapture();

  // 捕获报文按钮
  CButton m_capture;
  // 返回按钮
  CButton m_return;
  // 停止捕获按钮
  CButton m_stopCapture;
  // 过滤器控件
  CEdit m_filter;
  // 以太网接口编辑控件
//  CEdit m_interface;
```

```
  // 以太网接口信息编辑控件
//  CEdit m_interfaceInfo;
  // 日志控件
  CEdit m_log;
  // 网络接口列表
  CListBox m_list_interface;
  // 网络接口信息列表
  CListBox m_list_interfaceInfo;
  afx_msg void OnSelchangeListinterface();
  void UpdateInfo(); // 更新捕获接口的详细信息框
  void SendARP(BYTE* SrcMAC, BYTE* SendHa, DWORD SendIP, DWORD RecvIP); // 发送 ARP 请求
  void GetSelfMACaddr(char* IPaddr); // 获取网络接口卡自身的 MAC 地址
  afx_msg void OnBnClickedButtonstopcapture();
  afx_msg void OnClose();
  afx_msg void OnBnClickedButtonreturn();
  afx_msg void OnBnClickedButtongetmap();
  // 输入的 IP 地址
//  CIPAddressCtrl m_IPaddress;
  // 获取映射关系按钮
  CButton m_getMap;
  // IP 地址与 MAC 地址的映射关系 —— 编辑框控件
  CEdit m_IP_MAC_Map;
  // 输入的 IP 地址
  DWORD m_IPaddress;
  CString strIPaddress; // 字符串类型的输入的 IP 地址
  afx_msg void OnFieldchangedIpaddress(NMHDR* pNMHDR, LRESULT* pResult);
};

// 全局函数
UINT Capturer(PVOID hWnd); // 数据包捕获工作者线程的控制函数

#pragma pack(1)   //进入字节对齐方式
typedef struct FrameHeader_t {  //帧首部
  BYTE  DesMAC[6];  // 目的地址
  BYTE  SrcMAC[6];  // 源地址
  WORD  FrameType;  // 帧类型
} FrameHeader_t;

typedef struct ARPFrame_t {   // ARP 帧
  FrameHeader_t FrameHeader;  //帧头部结构体
  WORD      HardwareType; //硬件类型
  WORD      ProtocolType; //协议类型
  BYTE      HLen;     //硬件地址长度
  BYTE      PLen;     //协议地址长度
  WORD      Operation;    //操作字段 1 -> ARP 请求, 2 -> ARP 回复
  BYTE      SendHa[6];    //源 MAC 地址
  DWORD     SendIP;     //源 IP 地址
  BYTE      RecvHa[6];    //目的 MAC 地址
  DWORD     RecvIP;     //目的 IP 地址
} ARPFrame_t;
```

```
79
80 typedef struct IPHeader_t {   // IP 首部
81   BYTE  Ver_HLen;
82   BYTE  TOS;
83   WORD  TotalLen;
84   WORD  ID;
85   WORD  Flag_Segment;
86   BYTE  TTL;
87   BYTE  Protocol;
88   WORD  Checksum;
89   ULONG SrcIP;
90   ULONG DstIP;
91   WORD  Opt[20];
92 } IPHeader_t;
93
94 typedef struct Data_t { // 包含帧首部和 IP 首部的数据包
95   FrameHeader_t FrameHeader;
96   IPHeader_t    IPHeader;
97 } Data_t;
98
99 #pragma pack()  // 恢复缺省对齐方式
```

Listing 1: CapturePacketDlg.h

## 5.1 功能：检测出所有的网络接口卡

**代码摘要**：在对话框初始化函数 CCapturePacketDlg::OnInitDialog() 中查找设备。

1. CCapturePacketDlg::OnInitDialog()
   - 获取网络接口卡设备列表
   - 在列表框控件中打印设备列表
   - 完成界面初始化（默认选择第一个设备，显示第一个设备的详情；禁用“停止捕获”控件）

```
1 BOOL CCapturePacketDlg::OnInitDialog()
2 {
3   ...
4   int i = 0; // 标识找到 i 个设备
5
6   /* Retrieve the device list */
7   if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf) == -1)
8   {
9     fprintf(stderr, "Error in pcap_findalldevs: %s\n", errbuf);
10    AfxMessageBox((CString)errbuf);
```

```
    exit(1);
  }

  /* Print the list */
  for (pcap_if_t* d = alldevs; d; d = d->next)
    m_list_interface.InsertString(-1, (CString)("%d. %s\r\n", ++i, d->name));
  UpdateData(true);
  Invalidate(true);
  UpdateWindow(); // 更新窗口

  if (i == 0) // 如果没有检测到设备
  {
    AfxMessageBox(_T("No interfaces found! Make sure Npcap is installed."));
    return -1;
  }


  m_list_interface.SetCurSel(0); // 默认选中第一行
  int cur = m_list_interface.GetCurSel(); // 获取listbox被选中的行的数目

  // 找到当前指向的设备
  curdev = alldevs;
  while (cur--)
    curdev = curdev->next;
  UpdateInfo();

  // 禁用“停止捕获”控件
  m_stopCapture.EnableWindow(false);

  return TRUE;  // 除非将焦点设置到控件，否则返回 TRUE
}
```

Listing 2: OnInitDialog()

## 5.2 功能：显示设备详细信息

**代码摘要**：显示设备详细信息结合了两个自定义函数（OnSelchangeListinterface() 和 UpdateInfo()）进行实现。

1. CCapturePacketDlg::OnSelchangeListinterface()
   - 找到当前指向的设备
   - 调用 UpdateInfo() 刷新窗口
2. CCapturePacketDlg::UpdateInfo()
   - 更新捕获接口的详细信息（IP 地址、广播信息、网络掩码、目的地址）

```
void CCapturePacketDlg::OnSelchangeListinterface()
{
  int cur = m_list_interface.GetCurSel();// 获取listbox被选中的行的数目

  // 找到当前指向的设备
  curdev = alldevs;
  while (cur--)
    curdev = curdev->next;
  UpdateInfo();
}

void CCapturePacketDlg::UpdateInfo()
{
  // 更新捕获接口的详细信息
  m_list_interfaceInfo.ResetContent();// 清除原有框的内容
  m_list_interfaceInfo.InsertString(-1, (CString(curdev->name)));     // 显示该网络接口设备的名字
  m_list_interfaceInfo.InsertString(-1, (CString(curdev->description)));  // 显示该网络接口设备的描述信息

  sockaddr_in* temp;
  IN_ADDR temp1;
  char* temp_data;
  CString output;
  for (pcap_addr* a = curdev->addresses; a != NULL; a = a->next)
  {
    if (a->addr->sa_family == AF_INET) // 判断地址是否为 IP 地址
    {
      temp = (sockaddr_in*)(a->addr);
      temp1 = temp->sin_addr;
      temp_data = inet_ntoa(temp1);
      output = _T("IP address: ") + (CString(temp_data)); // 获取 IP 地址
      m_list_interfaceInfo.InsertString(-1, output);

      temp = (sockaddr_in*)(a->netmask);
      temp1 = temp->sin_addr;
      temp_data = inet_ntoa(temp1);
      output = _T("Netmask: ") + (CString(temp_data)); // 获取网络掩码
      m_list_interfaceInfo.InsertString(-1, output);

      temp = (sockaddr_in*)(a->broadaddr);
      temp1 = temp->sin_addr;
      temp_data = inet_ntoa(temp1);
      output = _T("Broadcast address: ") + (CString(temp_data)); // 获取广播地址
      m_list_interfaceInfo.InsertString(-1, output);

      temp = (sockaddr_in*)(a->dstaddr);
      if (temp != nullptr)
      {
```

```
48          temp1 = temp->sin_addr;
49          temp_data = inet_ntoa(temp1);
50          output = _T("Destination address: ") + (CString(temp_data)); // 获取目的地址
51          m_list_interfaceInfo.InsertString(-1, output);
52        }
53      }
54    }
55
56    return void();
57  }
```

Listing 3: OnSelchangeListinterface()、UpdateInfo()

## 5.3 功能：发送 ARP 请求

**代码摘要**:捕获数据报结合了两个自定义函数(OnBnClickedButtongetmap() 和 SendARP())进行实现。

1. OnBnClickedButtongetmap()
   - 读取当前输入的 IP，并发送 ARP 请求
2. SendARP()
   - 调用 AfxBeginThread() 函数启动工作者线程
   - 根据 OnBnClickedButtongetmap() 函数传入的参数，发送 ARP 请求

```
1  void CCapturePacketDlg::OnBnClickedButtongetmap()
2  {
3    m_arpCapStatus = true;
4
5
6    // 调用 AfxBeginThread() 函数启动工作者线程
7    // AfxBeginThread() 函数将返回一个指向新创建线程对象的指针
8    m_Capturer = AfxBeginThread(
9      (AFX_THREADPROC)Capturer, // pfnThreadProc: 指向工作者线程的控制函数，它的值不能为NULL
10     NULL, //
11     THREAD_PRIORITY_NORMAL // 用于指定线程的优先级
12   );
13
14   if (m_Capturer == NULL) {
15     AfxMessageBox(L"启动捕获数据包线程失败!", MB_OK | MB_ICONERROR);
16   }
17
18
19   BYTE* SrcMAC = new BYTE[6];
```

```
20    BYTE* SendHa = new BYTE[6];
21    DWORD SendIP;
22    DWORD RecvIP;
23
24    SrcMAC[0] = 0x50;
25    SrcMAC[1] = 0x2b;
26    SrcMAC[2] = 0x73;
27    SrcMAC[3] = 0xd5;
28    SrcMAC[4] = 0x50;
29    SrcMAC[5] = 0xea;
30
31    SendHa[0] = 0x50;
32    SendHa[1] = 0x2b;
33    SendHa[2] = 0x73;
34    SendHa[3] = 0xd5;
35    SendHa[4] = 0x50;
36    SendHa[5] = 0xea;
37    // 设置要发送的 ARP 数据帧
38    //for (int i = 0; i < 6; ++i)
39    //{
40    //  // SrcMAC[i] = htons(66);
41    //  SendHa[i] = htons(66);
42    //}
43
44    const char* ip = "192.168.1.161";
45    SendIP = inet_addr(ip);// 先自行指定，之后再改
46    RecvIP = htonl(m_IPaddress);
47
48    // 转换成 CString 型 IP 地址，存入变量
49    WORD hiWord = HIWORD(m_IPaddress);
50    WORD loWord = LOWORD(m_IPaddress);
51    BYTE nf1 = HIBYTE(hiWord);
52    BYTE nf2 = LOBYTE(hiWord);
53    BYTE nf3 = HIBYTE(loWord);
54    BYTE nf4 = LOBYTE(loWord);
55    strIPaddress.Format(L"%d.%d.%d.%d", nf1, nf2, nf3, nf4);
56
57    SendARP(SrcMAC, SendHa, SendIP, RecvIP);
58  }
59
60  void CCapturePacketDlg::SendARP(BYTE* SrcMAC, BYTE* SendHa, DWORD SendIP, DWORD RecvIP)
61  {
62    ARPFrame_t ARPFrame;
63
64    // 将 ARPFrame.FrameHeader.DesMAC 设置为广播地址
65    // 将 ARPFrame.FrameHeader.SrcMAC 设置为本机网卡的 MAC 地址
66    for (int i = 0; i < 6; ++i)
67    {
68      ARPFrame.FrameHeader.DesMAC[i] = 0xff;
69      ARPFrame.FrameHeader.SrcMAC[i] = (SrcMAC[i]);
```

```
70  }
71
72  ARPFrame.FrameHeader.FrameType = htons(0x0806); // 帧类型为ARP
73
74  ARPFrame.HardwareType = htons(0x0001); // 硬件类型为以太网
75  ARPFrame.ProtocolType = htons(0x0800); // 协议类型为 IP
76  ARPFrame.HLen = 6; // 硬件地址长度为 6
77  ARPFrame.PLen = 4; // 协议地址长度为 4
78  ARPFrame.Operation = htons(0x0001); // 操作为 ARP 请求
79
80  // 将 ARPFrame.SendHa 设置为本机网卡的 MAC 地址
81  // 将 ARPFrame.SendIP 设置为本机网卡上绑定的 IP 地址
82  // 将 ARPFrame.RecvHa 设置为 0
83  // 将 ARPFrame.RecvIP 设置为请求的 IP 地址
84  for (int i = 0; i < 6; ++i)
85  {
86    ARPFrame.SendHa[i] = (SrcMAC[i]);
87    ARPFrame.RecvHa[i] = 0x00;
88  }
89  ARPFrame.SendIP = SendIP;
90  ARPFrame.RecvIP = RecvIP;
91
92  int cur = m_list_interface.GetCurSel();// 获取listbox被选中的行的数目
93  curdev = alldevs;
94  while (cur--)
95    curdev = curdev->next;
96
97  // 在对某一网络接口卡进行监听之前，首先需要将其打开。打开某一网络接口设备可以使用 WinPcap
       提供的 pcap_open() 函数
98  adhandle = pcap_open(
99    curdev->name // 需要打开的网卡的名字
100   , 65536 // WinPcap 获取网络数据包的最大长度。设为 2^16
101   , PCAP_OPENFLAG_PROMISCUOUS // 它通知系统以混杂模式打开网络接口设备。
102   , 1000 // 数据包捕获函数等待一个数据包的最大时间，设为 1 秒
103   , NULL // 在远程设备中捕获网络数据包时使用。在编写捕获本机网络数据包的应用程序中，需要将
        auth 设置为 NULL
104   , errbuf // 用户定义的存放错误信息的缓冲区。
105 );
106
107
108 int ret = pcap_sendpacket(
109   adhandle, // 指定 pcap_sendpacket()（函数通过哪块接口网卡发送数据包）
110   (u_char*)&ARPFrame, // 指向需要发送的数据包，该数据包应该包括各层的头部信息。
111   sizeof(ARPFrame_t))// 指定发送数据包的大小
112   ;
113
114 if (ret != 0)
115 {
116   CString errorStr;
117   errorStr.Format(_T("pcap_sendpacket() 函数返回 -1.错误。"));
```

```
118     AfxMessageBox(errorStr);
119     // 发送错误处理
120   }
121   else
122   {
123     // 发送成功
124     // Sleep(5000); //等待获取成功
125   }
126
127   return;
128 }
```

Listing 4: OnBnClickedButtongetmap()、SendARP()

## 5.4 功能：捕获数据报

**代码摘要**：捕获数据报结合了两个自定义函数（OnBnClickedButtoncapture() 和 Capturer(PVOID hWnd)）进行实现。

1. Capturer(PVOID hWnd)
   - 找到当前指向的设备
   - 打开网络接口卡
   - 在打开的网络接口卡上捕获网络数据包
2. CCapturePacketDlg::OnBnClickedButtoncapture()
   - 调用 AfxBeginThread() 函数启动工作者线程

```
1  UINT Capturer(PVOID hWnd) // 数据包捕获工作者线程的控制函数
2  {
3    CCapturePacketDlg* dlg = (CCapturePacketDlg*)theApp.m_pMainWnd; //获取对话框句柄
4
5    int cur = dlg->m_list_interface.GetCurSel();// 获取listbox被选中的行的数目
6    dlg->curdev = dlg->alldevs;
7    while (cur--)
8      dlg->curdev = dlg->curdev->next;
9
10   char* errbuf = new char[100];
11
12   // 在对某一网络接口卡进行监听之前，首先需要将其打开。打开某一网络接口设备可以使用 WinPcap
       提供的 pcap_open() 函数
13   dlg->adhandle = pcap_open(
14     dlg->curdev->name // 需要打开的网卡的名字
15     , 65536 // WinPcap 获取网络数据包的最大长度。设为 2^16
```

```
    , PCAP_OPENFLAG_PROMISCUOUS // 它通知系统以混杂模式打开网络接口设备。
    , 1000 // 数据包捕获函数等待一个数据包的最大时间，设为 1 秒
    , NULL // 在远程设备中捕获网络数据包时使用。在编写捕获本机网络数据包的应用程序中，需要将 auth 设置为 NULL
    , errbuf // 用户定义的存放错误信息的缓冲区。
  );

  // 调用出错时，pcap_open() 函数返回 NULL，可以通过 errbuf 获取错误的详细信息
  if (dlg->adhandle == NULL)
  {
    AfxMessageBox(_T("打开网卡出错：") + (CString)errbuf);
    return -1;
  }


  while (dlg->m_capStatus == true)
  {
    // 在打开的网络接口卡上捕获网络数据包
    int pkt = pcap_next_ex(
      dlg->adhandle, // pcap_next_ex() 函数通过该参数指定捕获哪块网卡上的网络数据包。
      &pkt_header, // 在 pcap_next_ex() 函数调用成功后，
            // 该参数指向的 pcap_pkthdr 结构保存有所捕获网络数据包的一些基本信息
      &pkt_data // pkt_data：指向捕获到的网络数据包。
    );

    // 如果在调用过程中发生错误，那么 pcap_next_ex() 函数将返回 -1
    if (pkt == -1)
    {
      AfxMessageBox(_T("在调用过程中发生错误，pcap_next_ex() 函数返回 -1"));
      return -2;
    }
    // 指定的时间范围内（read_timeout）没有捕获到任何网络数据包，那么 pcap_next_ex() 函数将返回 0
    else if (pkt == 0)
    {
      // AfxMessageBox(_T("指定的时间范围内（read_timeout）没有捕获到任何网络数据包，pcap_next_ex() 函数返回 0"));
      // return -3;
      continue;
    }
    // 如果 pcap_next_ex() 函数，正确捕获到一个数据包，那么，它将返回 1。
    else if (pkt == 1)
    {

      dlg->SendMessage(WM_PACKET, 0, 0); // SendMessage 为同步式的消息发送，
                    // 它将消息放入窗口的消息队列后等待消息被处理后返回
    }
  }


```

```
63    return 0;
64  }
65
66  void CCapturePacketDlg::OnBnClickedButtoncapture()
67  {
68    m_capStatus = true;
69    m_capture.EnableWindow(false);
70    m_stopCapture.EnableWindow(true);
71
72    // 调用 AfxBeginThread() 函数启动工作者线程
73    // AfxBeginThread() 函数将返回一个指向新创建线程对象的指针
74    m_Capturer = AfxBeginThread(
75      (AFX_THREADPROC)Capturer, // pfnThreadProc: 指向工作者线程的控制函数, 它的值不能为NULL
76      NULL, //
77      THREAD_PRIORITY_NORMAL // 用于指定线程的优先级
78    );
79
80    if (m_Capturer == NULL) {
81      AfxMessageBox(L"启动捕获数据包线程失败!", MB_OK | MB_ICONERROR);
82      return;
83    }
84    else   /*打开选择的网卡 */
85    {
86      CString temp = _T("监听 ") + (CString(curdev->description)) + _T("\r\n");
87      displayString(m_log, temp + bar);
88    }
89    UpdateData(true);
90    Invalidate(true);
91    UpdateWindow();
92  }
```

Listing 5: OnBnClickedButtoncapture()、Capturer(PVOID hWnd)

## 5.5 功能：处理数据报

**代码摘要**：程序只需处理 ARP 回复数据包，且要过滤掉所有不需要显示的 ARP 回复（比如本机发送的 ARP 回复），避免关系对应出错。

```
1   LRESULT CCapturePacketDlg::OnPacket(WPARAM wParam, LPARAM lParam)
2   {
3     ...
4     // ARP 报文部分
5     if (m_arpCapStatus == true)
6     {
7       CString returnMAC; // ARP 帧回复的源 MAC 地址
8
9       ARPFrame_t* ARPPacket;
10      WORD Operation; //操作字段 1 -> ARP 请求, 2 -> ARP 回复
```

```
    BYTE SendHa[6]; //源 MAC 地址
    DWORD SendIP; //源 IP 地址
    BYTE RecvHa[6]; //目的 MAC 地址
    DWORD RecvIP; //目的 IP 地址

    ARPPacket = (ARPFrame_t*)pkt_data;
    Operation = ntohs(ARPPacket->Operation);
    SendIP = (ARPPacket->SendIP);
    RecvIP = (ARPPacket->RecvIP);

    // 输入的 IP
    CString inputIP = ConvertInputIP(m_IPaddress);

    // 转换成 CString 型 IP 地址，存入变量
    CString tempSendIP = ConvertIP(SendIP);

    // 转换成 CString 型 IP 地址，存入变量
    CString tempRecvIP = ConvertIP(RecvIP);

    if (Operation == 2) // 保证是 ARP 回复
    {
      map<CString, CString>::iterator it;
      it = devMAC.find(tempSendIP);

      sockaddr_in* temp;
      IN_ADDR temp1;
      char* temp_data;
      CString curdevIP;
      // CString IP; // 存放 IP 的临时变量

      for (pcap_addr* a = curdev->addresses; a != NULL; a = a->next)
      {
        if (a->addr->sa_family == AF_INET) // 判断地址是否为 IP 地址
        {
          temp = (sockaddr_in*)(a->addr);
          temp1 = temp->sin_addr;
          temp_data = inet_ntoa(temp1); // inet_ntoa 函数转换网络字节排序的地址为标准的ASCII以点分开的地址,该函数返回指向点分开的字符串地址的指针
          curdevIP = CString(temp_data); // 获取 IP 地址
        }
      }

      // 一般情况(SendIP = 输入的 IP; RecvIP = 本机 IP)
      // 其他设备发出的，用于回应本机请求的回复
      if (tempSendIP == inputIP && tempRecvIP == curdevIP)
      {
        // m_gotARPreply = true; // 已捕获 ARP 回复

        for (int i = 0; i < 6; i++)
        {
```

```
60            SendHa[i] = ARPPacket->SendHa[i];
61            RecvHa[i] = ARPPacket->RecvHa[i];
62          }
63
64          returnMAC.Format(L"%02x:%02x:%02x:%02x:%02x:%02x\r\n", int(SendHa[0]), int(SendHa
    [1]), int(SendHa[2]), int(SendHa[3]), int(SendHa[4]), int(SendHa[5]));
65          displayString(m_IP_MAC_Map, strIPaddress + arrow + returnMAC);
66        }
67
68        // SendIP == 本机 IP, RecvIP == 192.168.1.250
69        // 本机发出的, 用于回应假 IP(192.168.1.250)发出请求的回复
70        else if (tempRecvIP == "192.168.1.250")
71        {
72          for (int i = 0; i < 6; i++)
73          {
74            SendHa[i] = ARPPacket->SendHa[i];
75            RecvHa[i] = ARPPacket->RecvHa[i];
76          }
77
78          returnMAC.Format(L"%02x:%02x:%02x:%02x:%02x:%02x\r\n", int(SendHa[0]), int(SendHa
    [1]), int(SendHa[2]), int(SendHa[3]), int(SendHa[4]), int(SendHa[5]));
79          devMAC.insert(pair<CString, CString>(tempSendIP, returnMAC)); // 插入映射表
80        }
81
82        // SendIP == 本机 IP, RecvIP == 其他 IP
83        // 本机发出的, 用于回应其他设备的请求的回复
84        else
85        {
86          // 这种情况下就啥也不干
87        }
88      }
89    }
90
91    return LRESULT();
92  }
```

Listing 6: CCapturePacketDlg::OnPacket()

# 6 程序演示

## 6.1 程序界面介绍

程序分为 4 个部分，分别是网络接口卡显示区、结果显示区、控制区以及 ARP 请求区。

其中，网络接口卡显示区由程序上方的两个列表框控件组成，结果显示区由一个编辑框控件组成，控制区由三个按钮控件组成，ARP 请求区由一个 IP 地址控件、一个“获取映射关系”按钮和一个编辑框控件组成。中间的过滤条件未实现，故无效。

## 6.2 程序逻辑介绍

1. IP 数据报部分：

   - 在左部的列表框控件中点击任意网络接口卡后，右部的列表框控件中会显示该卡相关信息，包括名字、IP 地址、网络掩码、广播地址等。
   - 点击“捕获报文”按钮后，程序会自动捕获通过当时选中的网络接口卡的 IPv4 报文。
   - 在下方的编辑框控件中显示 IPv4 报文相关信息，包括标识、（数据报中的）头部校验和、（程序）计算出的头部校验和等。
   - 点击“停止捕获”按钮后，停止捕获报文。
   - 点击“返回（Clear）”按钮能够清空当前下方的编辑框控件中的内容。

2. ARP 部分：

   - 在 IP 地址控件中输入请求 IP，输入完成后，点击“获取映射关系”按钮
   - 稍等一会儿后，就能在它们下方的第一个编辑框控件中看到结果

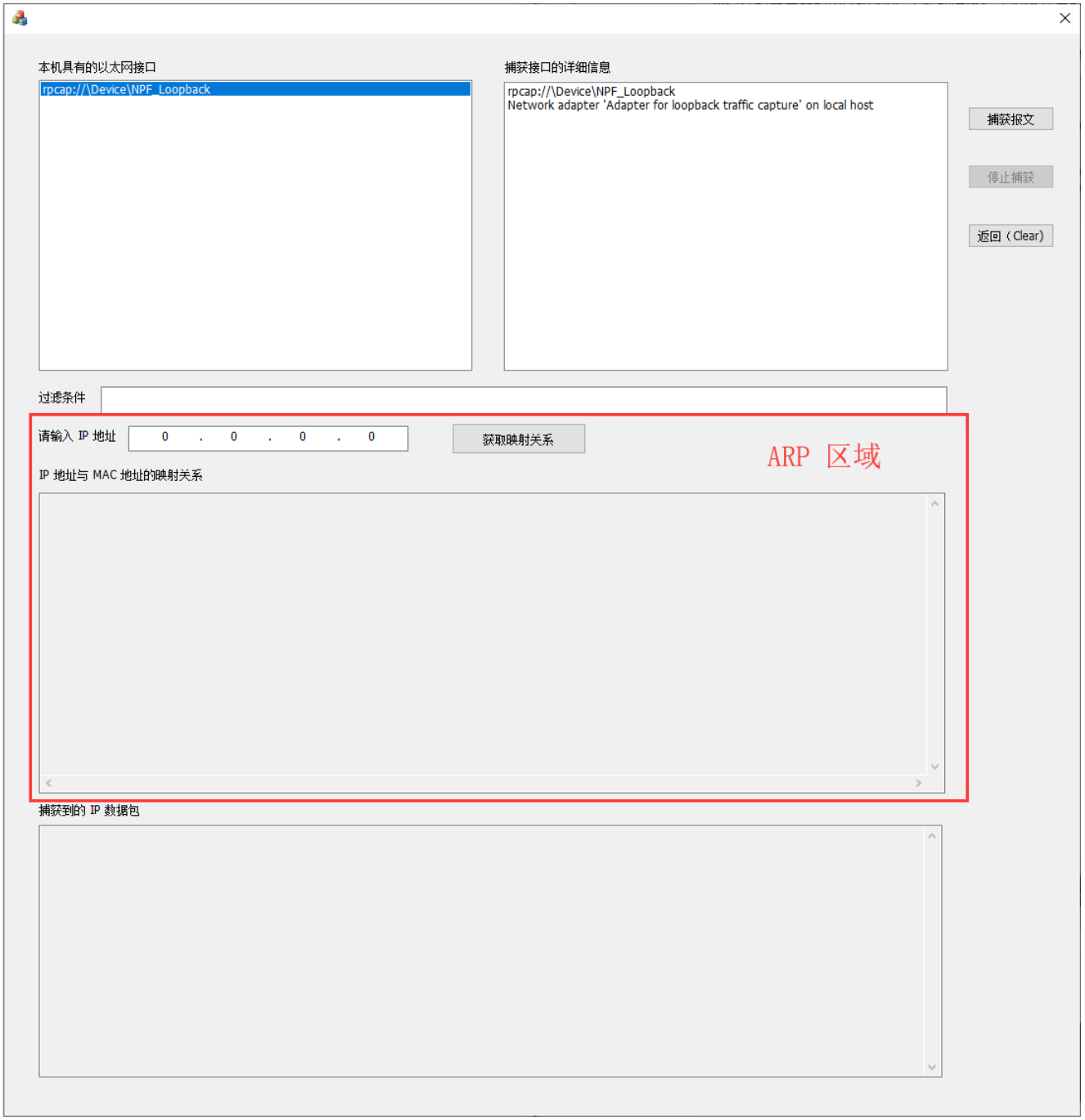
本机具有的以太网接口
rpcap://\Device\NPF_Loopback
捕获接口的详细信息
rpcap://\Device\NPF_Loopback
Network adapter 'Adapter for loopback traffic capture' on local host
捕获报文
停止捕获
返回（Clear）
过滤条件
请输入 IP 地址
0 . 0 . 0 . 0
获取映射关系
ARP 区域
IP 地址与 MAC 地址的映射关系
捕获到的 IP 数据包

图 2: 程序界面介绍

# A ARP：地址解析协议

**地址解析协议**（英语：**A**ddress **R**esolution **P**rotocol，缩写：**ARP**）是一个通过解析网络层地址来找寻数据链路层地址的网络传输协议，它在 IPv4 中极其重要。ARP 最初在 1982 年的 RFC 826 中提出并纳入互联网标准 STD 37. **ARP** 也可能指是在多数操作系统中管理其相关地址的一个进程。

ARP 是通过网络地址来定位 MAC 地址。ARP 已经在很多网路层和数据链接层之间得以实现，包括 IPv4，Chaosnet, DECnet 和 Xerox PARC Universal Packet（PUP）使用 IEEE 802 标准, 光纤分布式数据接口, X.25, 帧中继和异步传输模式（ATM），IEEE 802.3 和 IEEE 802.11 标准上 IPv4 占了多数流量。

在 IPv6 中邻居发现协议（NDP）用于代替地址解析协议（ARP）。

## A.1 基本功能

在以太网协议中规定，同一局域网中的一台主机要和另一台主机进行直接通信，必须要知道目标主机的 MAC 地址。而在 TCP/IP 协议中，网络层和传输层只关心目标主机的 IP 地址。这就导致在以太网中使用 IP 协议时，数据链路层的以太网协议接到上层 IP 协议提供的数据中，只包含目的主机的 IP 地址。于是需要一种方法，根据目的主机的 IP 地址，获得其 MAC 地址。这就是 ARP 协议要做的事情。所谓**地址解析（address resolution）**就是主机在发送帧前将目标 IP 地址转换成目标 MAC 地址的过程。

另外，当发送主机和目的主机不在同一个局域网中时，即便知道对方的 MAC 地址，两者也不能直接通信，必须经过路由转发才可以。所以此时，发送主机通过 ARP 协议获得的将不是目的主机的真实 MAC 地址，而是一台可以通往局域网外的路由器的 MAC 地址。于是此后发送主机发往目的主机的所有帧，都将发往该路由器，通过它向外发送。这种情况称为委托 ARP 或 **ARP 代理（ARP Proxy）**。

在点对点链路中不使用 ARP，实际上在点对点网络中也不使用 MAC 地址，因为在此类网络中分别已经获取了对端的 IP 地址。

## A.2 数据包结构

地址解析协议的消息格式很简单，仅包含单一的地址解析请求或响应。ARP 消息的长度取决于上下两层地址的大小，上层地址由所使用的网络协议类型（通常是 IPv4）决定，下层地址则由上层协议所使用的硬件或虚拟链路层的类型决定。消息的报头中包含了这些类型以及对应的地址长度信息，此外还包含了表示请求（1）和应答（2）的操作码。数据包的有效负载为收发双方的硬件地址、协议地址，总计四个地址。

为了把 IP 地址映射到 48 位以太网地址用于传输，需要一个体现地址转换协议的包格式。具体格式可参考图 3 。

### 以太网链路层 [编辑]

- 目标以太网地址：目标MAC地址。FF:FF:FF:FF:FF:FF （二进制全1）为广播地址。
- 源以太网地址：发送方MAC地址。
- 帧类型：以太类型，ARP为0x0806。

### 以太网报文数据 [编辑]

- 硬件类型：如以太网（0x0001）、分组无线网。
- 协议类型：如网际协议(IP)（0x0800）、IPv6（0x86DD）。
- 硬件地址长度：每种硬件地址的字节长度，一般为6（以太网）。
- 协议地址长度：每种协议地址的字节长度，一般为4（IPv4）。
- 操作码：1为ARP请求，2为ARP应答，3为RARP请求，4为RARP应答。
- 源硬件地址：n个字节，n由硬件地址长度得到，一般为发送方MAC地址。
- 源协议地址：m个字节，m由协议地址长度得到，一般为发送方IP地址。
- 目标硬件地址：n个字节，n由硬件地址长度得到，一般为目标MAC地址。
- 目标协议地址：m个字节，m由协议地址长度得到，一般为目标IP地址。

### 报文格式 [编辑]

| 长度(位) | 48 | 48 | 16 | 16 | 16 | 8 | 8 | 16 | 48 | 32 | 48 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 数据类型 | 目标以太网地址 | 源以太网地址 | 帧类型 | 硬件类型 | 协议类型 | 硬件地址长度 | 协议地址长度 | 操作码 | 源硬件地址 | 源协议地址 | 目标硬件地址 | 目标协议地址 |
| 组成 | 14字节 以太网首部 | | | 28字节 ARP请求/应答 | | | | | | | | |

图 3: ARP 数据包结构

# References

[1] 张建忠、徐敬东. 计算机网络技术与应用. 北京清华大学学研大厦 A 座: 清华大学出版社, 2019.

[2] 李纪. “实验 2：IP 数据报捕获与分析（实验报告）”. In: (2019).