

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

数据包捕获与分析编程实验

二、实验要求

- (1)了解NPcap的架构。
- (2)学习NPcap的设备列表获取方法、网卡设备打开方法，以及数据包捕获方法
- (3)通过NPcap编程，实现本机的数据包捕获，显示捕获数据帧的源MAC地址和目的MAC地址，以及类型!长度字段的值。
- (4)捕获的数据包不要求硬盘存储，但应以简单明了的方式在屏幕上显示，必显字段包括源MAC地址、目的MAC地址和类型!长度字段的值。
- (5)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。

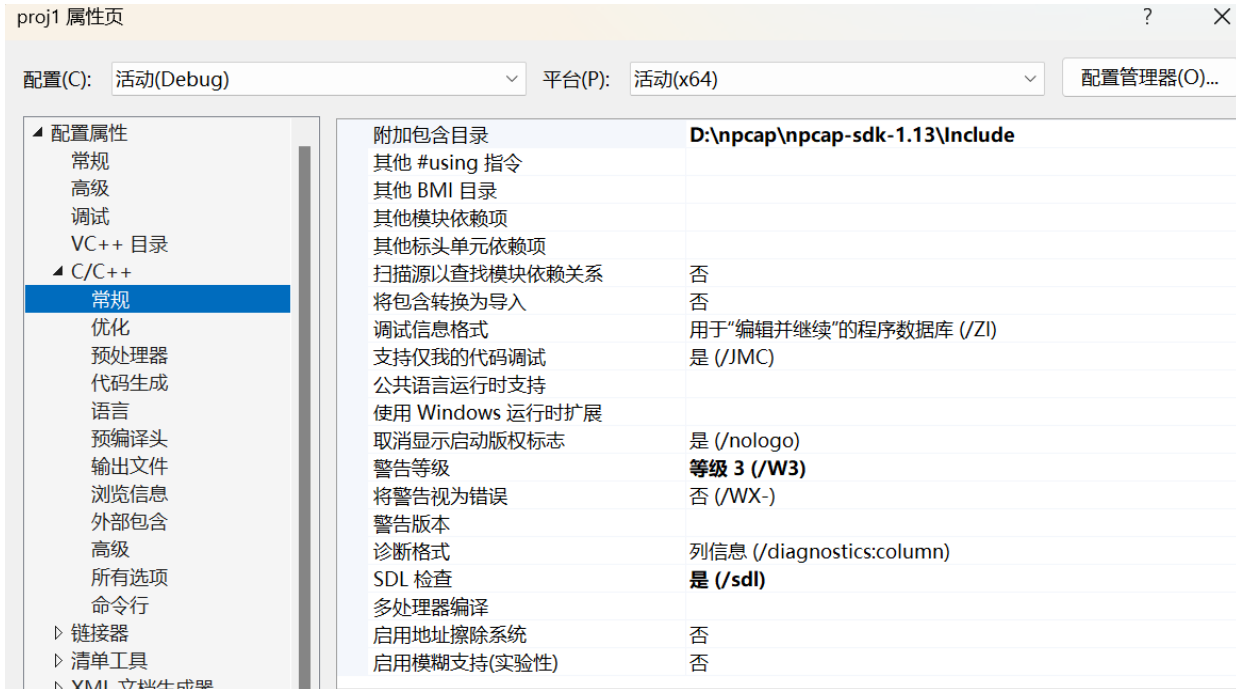
第二步：把Npcap SDK 1.13.zip解压，解压后如下图所示。

docs	2024/10/13 9:54	文件夹
Examples-pcap	2024/10/13 9:54	文件夹
Examples-remote	2024/10/13 9:54	文件夹
Include	2024/10/13 9:54	文件夹
Lib	2024/10/13 9:54	文件夹
Npcap_Guide.html	2024/10/13 9:54	Firefox HTML Docu... 1 KB

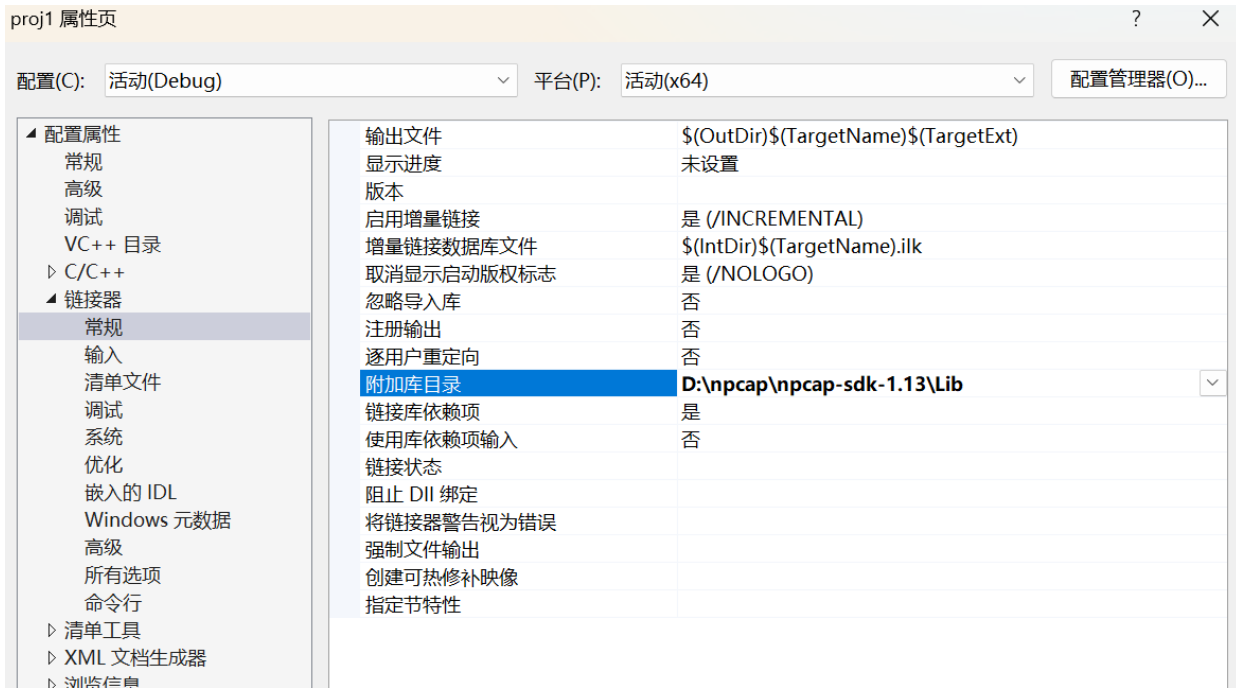
只需要上图所示的两个文件夹，拷贝至VS code工程路径下即可

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

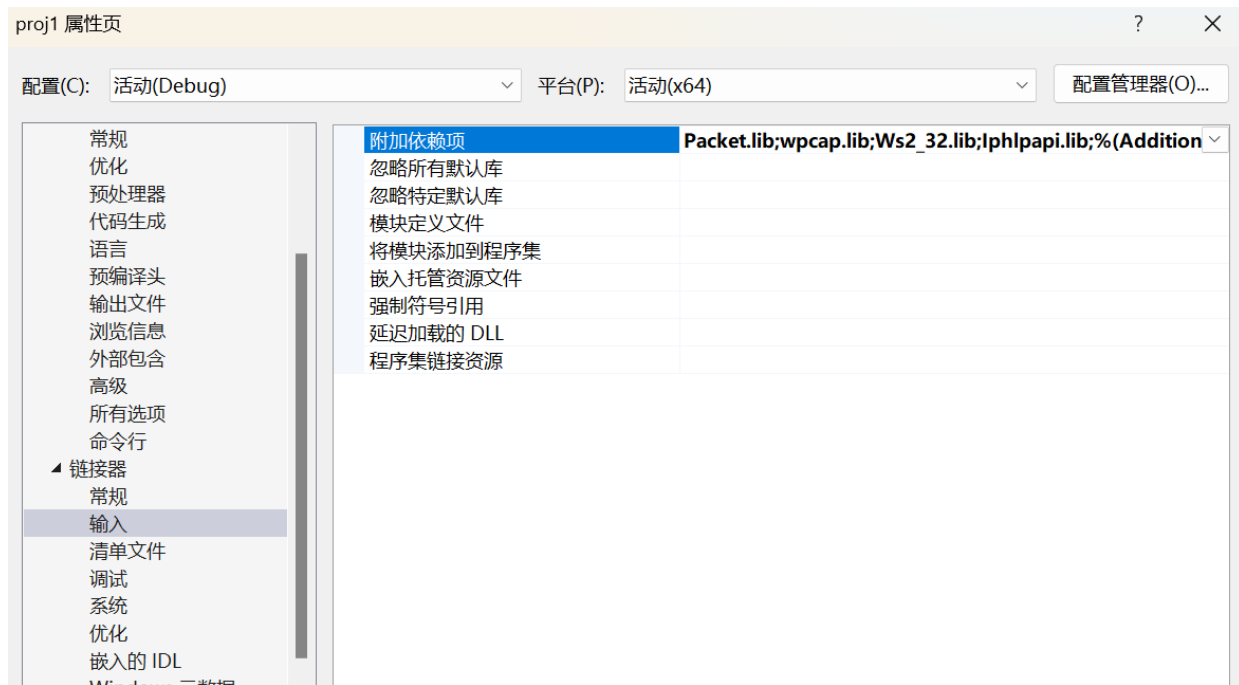


(3) 项目→属性→链接器→输入→附加依赖项：添加Packet.lib;wpcap.lib;Ws2_32.lib;Iphlpapi.lib;

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。

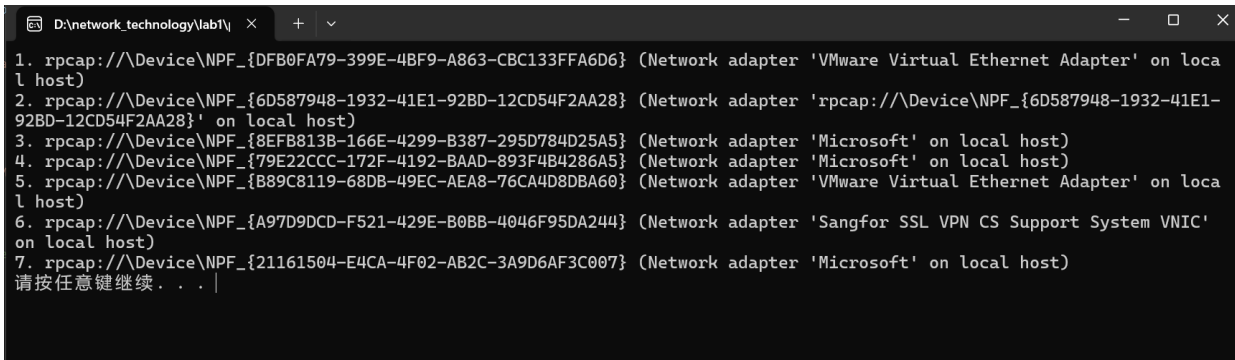


(4) 输入官网的测试代码，若成功检测则环境配置成功

```
#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");  
}
```



```
D:\network_technology\lab1\ > + v  
1. rpcap://\Device\NPF_{DFB0FA79-399E-4BF9-A863-CBC133FFA6D6} (Network adapter 'VMware Virtual Ethernet Adapter' on local host)  
2. rpcap://\Device\NPF_{6D587948-1932-41E1-92BD-12CD54F2AA28} (Network adapter 'rpcap://\Device\NPF_{6D587948-1932-41E1-92BD-12CD54F2AA28}' on local host)  
3. rpcap://\Device\NPF_{8EFB813B-166E-4299-B387-295D784D25A5} (Network adapter 'Microsoft' on local host)  
4. rpcap://\Device\NPF_{79E22CCC-172F-4192-BAAD-893F4B4286A5} (Network adapter 'Microsoft' on local host)  
5. rpcap://\Device\NPF_{B89C8119-68DB-49EC-AEA8-76CA4D8DBA60} (Network adapter 'VMware Virtual Ethernet Adapter' on local host)  
6. rpcap://\Device\NPF_{A97D9DCD-F521-429E-B0BB-4046F95DA244} (Network adapter 'Sangfor SSL VPN CS Support System VNIC' on local host)  
7. rpcap://\Device\NPF_{21161504-E4CA-4F02-AB2C-3A9D6AF3C007} (Network adapter 'Microsoft' on local host)  
请按任意键继续 . . .
```

2. Npcap基本操作

Npcap 是基于 Windows 平台的高性能网络数据包捕获和传输库，通常用于网络监控、分析以及数据包捕获。它是 WinPcap 的替代方案，提供了更好的兼容性和功能。

2.1 架构概述

内核驱动 (Kernel Driver):

- Npcap 的核心部分是内核模式的驱动程序，它负责在系统的网络堆栈中捕获进出网络适配器的数据包。
- 通过网络驱动层与 Windows 网络堆栈进行通信，能直接读取数据链路层的帧，也就是可以捕获到原始数据包。
- 内核驱动能够支持“环回捕获” (loopback capture)，即它能够捕获本地应用程序之间发送的网络流量。

用户空间库 (User Space Library):

- 在用户空间，Npcap 提供了 libpcap 风格的 API，使得开发者能够使用熟悉的接口来处理数据包捕获任务。使用这些接口，开发者可以访问网络适配器、读取捕获的数据包、并将其解析为更高层的网络协议。
- 用户通过调用这些接口与内核驱动进行通信。通常包括打开适配器、设置过滤器、开始捕获等操作。

支持的协议和功能:

- Npcap 允许用户捕获以太网帧 (Ethernet frame)，并能处理各种网络协议 (如 ARP、IP、TCP、UDP 等)。
- 支持监控模式 (Monitor Mode)，可以用于无线网卡捕获 Wi-Fi 数据包。
- 支持环回 (Loopback) 接口，使得能够捕获到本地应用程序之间的数据包，而不仅仅是通过网卡发送的流量。

安全性和性能改进:

- Npcap 的架构特别关注性能优化，能够在高负载情况下高效捕获大量的数据包。
- 支持 Windows 网络驱动模型 (NDIS 6)，能够与 Windows 10 及更高版本兼容，并且支持 IPv6。

2.2 Npcap工作流程

- **网络设备的检测**：程序使用库函数获取可用的网络设备列表。
- **打开设备**：选择合适的网络设备（网卡）并打开，准备开始捕获数据包。
- **设置过滤器**：可以通过设置过滤器，只捕获特定类型的流量（如 TCP 流量，特定端口号的数据包）。
- **捕获数据包**：程序使用回调机制或轮询来实时捕获传入和传出的数据包。
- **解析数据包**：解析捕获的数据包，提取出需要的字段（如 MAC 地址、协议类型、数据长度等）。
- **显示或处理数据**：将解析后的信息以人类可读的形式展示在终端，或者用于其他网络分析目的。

2.3 获取设备列表

使用 `pcap_findalldevs()` 函数获取本地所有网络适配器的列表：

```
#include <pcap.h>
#include <stdio.h>

int main() {
    pcap_if_t *alldevs;
    pcap_if_t *device;
    char errbuf[PCAP_ERRBUF_SIZE];

    // 获取本地所有的设备列表
    if (pcap_findalldevs(&alldevs, errbuf) == -1) {
        fprintf(stderr, "Error finding devices: %s\n", errbuf);
        return 1;
    }

    // 输出设备列表
    printf("Available devices:\n");
    for (device = alldevs; device != NULL; device = device->next) {
        printf("%s - %s\n", device->name, device->description ? device->description
: "No description available");
    }

    // 释放设备列表
    pcap_freealldevs(alldevs);
    return 0;
}
```

该程序列出了系统中所有可用的网络设备，包括其名称和描述。`pcap_findalldevs` 会返回链表，包含每个设备的信息。显示结果如下：

```
Microsoft Visual Studio 调试 × + v

Available devices:
\Device\NPF_{DFB0FA79-399E-4BF9-A863-CBC133FFA6D6} - VMware Virtual Ethernet Adapter
\Device\NPF_{6D587948-1932-41E1-92BD-12CD54F2AA28} - 
\Device\NPF_{8EFB813B-166E-4299-B387-295D784D25A5} - Microsoft
\Device\NPF_{79E22CCC-172F-4192-BAAD-893F4B4286A5} - Microsoft
\Device\NPF_{B89C8119-68DB-49EC-AEA8-76CA4D8DBA60} - VMware Virtual Ethernet Adapter
\Device\NPF_{A97D9DCD-F521-429E-B0BB-4046F95DA244} - Sangfor SSL VPN CS Support System VNIC
\Device\NPF_{21161504-E4CA-4F02-AB2C-3A9D6AF3C007} - Microsoft

D:\network_technology\lab1\proj1\x64\Debug\proj1.exe (进程 41816)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...|
```

程序成功列出了当前系统中可用的网络设备，包括 VMware 虚拟网卡、Microsoft 虚拟适配器、以及 VPN 相关设备。每个设备都以其对应的 `\Device\NPF_` 前缀和唯一标识符显示。

2.4 打开网卡设备

使用 `pcap_open_live()` 函数可以打开一个指定的网络适配器以捕获数据包。

```
#include <pcap.h>
#include <stdio.h>

int main() {
    pcap_if_t *alldevs;
    pcap_if_t *device;
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];

    // 获取设备列表
    if (pcap_findalldevs(&alldevs, errbuf) == -1) {
        fprintf(stderr, "Error finding devices: %s\n", errbuf);
        return 1;
    }

    // 打开第一个设备进行捕获
    device = alldevs; // 假设选择第一个设备
    handle = pcap_open_live(device->name, 65536, 1, 1000, errbuf);
    if (handle == NULL) {
        fprintf(stderr, "Couldn't open device %s: %s\n", device->name, errbuf);
        pcap_freealldevs(alldevs);
        return 2;
    }

    printf("Device %s opened for capturing\n", device->name);

    // 释放设备列表
    pcap_freealldevs(alldevs);
    return 0;
}
```

该程序通过 `pcap_open_live()` 打开网卡设备。65536 是捕获数据包的最大字节数，1 表示是否设置成混杂模式，1000 表示捕获的超时时间（单位为毫秒）。显示结果如下：

```
Device \Device\NPF_{DFB0FA79-399E-4BF9-A863-CBC133FFA6D6} opened for capturing
D:\network_technology\lab1\proj1\x64\Debug\proj1.exe (进程 30548)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...|
```

程序成功打开了第一个设备（\Device\NPF_{DFB0FA79-...}），并提示设备已准备好捕获数据包，随后程序退出。

2.5 获取数据包

```
#include <pcap.h>
#include <stdio.h>
#include <winsock2.h> // 用于ntohs函数
#pragma comment(lib, "ws2_32.lib") // 链接winsock库

// 定义以太网帧头结构
struct ether_header {
    u_char ether_dhost[6]; // 目的MAC地址
    u_char ether_shost[6]; // 源MAC地址
    u_short ether_type;    // 类型/长度字段
};

void packet_handler(u_char* user, const struct pcap_pkthdr* pkthdr, const u_char*
packet) {
    struct ether_header* eth_header;
    eth_header = (struct ether_header*)packet;

    printf("Source MAC: %02x:%02x:%02x:%02x:%02x:%02x\n",
        eth_header->ether_shost[0],
        eth_header->ether_shost[1],
        eth_header->ether_shost[2],
        eth_header->ether_shost[3],
        eth_header->ether_shost[4],
        eth_header->ether_shost[5]);

    printf("Destination MAC: %02x:%02x:%02x:%02x:%02x:%02x\n",
        eth_header->ether_dhost[0],
        eth_header->ether_dhost[1],
        eth_header->ether_dhost[2],
        eth_header->ether_dhost[3],
        eth_header->ether_dhost[4],
        eth_header->ether_dhost[5]);

    printf("Type/Length: 0x%04x\n", ntohs(eth_header->ether_type));
}

int main() {
    pcap_if_t* alldevs;
    pcap_if_t* device;
    pcap_t* handle;
    char errbuf[PCAP_ERRBUF_SIZE];
```

```

// 获取设备列表
if (pcap_findalldevs(&alldevs, errbuf) == -1) {
    fprintf(stderr, "Error finding devices: %s\n", errbuf);
    return 1;
}

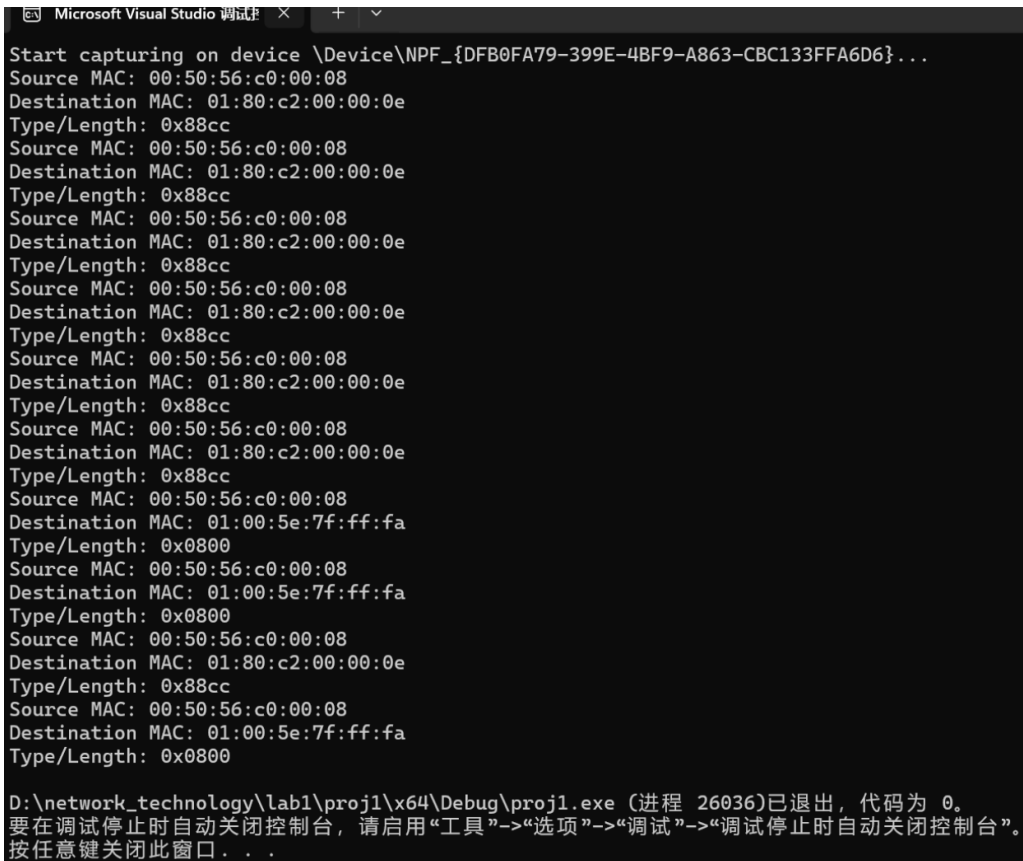
// 打开第一个设备进行捕获
device = alldevs; // 假设选择第一个设备
handle = pcap_open_live(device->name, 65536, 1, 1000, errbuf);
if (handle == NULL) {
    fprintf(stderr, "Couldn't open device %s: %s\n", device->name, errbuf);
    pcap_freealldevs(alldevs);
    return 2;
}

printf("Start capturing on device %s...\n", device->name);

// 开始捕获数据包
pcap_loop(handle, 10, packet_handler, NULL); // 捕获10个数据包

// 释放设备列表
pcap_freealldevs(alldevs);
pcap_close(handle);
return 0;
}

```



```

Start capturing on device \Device\NPF_{DFB0FA79-399E-4BF9-A863-CBC133FFA6D6}...
Source MAC: 00:50:56:c0:00:08
Destination MAC: 01:80:c2:00:00:0e
Type/Length: 0x88cc
Source MAC: 00:50:56:c0:00:08
Destination MAC: 01:80:c2:00:00:0e
Type/Length: 0x88cc
Source MAC: 00:50:56:c0:00:08
Destination MAC: 01:80:c2:00:00:0e
Type/Length: 0x88cc
Source MAC: 00:50:56:c0:00:08
Destination MAC: 01:80:c2:00:00:0e
Type/Length: 0x88cc
Source MAC: 00:50:56:c0:00:08
Destination MAC: 01:80:c2:00:00:0e
Type/Length: 0x88cc
Source MAC: 00:50:56:c0:00:08
Destination MAC: 01:80:c2:00:00:0e
Type/Length: 0x88cc
Source MAC: 00:50:56:c0:00:08
Destination MAC: 01:00:5e:7f:ff:fa
Type/Length: 0x0800
Source MAC: 00:50:56:c0:00:08
Destination MAC: 01:00:5e:7f:ff:fa
Type/Length: 0x0800
Source MAC: 00:50:56:c0:00:08
Destination MAC: 01:80:c2:00:00:0e
Type/Length: 0x88cc
Source MAC: 00:50:56:c0:00:08
Destination MAC: 01:00:5e:7f:ff:fa
Type/Length: 0x0800

D:\network_technology\lab1\proj1\x64\Debug\proj1.exe (进程 26036)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。 . . .

```

1. **Source MAC:** 表示发送该数据包的源设备的 MAC 地址。例如，`00:50:56:c0:00:08` 是捕获到的某个设备的源 MAC 地址。

2. **Destination MAC:** 表示接收该数据包的设备的 MAC 地址。例如，01:80:c2:00:00:0e 是某个数据包的目的 MAC 地址。
3. **Type/Length:** 这是以太网帧中的类型或长度字段。常见的类型包括：
- 0x0800: 表示该数据包是一个 IPv4 数据包。
 - 0x0806: 表示该数据包是 ARP 请求或响应。
 - 0x88cc: 表示该数据包与 Link Layer Discovery Protocol (LLDP) 相关。

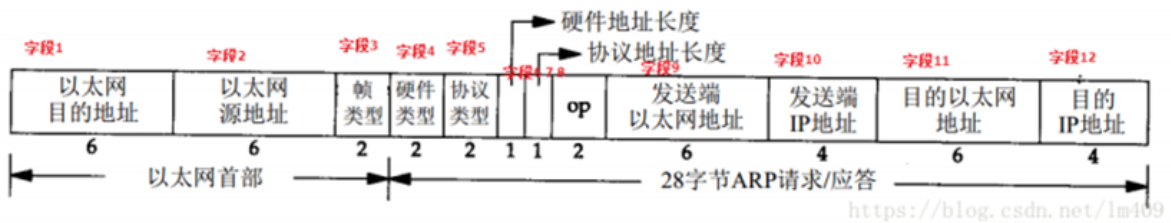
数据包类型说明：

- **LLDP:** 在结果中 Type/Length 值为 0x88cc，表示这是 LLDP 数据包，通常用于网络设备之间进行拓扑发现。
- **IPv4:** 当 Type/Length 值为 0x0800 时，这表示这是一个 IPv4 数据包。

3. ARP数据包

地址解析协议（Address Resolution Protocol），其基本功能为透过目标设备的IP地址，查询目标设备的MAC地址，以保证通信的顺利进行。它是IPv4中网络层必不可少的协议，不过在IPv6中已不再适用，并被邻居发现协议（NDP）所替代。

ARP报文格式：



- (1). 字段1是ARP请求的目的以太网地址，全1时代表广播地址。
- (2). 字段2是发送ARP请求的以太网地址。
- (3). 字段3以太网帧类型表示的是后面的数据类型，ARP请求和ARP应答这个值为0x0806。
- (4). 字段4表示硬件地址的类型，硬件地址不只以太网一种，是以太网类型时此值为1。
- (5). 字段5表示要映射的协议地址的类型，要对IP地址进行映射，此值为0x0800。
- (6). 字段6和7表示硬件地址长度和协议地址长度，MAC地址占6字节，IP地址占4字节。
- (7). 字段8是操作类型字段，值为1，表示进行ARP请求；值为2，表示进行ARP应答；值为3，表示进行 RARP 请求；值为4，表示进行RARP应答。
- (8). 字段9是发送端ARP请求或应答的硬件地址，这里是以太网地址，和字段2相同。
- (9). 字段10是发送ARP请求或应答的IP地址。
- (10). 字段11和12是目的端的硬件地址和协议地址

4. 完整程序代码

4.1 流程

初始化：加载必要的头文件和库，定义辅助函数 `getaddress()` 来获取网络地址。

获取设备列表：调用 `pcap_findalldevs()` 函数，获取当前系统的所有网络接口设备。

打印设备信息：遍历设备链表，输出每个设备的名称、描述、IP 地址、子网掩码和广播地址。

打开设备捕获数据包：尝试逐个设备进行数据包捕获，通过 `pcap_open()` 打开设备。

捕获和解析数据包：使用 `pcap_next_ex()` 进入循环捕获数据包，成功捕获后解析源 MAC、目的 MAC、类型/长度、源 IP、目的 IP 等信息。

资源管理：在捕获完成后，关闭设备并释放设备列表，确保资源正常回收。

4.2 完整代码及注释

```
#include <iostream>
#include <pcap.h>
#pragma comment(lib, "ws2_32.lib") //否则 ntohs() 会导致编译失败。
#pragma warning( disable : 4996 ) //要使用旧函数
using namespace std;

void* getaddress(struct sockaddr* sa) //得到对应的IP地址
{
    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
}

int main() {
    char errbuf[PCAP_ERRBUF_SIZE]; //用于存储 libpcap 函数或操作出现问题时的错误消息或诊断信息。

    pcap_if_t* devices; //指向设备列表第一个
    // 获取网络接口设备列表,返回0表示正常, -1表示出错,输出errbuf里的错误信息并返回
    if (pcap_findalldevs(&devices, errbuf) == -1) {
        cerr << "查找设备失败: " << errbuf << endl;
        return 0;
    }
    //打印设备列表中设备信息
    pcap_if_t* count; //遍历用的指针
    pcap_addr_t* a; //地址指针
    int i = 0; //设备数量计数
    //输出设备名和描述信息
    for (count = devices; count; count = count->next) //借助count指针从第一个设备开始访问
    到最后一个设备
    {
        cout << ++i << ". " << count->name; //输出设备信息和描述
        if (count->description)
            cout << "(" << count->description << ")" << endl;
```

```

else
    cout << "(无描述!)" << endl;
for (a = count->addresses; a != NULL; a = a->next) {
    if (a->addr->sa_family == AF_INET) {
        char str[INET_ADDRSTRLEN];
        inet_ntop(AF_INET, getaddress((struct sockaddr*)a->addr), str,
sizeof(str)); //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 << "IP地址: " << str << endl;
        inet_ntop(AF_INET, getaddress((struct sockaddr*)a->netmask), str,
sizeof(str)); //将 a->netmask 强制转换为 struct sockaddr_in 类型的指针，从a->netmask这个
结构中提取子网掩码。
        cout << "子网掩码: " << str << endl;
        inet_ntop(AF_INET, getaddress((struct sockaddr*)a->broadaddr), str,
sizeof(str)); //将 a->netmask 强制转换为 struct sockaddr_in 类型的指针，从a->broadaddr这个
结构中提取广播地址。
        cout << "广播地址: " << str << endl;
    }
}
//设备数量为0
if (i == 0)
{
    cout << endl << "存在错误! 无查找设备!" << endl;
    return 0;
}
cout << "-----" << endl;

// 开始捕获数据包
int flag = 1; //判断循环的标志位
pcap_if_t* count2; //遍历用的指针2
for (count2 = devices; count2 != NULL; count2 = count2->next) {
    cout << "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，继续下一个接口
        continue;
    }
    flag = 1;
    while (flag) { //在打开的网络接口卡上捕获其所有的网络数据包
        struct pcap_pkthdr* pkt_header; //保存了捕获数据包的基本信息，比如捕获的时间戳、
数据包的长度等
        const u_char* packetData; //指向捕获到的数据包

        int result = pcap_next_ex(point, &pkt_header, &packetData); //得到
pcap_next_ex的返回结果
        if (result == 0) { //未捕获到数据包
            cout << "在指定时间范围 (read_timeout)内未捕获到数据包" << endl;

```

```

        flag = 0;
        continue;
    }
    else if (result == -1) { //调用过程发生错误
        cout << "捕获数据包出错" << endl;
        break;
    }
    else { //result=1, 捕获成功

// 解析数据包
        // 提取源MAC地址（前6字节）
        cout << "源MAC地址: ";
        for (int i = 0; i < 6; ++i) {
            printf("%02X", packetData[i]);
            if (i < 5) cout << ":";
        }
        cout << endl;

        // 提取目的MAC地址（接下来的6字节）
        cout << "目的MAC地址: ";
        for (int i = 6; i < 12; ++i) {
            printf("%02X", packetData[i]);
            if (i < 11) cout << ":";
        }
        cout << endl;

        // 提取类型/长度字段的值（接下来的2字节）
        uint16_t type = (packetData[12] << 8) + packetData[13];
        cout << "类型/长度: " << hex << type << dec << endl;

        //提取源IP地址（IPv4头部中的26到30字节）
        cout << "源IP地址: ";
        for (int i = 26; i < 30; ++i) {
            printf("%d", packetData[i]);
            if (i < 29) cout << ".";
        }
        cout << std::endl;

        // 提取目的IP地址（IPv4头部中的30到34字节）
        std::cout << "目的IP地址: ";
        for (int i = 30; i < 34; ++i) {
            printf("%d", packetData[i]);
            if (i < 33) cout << ".";
        }
        cout << endl;
        cout << "-----" << endl;

    }
}

// 关闭设备
pcap_close(point);
}

pcap_freealldevs(devices); //释放网络接口设备列表
return 0;

```

```
}
```

4.3 关键步骤

`getaddress()`：检查 `sa_family` 是 `AF_INET` (IPv4) 还是 `AF_INET6` (IPv6)，然后返回对应的地址部分。

```
pcap_if_t* devices; // 指向设备列表的指针
char errbuf[PCAP_ERRBUF_SIZE]; // 用于存储错误信息
if (pcap_findalldevs(&devices, errbuf) == -1) {
    cerr << "查找设备失败：" << errbuf << endl;
    return 0;
}
```

`pcap_findalldevs(&devices, errbuf)`：调用 `pcap_findalldevs()` 获取设备列表。获取系统中的所有网络设备并返回一个链表。如果获取失败，函数返回 -1 并将错误信息保存在 `errbuf` 中，输出错误信息并退出程序。

```
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;
    continue;
}
```

`pcap_open()`：打开指定的网络设备并返回一个设备句柄，设置为混杂模式通过这个句柄进行数据包捕获。参数包括设备名称、最大捕获包大小（65536 字节）、混杂模式标志等。

参数设置：

`count2->name`：要打开的设备名称。

`65536`：捕获数据包的最大长度（单位：字节），可以确保程序可以抓到整个数据包。

`PCAP_OPENFLAG_PROMISCUOUS`：设置设备为混杂模式（捕获所有通过该网卡的数据包）。

`200`：超时时间为 200 毫秒。

`NULL`：不使用身份验证。

`errbuf`：存储可能的错误信息。

进入循环使用 `pcap_next_ex()` 捕获数据包。

```
int result = pcap_next_ex(point, &pkt_header, &packetData);
if (result == 1) { // 捕获成功
    // 解析源MAC地址
    for (int i = 0; i < 6; ++i) {
        printf("%02X", packetData[i]);
        if (i < 5) cout << ":";
    }
}
```

`pcap_next_ex()`：获取下一个数据包，返回值为 1 表示成功捕获，0 表示超时，-1 表示错误。

这是程序的**核心步骤**，用于捕获数据包。捕获成功后，程序将对数据包进行解析，从底层网络接口获取下一个数据包，返回数据包的相关信息（如时间戳、长度等），以及捕获到的数据包内容；如果超时或捕获出错，则跳出循环。

```
uint16_t type = (packetData[12] << 8) + packetData[13];
cout << "类型/长度: " << hex << type << dec << endl;
```

通过 `packetData[i]` 提取数据包的前 6 个字节为源 MAC 地址，接下来的 6 个字节为目的 MAC 地址。以太网帧的前 12 个字节分别存储了源 MAC 和目的 MAC 地址。数据包的第 12 和 13 个字节表示类型或长度字段，解析它可以确定数据包的类型（例如 IPv4、ARP 等）。通过这个字段可以进一步判断数据包携带的是哪种协议的数据。

`pcap_close(point)`：关闭当前设备的捕获会话。

`pcap_freealldevs(devices)`：释放已获取的设备列表，避免内存泄漏。

四、结果分析

1.打印设备信息如图所示：将所有通信网卡设备的名称及IP地址等相关信息打印在屏幕上：

```
1. \Device\NPF_{DFB0FA79-399E-4BF9-A863-CBC133FFA6D6}(VMware Virtual Ethernet Adapter)
IP地址: 192.168.40.1
子网掩码: 255.255.255.0
广播地址: 255.255.255.255
2. \Device\NPF_{6D587948-1932-41E1-92BD-12CD54F2AA28}( )
IP地址: 192.168.1.99
子网掩码: 255.255.255.0
广播地址: 255.255.255.255
3. \Device\NPF_{8EFB813B-166E-4299-B387-295D784D25A5}(Microsoft)
IP地址: 10.128.9.228
子网掩码: 255.255.128.0
广播地址: 255.255.255.255
4. \Device\NPF_{79E22CCC-172F-4192-BAAD-893F4B4286A5}(Microsoft)
5. \Device\NPF_{B89C8119-68DB-49EC-AEA8-76CA4D8DBA60}(VMware Virtual Ethernet Adapter)
IP地址: 192.168.153.1
子网掩码: 255.255.255.0
广播地址: 255.255.255.255
6. \Device\NPF_{A97D9DCD-F521-429E-B0BB-4046F95DA244}(Sangfor SSL VPN CS Support System VNIC)
IP地址: 0.0.0.0
子网掩码: 0.0.0.0
广播地址: 255.255.255.255
7. \Device\NPF_{21161504-E4CA-4F02-AB2C-3A9D6AF3C007}(Microsoft)
-----
```

2.抓捕数据包提取的信息如图所示：将当前使用的网络设备接口名字打印在屏幕上，且显示出抓捕数据包中的源MAC地址，目的MAC地址，类型/长度，源IP地址，目的IP地址：

```
-----
当前网络设备接口卡名字为: \Device\NPF_{DFB0FA79-399E-4BF9-A863-CBC133FFA6D6}
在指定时间范围 (read_timeout)内未捕获到数据包
当前网络设备接口卡名字为: \Device\NPF_{6D587948-1932-41E1-92BD-12CD54F2AA28}
在指定时间范围 (read_timeout)内未捕获到数据包
当前网络设备接口卡名字为: \Device\NPF_{8EFB813B-166E-4299-B387-295D784D25A5}
源MAC地址: 00:00:5E:00:01:0B
目的MAC地址: 70:A8:D3:B0:77:A0
类型/长度: 800
源IP地址: 10.128.9.228
目的IP地址: 36.150.95.206
-----
源MAC地址: 70:A8:D3:B0:77:A0
目的MAC地址: 00:00:5E:00:01:0B
类型/长度: 800
源IP地址: 36.150.95.206
目的IP地址: 10.128.9.228
-----
源MAC地址: 00:00:5E:00:01:0B
目的MAC地址: 70:A8:D3:B0:77:A0
类型/长度: 800
源IP地址: 10.128.9.228
目的IP地址: 36.150.95.206
-----
```

超时时间为 200 毫秒，如果在这个时间内没有捕获到数据包，则程序会提示未捕获到数据包。

部分设备在捕获时没有实际的网络通信，或者是虚拟设备，则会显示未捕获到数据包。

五、实验总结

在本次数据包捕获与解析实验中，通过使用 Npcap 库成功实现了对本地网络设备的捕获和解析，进一步加深了对网络数据传输和以太网帧结构的理解。实验的总体流程包括获取网络设备列表、打开设备进行数据包捕获，并对捕获的数据包进行解析，提取了关键信息如 MAC 地址、IP 地址等。

首先，通过使用 `pcap_findalldevs()` 函数可以轻松获取本机所有可用的网络接口设备。这一步的实现帮助我们了解了计算机中所有网卡设备的状态，甚至包括虚拟网卡和 VPN 适配器。随后，通过 `pcap_open()` 打开指定设备成功进入数据包捕获阶段。

在捕获数据包的过程中，逐一提取了以太网帧的源 MAC 地址、目的 MAC 地址，以及类型/长度字段。更进一步，通过解析 IPv4 头部，成功提取了数据包中的源 IP 和目的 IP。实验中，还遇到了一些挑战，例如捕获超时和处理错误等情况，但这些都通过合理的错误处理机制得到了妥善解决。

值得一提的是，实验不仅仅停留在书本理论的层面，更让我从实践中深刻体会到了网络协议的实际应用。通过解析数据包的各个字段，我们对数据链路层与网络层之间的交互有了更直观的理解，也对以太网帧、IP 数据报的结构有了更深刻的认识。

反思与改进

实验中，我虽然成功完成了基本功能，但也发现了可以改进的地方。例如，捕获的数据包有时可能因为网络繁忙而产生大量冗余信息，如果能够引入更灵活的过滤机制，针对特定协议或端口进行过滤捕获，效率会更高。另外，在数据包解析方面，虽然提取了基本字段，但可以尝试解析更高层协议（如 TCP/UDP），这样可以进一步提高对网络流量的分析能力。

实验收获

本次实验不仅让我掌握了如何利用 NPcap 库进行数据包捕获，还培养了我对网络流量分析的能力。通过实践，我更加理解了网络通信的底层原理，为后续的网络安全、协议分析打下了坚实的基础。