

网络技术与应用课程实验报告

姓名：孟启轩

学号：2212452

专业：计算机科学与技术

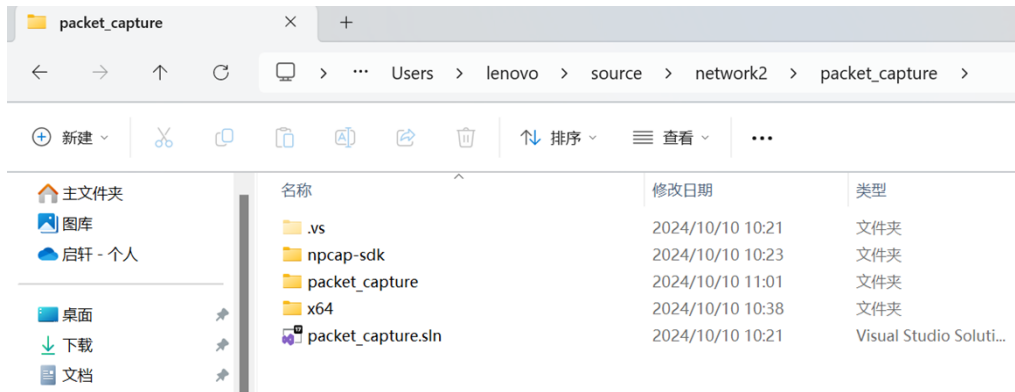
实验 2：数据包捕获与分析

一、实验内容：

- (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。

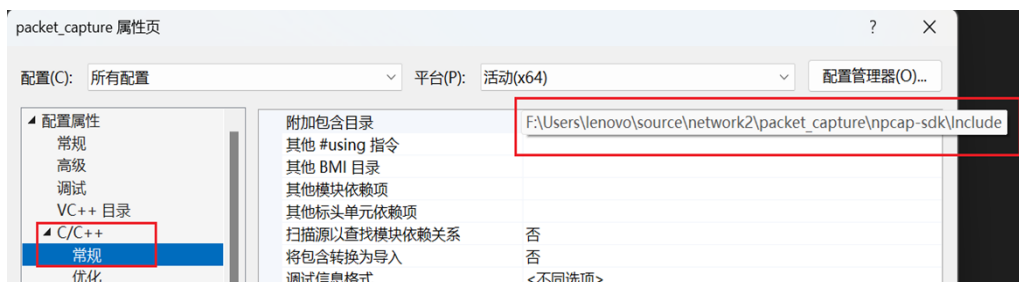
二、实验准备：

使用 visual studio 2022 进行编程实现，首先，到 Npcap 官网下载 Npcap 驱动程序和 DLL 程序以及 sdk 依赖，在 visual studio 中新建项目，并将 npcac 中的 sdk 移至项目目录下。



再进入项目文件中对 npcac 进行相关配置：

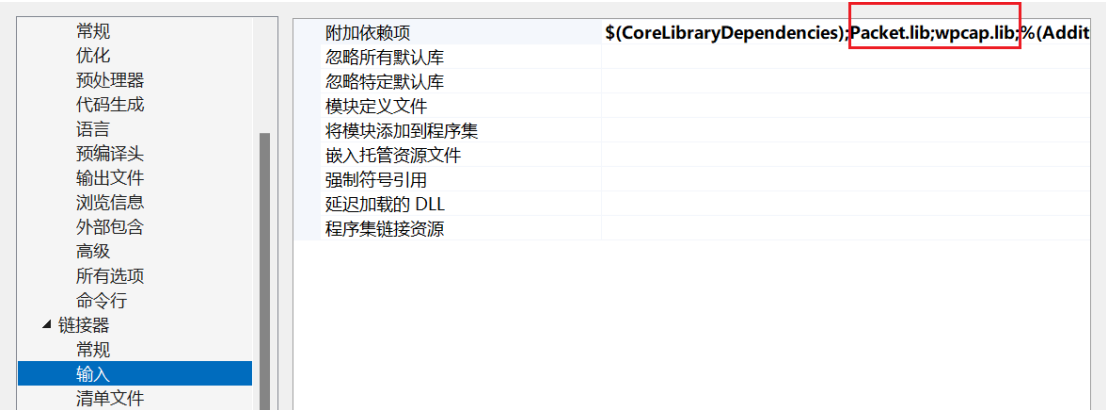
在 C/C++ 属性的常规属性中，将 include 目录添加至附加包含目录中



在链接器的附加库目录中添加 Lib 目录（注意这里一定要添加 Lib 中的 X64）



在输入的附加依赖项中添加 Packet.lib 和 wpcap.lib



三、实验过程：

(1) 了解 NPcap 的架构。

NPcap 是 Windows 平台上的高性能网络数据包捕获库，它基于 WinPcap 项目的分支，专门为现代 Windows 系统设计，提供了增强的功能和改进的性能。NPcap 的架构主要包括以下几个关键组件：

1. 内核级驱动程序 (Kernel-Level Driver)

NPcap 的核心是一个位于内核级别的驱动程序，它直接与 Windows 网络适配器进行交互。该驱动程序允许捕获通过网络适配器发送和接收的数据包，实现低延迟和高吞吐量的网络数据包捕获。此外，它还支持数据包注入功能，能够将用户自定义的数据包发送到网络上。内核级驱动的设计使得 NPcap 能够深入操作系统网络栈，提供准确和高效的流量捕获。

2. 用户级 API (User-Level API)

NPcap 提供用户级别的 API，方便开发者进行应用程序开发。开发者可以通过 API 配置捕获参数、启动或停止数据包捕获，以及访问和处理捕获的数据包。NPcap 的 API 兼容 WinPcap 接口，支持多种编程语言（如 C/C++ 和 Python），使得已有的 WinPcap 应用程序可以较为轻松地迁移到 NPcap。

3. 数据包过滤与注入 (Packet Filtering and Injection)

NPcap 支持使用 BPF (Berkeley Packet Filter) 语法定义数据包过滤规则，这些规则可以基于 IP 地址、端口、协议类型等条件进行配置，从而选择需要捕获的数据包类型。NPcap 还支持数据包注入功能，允许开发者发送自定义的数据包到网络上，从而实现模拟网络通信、测试和调试等功能。

4. 回环捕获 (Loopback Capture)

NPcap 支持本地回环接口的捕获，可以捕获本地应用程序之间的通信流量。该功能在调试和分析本地服务、检测本地系统行为时非常有用。这一功能弥补了 WinPcap 不支持回

环捕获的不足，使 Npcap 成为更全面的网络流量分析工具。

5. 辅助库和工具 (Libraries and Utilities)

Npcap 提供了一些辅助库和实用工具，帮助开发者方便地与 Npcap 进行交互。这些工具可以用于检测可用的网络适配器、配置适配器参数、分析和解码捕获的数据包等，提升开发效率。

6. 与 Wireshark 的集成 (Integration with Wireshark)

Npcap 常常被用于与 Wireshark 等网络协议分析工具相结合，以实现全面的网络数据分析解决方案。Wireshark 能够与 Npcap 无缝集成，用户可以使用 Wireshark 通过 Npcap 捕获数据包，并对数据包进行深入的协议分析和可视化展示。这种组合使 Npcap 成为网络安全、故障排查和网络性能优化等领域的重要工具。

7. 安全增强和性能改进

Npcap 在 WinPcap 的基础上做了许多改进。它增加了安全性措施，如对驱动程序签名的严格要求、在捕获过程中默认禁用混杂模式等。此外，Npcap 采用了更高效的内存管理机制和改进的驱动架构，显著提升了数据包捕获的性能和可靠性。

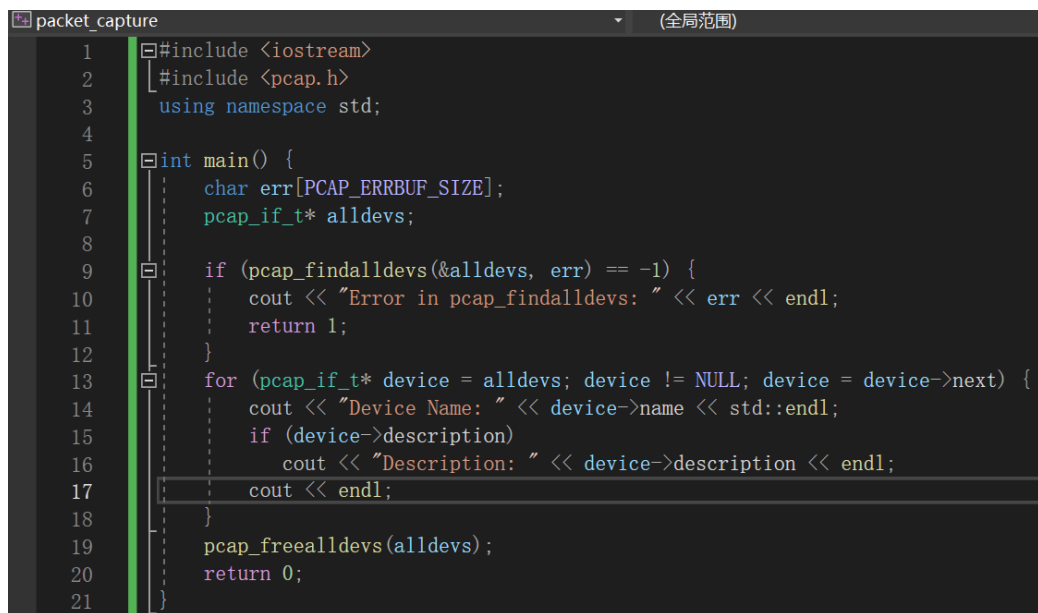
综上所述，Npcap 通过内核级驱动程序、灵活的用户级 API 以及多种实用功能，如数据包过滤、注入和回环捕获等，为网络流量分析、网络调试和安全监测等任务提供了强大的工具支持。它的广泛应用不仅限于开发者，还包括网络安全专家、系统管理员和网络工程师等专业人员。

(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。

1. 网络接口设备（网卡）列表获取方法

NpCap 提供了 `pcap_findalldevs_ex` 和 `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。除此之外，`pcap_if_t` 结构体还包含了一个 `pcap_addr` 结构体；后者包含了一个地址列表、一个掩码列表、一个广播地址列表和一个目的地址的列表；此外，`pcap_findalldevs_ex` 还能返回远程适配器信息和一个位于所给的本地文件夹的 `pcap` 文件列表。

编写代码获取设备列表：



```
1  #include <iostream>
2  #include <pcap.h>
3  using namespace std;
4
5  int main() {
6      char err[PCAP_ERRBUF_SIZE];
7      pcap_if_t* alldevs;
8
9      if (pcap_findalldevs(&alldevs, err) == -1) {
10         cout << "Error in pcap_findalldevs: " << err << endl;
11         return 1;
12     }
13     for (pcap_if_t* device = alldevs; device != NULL; device = device->next) {
14         cout << "Device Name: " << device->name << std::endl;
15         if (device->description)
16             cout << "Description: " << device->description << endl;
17         cout << endl;
18     }
19     pcap_freealldevs(alldevs);
20     return 0;
21 }
```

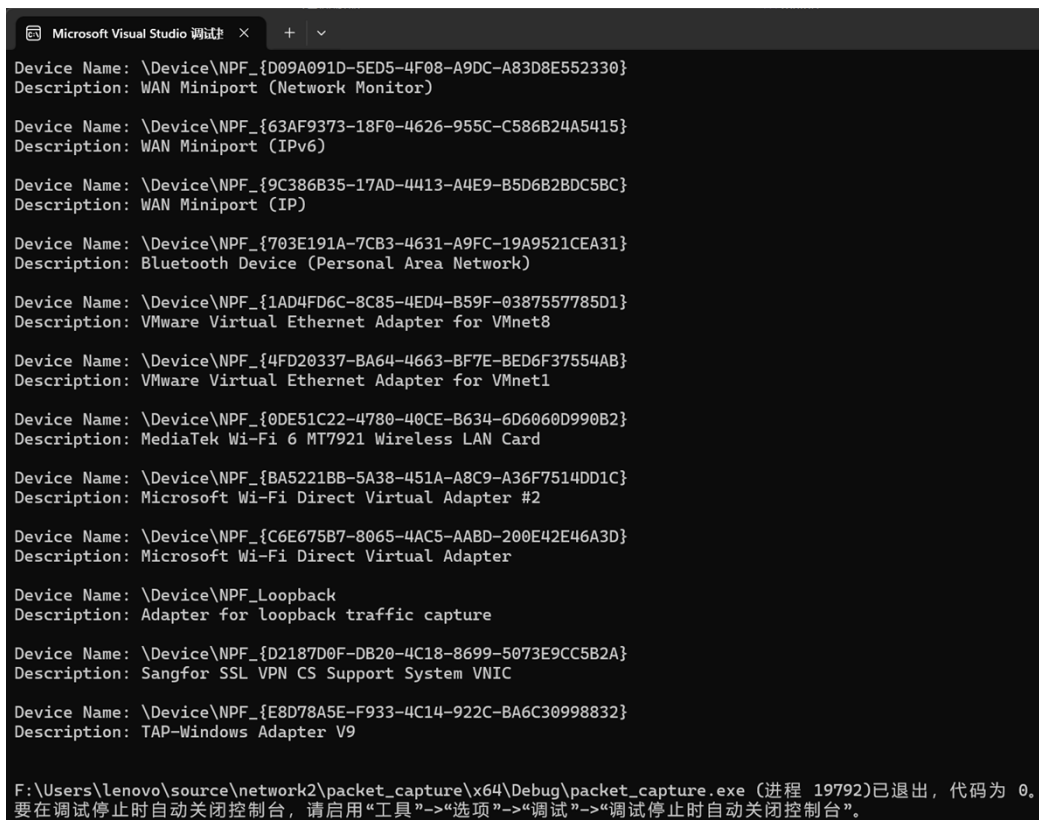
此程序的编写思路：定义一个 `err` 数组用来接受错误信息，定义一个指向 `pcap_if_t`

结构体的指针：

```
pcap_if*   next, // 指向下一个元素，为NULL则表示该元素为列表的最后一个元素
char*      name, // 表示该网络设备的名称
char*      description, // 表示该网络设备的描述
pcap_addr* address, // 该网络设备中的地址信息，包括IPV4、IPV6、子网掩码等
u_int      flags, // 用于表示是否为回环端口 (loopback interface)
```

由于 pcap_findalldevs 函数传入 pcap_if_t* 类型的参数，再对 alldevs 进行取地址操作，若这个函数的返回值为-1，证明获取设备列表有误，结束程序。使用遍历输出 alldevs 中的数据，当 device 指向不为空，则证明有对应的 pcap_if_t，进行输出，若 device 指向的 description 不为空，则输出 description。

输出的网络接口设备（网卡）列表如下：



```
Microsoft Visual Studio 调试  x + -
Device Name: \Device\NPF_{D09A091D-5ED5-4F08-A9DC-A83D8E552330}
Description: WAN Miniport (Network Monitor)

Device Name: \Device\NPF_{63AF9373-18F0-4626-955C-C586B24A5415}
Description: WAN Miniport (IPv6)

Device Name: \Device\NPF_{9C386B35-17AD-4413-A4E9-B5D6B2BDC5BC}
Description: WAN Miniport (IP)

Device Name: \Device\NPF_{703E191A-7CB3-4631-A9FC-19A9521CEA31}
Description: Bluetooth Device (Personal Area Network)

Device Name: \Device\NPF_{1AD4FD6C-8C85-4ED4-B59F-0387557785D1}
Description: VMware Virtual Ethernet Adapter for VMnet8

Device Name: \Device\NPF_{4FD20337-BA64-4663-BF7E-BED6F37554AB}
Description: VMware Virtual Ethernet Adapter for VMnet1

Device Name: \Device\NPF_{0DE51C22-4780-40CE-B634-6D6060D990B2}
Description: MediaTek Wi-Fi 6 MT7921 Wireless LAN Card

Device Name: \Device\NPF_{BA5221BB-5A38-451A-A8C9-A36F7514DD1C}
Description: Microsoft Wi-Fi Direct Virtual Adapter #2

Device Name: \Device\NPF_{C6E675B7-8065-4AC5-AABD-200E42E46A3D}
Description: Microsoft Wi-Fi Direct Virtual Adapter

Device Name: \Device\NPF_{Loopback}
Description: Adapter for loopback traffic capture

Device Name: \Device\NPF_{D2187D0F-DB20-4C18-8699-5073E9CC5B2A}
Description: Sangfor SSL VPN CS Support System VNIC

Device Name: \Device\NPF_{E8D78A5E-F933-4C14-922C-BA6C30998832}
Description: TAP-Windows Adapter V9

F:\Users\lenovo\source\network2\packet_capture\x64\Debug\packet_capture.exe (进程 19792)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”。
```

2. 网卡设备打开方法

`pcap_open_live()` 是一个用于获取数据包捕获描述符的函数，开发者可以通过它来查看网络上的数据包。该函数返回一个 `pcap_t` 类型的 `handle`。其主要参数和功能说明如下：

①device：这是一个字符串参数，用于指定要打开的网络设备。在 Linux 系统（内核版本 2.2 或更高）上，可以使用特殊的设备名称 `"any"` 或者 `NULL` 来捕获来自所有接口的数据包。

②snaplen：定义要捕获的最大字节数。如果捕获的数据包大小超过 `snaplen`，那么只会捕获数据包的前 `snaplen` 字节，并将其作为数据包数据提供。对于大多数网络，65535 的值通常足以捕获数据包中所有可用的数据。

③promisc：用于指定是否将网络接口置于混杂模式。如果设置为 `true`，接口将进入混杂模式，从而捕获经过网络的所有数据包，而不仅仅是发送到该接口的包。请注意，即使设置为 `false`，网络接口可能由于其他原因处于混杂模式。对于设备 `"any"` 或

`NULL`，该参数会被忽略。


④to_ms：指定读取操作的超时时间（以毫秒为单位）。读取超时允许在捕获到数据包后稍作等待，以便获取更多数据包，从而减少频繁的读取操作。这样可以在单次操作中从操作系统内核读取多个数据包，提高捕获效率。

⑤errbuf：用于存储错误信息字符串的缓冲区

通过这些参数，`pcap_open_live()` 提供了一种灵活的方式来设置数据包捕获行为，适用于各种网络流量分析和监控需求。

进行编程实现与验证：

```
27 #include <pcap.h>
28 using namespace std;
29 #include<iostream>
30 int main() {
31     char errbuf[PCAP_ERRBUF_SIZE]; //用以接受错误信息
32     const char* deviceName = "\\Device\\NPF_{D09A091D-5ED5-4F08-A9DC-A83D8E552330}";
33     pcap_t* dev;
34
35     dev = pcap_open_live(deviceName, 65535, 1, 1000, errbuf);
36
37     if (dev == NULL) {
38         cout<<"Could not open" <<deviceName;
39         return 1;
40     }
41     else {
42         cout << "you succeed";
43     }
44     // 成功打开网卡设备，可以开始捕获数据包
45
46     pcap_close(dev);
47
48     return 0;
49 }
```



3. 数据包捕获方法

在使用 pcap_open_live 打开网卡设备后，输入要查看的网卡编号，手动编写死循环，使用 pcap_next_ex 捕获数据包，捕获成功时对数据包进行打印，捕获失败或捕获结束时结束循环。

完成捕获后释放网卡设备并关闭 pcap 会话句柄。

```

52 #include <pcap.h>
53 #include <iostream>
54 using namespace std;
55
56 int main() {
57     int device_num = 0;
58     pcap_if_t* alldevs;
59     char err[PCAP_ERRBUF_SIZE]; // 用以接受错误信息
60
61     if (pcap_findalldevs(&alldevs, err) == -1) {
62         cerr << "Error in pcap_findalldevs: " << err << endl;
63         return 1;
64     }
65
66     for (pcap_if_t* device = alldevs; device != nullptr; device = device->next) {
67         cout << "Device Name: " << device->name << endl;
68         device_num++;
69         if (device->description)
70             cout << "Description: " << device->description << endl;
71         cout << endl;
72     }
73     cout << "Total devices: " << device_num << endl;
74
75     int device_number;
76     cout << "请输入要查找的设备: ";
77     cin >> device_number;
78
79     if (device_number >= device_num) {
80         cerr << "无效的设备编号" << endl;
81         pcap_freealldevs(alldevs);
82         return 1;
83     }
84     pcap_if_t* device = alldevs;
85     for (int i = 0; i < device_number; i++) {
86         device = device->next;
87     }
88     pcap_t* handle = pcap_open_live(device->name, BUFSIZ, 1, 1000, err);
89     while (true) {
90         struct pcap_pkthdr* header;
91         const u_char* packet;
92         int result = pcap_next_ex(handle, &header, &packet);
93         if (result == 1) {
94             cout << "----- Got a packet with length of " << header->len << " -----" << endl;
95             // 打印数据包的简要信息
96             for (int i = 0; i < header->len; i++) {
97                 cout << hex << (int)packet[i] << " ";
98             }
99             cout << endl;
100         }
101         else if (result == 0) {
102             // 未收到数据包，继续捕获。
103         }
104         else if (result == -1) {
105             cerr << "pcap_next_ex error: " << pcap_geterr(handle) << endl;
106             break;
107         }
108         else if (result == -2) {
109             cerr << "结束" << endl;
110             break;
111         }
112     }
113     pcap_freealldevs(alldevs);
114     pcap_close(handle);
115     return 0;
116 }

```

捕获的数据包如下：


```
F:\Users\lenovo\source\netw... x + v

76 63 4 5f 74 63 70 5 6c 6f 63 61 6c 0 0 ff 0 1
===== Got a packet with length of 5d =====
1 0 5e 0 0 fb 0 50 56 c0 0 1 8 0 45 0 0 4f 9a e4 0 0 1 11 8f 14 c0 a8 ee 1 e0 0 0 fb 14 e9 14 e9 0 3b 72 74 0 0 0 0 1
0 0 0 0 0 0 f 4c 41 50 54 4f 50 2d 32 42 43 47 44 34 4a 49 6 5f 64 6f 73 76 63 4 5f 74 63 70 5 6c 6f 63 61 6c 0 0 ff 0 1

===== Got a packet with length of 71 =====
33 33 0 0 0 fb 0 50 56 c0 0 1 86 dd 60 e f7 e1 0 3b 11 1 fe 80 0 0 0 0 0 2f 8e 5f 19 69 90 46 82 ff 2 0 0 0 0 0 0 0 0
0 0 0 0 0 fb 14 e9 14 e9 0 3b c4 e1 0 0 0 0 0 1 0 0 0 0 0 0 f 4c 41 50 54 4f 50 2d 32 42 43 47 44 34 4a 49 6 5f 64 6f 73
76 63 4 5f 74 63 70 5 6c 6f 63 61 6c 0 0 ff 0 1

===== Got a packet with length of 5d =====
1 0 5e 0 0 fb 0 50 56 c0 0 1 8 0 45 0 0 4f 9a e5 0 0 1 11 8f 13 c0 a8 ee 1 e0 0 0 fb 14 e9 14 e9 0 3b 72 74 0 0 0 0 1
0 0 0 0 0 0 f 4c 41 50 54 4f 50 2d 32 42 43 47 44 34 4a 49 6 5f 64 6f 73 76 63 4 5f 74 63 70 5 6c 6f 63 61 6c 0 0 ff 0 1

===== Got a packet with length of 71 =====
33 33 0 0 0 fb 0 50 56 c0 0 1 86 dd 60 e f7 e1 0 3b 11 1 fe 80 0 0 0 0 0 2f 8e 5f 19 69 90 46 82 ff 2 0 0 0 0 0 0 0 0
0 0 0 0 0 fb 14 e9 14 e9 0 3b c4 e1 0 0 0 0 0 1 0 0 0 0 0 0 f 4c 41 50 54 4f 50 2d 32 42 43 47 44 34 4a 49 6 5f 64 6f 73
76 63 4 5f 74 63 70 5 6c 6f 63 61 6c 0 0 ff 0 1

===== Got a packet with length of 12b =====
1 0 5e 0 0 fb 0 50 56 c0 0 1 8 0 45 0 1 1d 9a e6 0 0 1 11 8e 44 c0 a8 ee 1 e0 0 0 fb 14 e9 14 e9 1 9 2d 3c 0 0 84 0 0 0
0 1 0 0 0 4 6 5f 64 6f 73 76 63 4 5f 74 63 70 5 6c 6f 63 61 6c 0 0 c 80 1 0 0 11 94 0 23 f 4c 41 50 54 4f 50 2d 32 42 43
47 44 34 4a 49 6 5f 64 6f 73 76 63 4 5f 74 63 70 5 6c 6f 63 61 6c 0 f 4c 41 50 54 4f 50 2d 32 42 43 47 44 34 4a 49 6 5f
64 6f 73 76 63 4 5f 74 63 70 5 6c 6f 63 61 6c 0 0 21 80 1 0 0 78 0 1d 0 0 0 1e 0 f 4c 41 50 54 4f 50 2d 32 42 43 47
44 34 4a 49 5 6c 6f 63 61 6c 0 c0 4c 0 10 80 1 0 0 11 94 0 1e 7 50 3d 36 35 32 38 30 15 53 48 30 3d 6e 59 51 6b 68 5
8 4e 2b 5a 4e 62 70 63 66 39 2b f 4c 41 50 54 4f 50 2d 32 42 43 47 44 34 4a 49 5 6c 6f 63 61 6c 0 0 1 80 1 0 0 3c 0 4
c0 a8 ee 1 c0 c0 1c 80 1 0 0 3c 0 10 fe 80 0 0 0 0 0 2f 8e 5f 19 69 90 46 82

===== Got a packet with length of eb =====
1 0 5e 0 0 fb 0 50 56 c0 0 1 8 0 45 0 0 dd 9a e7 0 0 1 11 8e 83 c0 a8 ee 1 e0 0 0 fb 14 e9 14 e9 0 c9 96 cc 0 0 84 0 0 0
0 1 0 0 0 3 f 4c 41 50 54 4f 50 2d 32 42 43 47 44 34 4a 49 6 5f 64 6f 73 76 63 4 5f 74 63 70 5 6c 6f 63 61 6c 0 0 21 80
1 0 0 0 78 0 1d 0 0 0 0 1e 0 f 4c 41 50 54 4f 50 2d 32 42 43 47 44 34 4a 49 5 6c 6f 63 61 6c 0 c0 c 0 10 80 1 0 0 11 94
0 1e 7 50 3d 36 35 32 38 30 15 53 48 30 3d 6e 59 51 6b 68 58 4e 2b 5a 4e 62 70 63 66 39 2b f 4c 41 50 54 4f 50 2d 32
```

(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) 捕获的数据报不要求硬盘存储，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。

编写程序对已抓到的数据包进行分析，输出源 MAC 地址、目的 MAC 地址和类型/长度字段的值, 总长度, 生存周期, 协议, 头部校验码。代码见 `packet_capture.cpp`

运行结果:

```
F:\Users\lenovo\source\netw... x + v

=====
以太网类型 : 0800
IPv4
Mac源地址:
00:50:56:c0:00:01:
Mac目的地址:
01:00:5e:7f:ff:fa:
版本: 4
IP协议首部长度: 20 Bytes
总长度: 193
生存时间:
协议号: 124
协议种类: 首部检验和: 0x0
源地址: 1.17.78.11
目的地址: 192.168.238.1
=====
以太网类型 : 0800
IPv4
Mac源地址:
00:50:56:c0:00:01:
Mac目的地址:
01:00:5e:7f:ff:fa:
版本: 4
IP协议首部长度: 20 Bytes
总长度: 193
生存时间:
协议号: 125
协议种类: 首部检验和: 0x0
源地址: 1.17.78.10
目的地址: 192.168.238.1
=====
```

四、实验感悟:

通过本次 Npcap 数据包捕获与分析实验，我深入理解了 Npcap 的架构及其高效的数据包捕获机制，掌握了设备列表获取、网络接口打开和数据包捕获的基本流程，并通过编程实践提取了数据帧的关键信息。实验不仅强化了我对网络协议的理解，还培养了良好的编程习惯和代码规范意识，为后续的实验打下了坚实的基础。