

# 网络技术与应用课程实验报告

姓名：汤志文

学号：2111441

专业：物联网工程

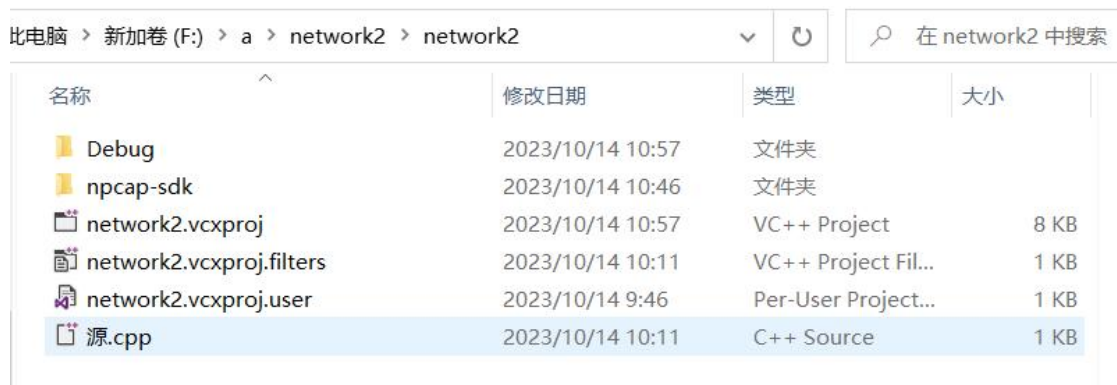
## 实验 2：数据包捕获与分析

### 一、实验内容：

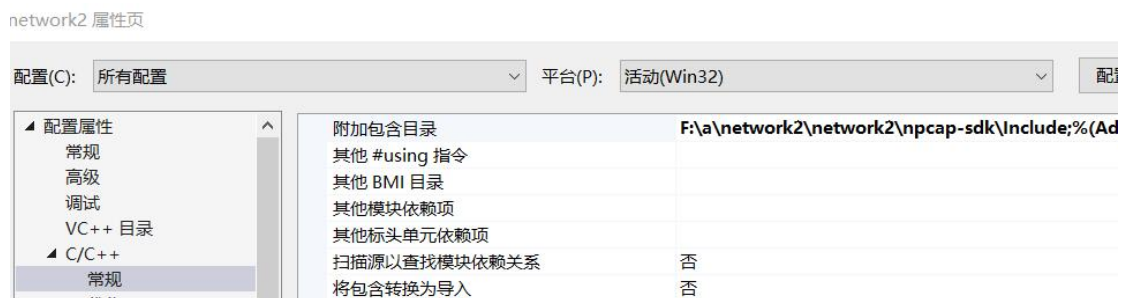
- (1) 了解 NPcap 的架构。
- (2) 学习 NPcap 的设备列表获取方法、网卡设备打开方法，以及数据包捕获方法。
- (3) 通过 NPcap 编程，实现本机的数据包捕获，显示捕获数据帧的源 MAC 地址和目的 MAC 地址，以及类型/长度字段的值。
- (4) 捕获的数据报不要求硬盘存储，但应以简单明了的方式在屏幕上显示。必显字段包括源 MAC 地址、目的 MAC 地址和类型/长度字段的值。
- (5) 编写的程序应结构清晰，具有较好的可读性。

### 二、实验准备：

使用 visual studio 2019 进行编程实现，首先，将老师提供的 npcap 文件进行解压安装，在 visual studio 中新建项目，并将 npcap 中的 sdk 移至项目目录下



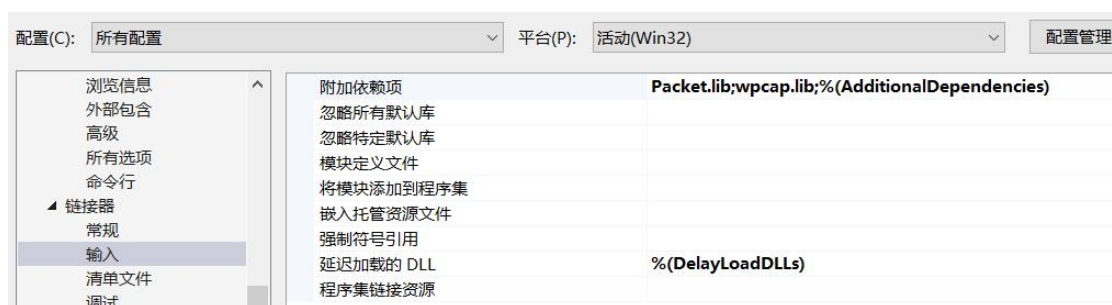
再进入项目文件中对 npcap 进行相关配置：



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



在输入的附加依赖项中添加 Packet.lib wpcap.lib

在运行过程中发现 wpcap.dll 丢失,查找解决方案:64 位系统,在 C:\Windows\SysWOW64 目录下找到 npcap 文件夹,将其中的 wpcap.dll 移动至 C:\Windows\SysWOW64 中,并运行 regsvr32 C:\Windows\SysWOW64\wpcap.dll 将该 dll 进行配置,再使用 visual studio 验证,在包含头文件并编写设备获取程序后正常运行,说明环境配置成功。接下来继续进行实验。

### 三、实验过程:

#### (1) 了解 NPcap 的架构。

NPcap 是 Windows 平台上的一个网络数据包捕获库,它是 WinPcap 的一个分支,用于在 Windows 系统上进行数据包捕获和网络流量分析。下面是 NPcap 的基本架构:

##### 1. Kernel-Level Driver:

NPcap 的核心组件是一个位于内核级别的驱动程序。这个驱动程序允许 NPcap 与网络适配器交互,捕获和注入数据包,以及提供高性能的数据包处理。它与 Windows 内核紧密集成,允许捕获从网络适配器接收到的数据包以及发送给网络适配器的数据包。

##### 2. User-Level API:

为了方便开发者和应用程序与 NPcap 进行交互,NPcap 提供了一个用户级别的 API。这个 API 允许开发者编写应用程序来配置捕获参数、启动和停止数据包捕获,以及处理捕获的数据包。开发者可以使用不同的编程语言,如 C/C++、Python、或者其他支持的语言来访问 NPcap API。

##### 3. Packet Filtering and Injection:

NPcap 允许用户定义特定的数据包过滤规则,以选择要捕获的数据包类型。这些过滤规则基于数据包的源地址、目标地址、端口号、协议类型等等。此外,NPcap 还支持数据包的注入,允许应用程序将构建的数据包发送到网络适配器上。

#### 4. Loopback Capture:

NPcap 还支持捕获本地回环 (Loopback) 流量, 这是一个非常有用的功能, 用于分析本地计算机与本地应用程序之间的通信。

#### 5. Libraries and Utilities:

NPcap 提供了一些辅助库和实用工具, 帮助开发者更容易地与 NPcap 进行交互。这包括用于配置适配器、检测可用适配器、分析捕获数据的库和工具。

#### 6. Integration with Wireshark:

NPcap 通常与 Wireshark 等网络协议分析工具一起使用, 以提供一个全面的网络流量分析解决方案。Wireshark 能够与 NPcap 集成, 允许用户捕获数据包并对其进行详细的分析。

它能够提供内核级别的数据包捕获和用户级别的 API, 为开发者提供了广泛的控制和定制能力。这使得它成为网络安全、网络监控和网络调试等领域的重要工具。

(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. 设备列表获取方法: 获取本地网络设备列表是通过函数 `pcap_findalldevs` 实现的, 这个函数接受两个参数。

```
int pcap_findalldevs(pcap_if_t * *alldevsp, char* errbuf);  
// alldevsp: 存储本地网络设备列表  
// errbuf: 存储错误信息  
// 返回值: int, 如果发生错误将返回-1, 执行成功将返回0
```

编写代码获取设备列表:

```
#include <iostream>  
#include <pcap.h>  
using namespace std;  
  
int main() {  
    char err[PCAP_ERRBUF_SIZE];  
    pcap_if_t* alldevs;  
  
    if (pcap_findalldevs(&alldevs, err) == -1) {  
        cout << "Error in pcap_findalldevs: " << err << endl;  
        return 1;  
    }  
  
    for (pcap_if_t* device = alldevs; device != NULL; device = device->next) {  
        cout << "Device Name: " << device->name << std::endl;  
        if (device->description)  
            cout << "Description: " << device->description << endl;  
        cout << endl;  
    }  
  
    pcap_freealldevs(alldevs);  
    return 0;  
}
```

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

```

Device Name: \Device\NPF_{AD31D546-85F2-4197-83B1-7A71BC2D2F6B}
Description: WAN Miniport (Network Monitor)

Device Name: \Device\NPF_{B0E297E9-9E9F-4609-AE0C-FDDB0C7DD7E6}
Description: WAN Miniport (IPv6)

Device Name: \Device\NPF_{C963CE36-03E0-480C-B736-363B3DDB3A64}
Description: WAN Miniport (IP)

Device Name: \Device\NPF_{F83E916E-0D85-464A-A474-73D9567A936B}
Description: Bluetooth Device (Personal Area Network)

Device Name: \Device\NPF_{A456CE68-5523-481A-820C-EFE500CE411E}
Description: Intel(R) Wi-Fi 6 AX201 160MHz

Device Name: \Device\NPF_{F608CB84-2A76-4506-B7C4-3D3AD20C2EC9}
Description: VMware Virtual Ethernet Adapter for VMnet8

Device Name: \Device\NPF_{0DD027BE-4B80-4C37-B5F1-C6A58AE25018}
Description: VMware Virtual Ethernet Adapter for VMnet1

Device Name: \Device\NPF_{4F842E4D-216D-45AE-8446-5A621F329B23}
Description: Microsoft Wi-Fi Direct Virtual Adapter #2

Device Name: \Device\NPF_{77415E18-6161-4BA4-A1BE-932B82265B91}
Description: Microsoft Wi-Fi Direct Virtual Adapter

Device Name: \Device\NPF_{724838BC-EDF0-4C00-BD4E-32EE07BC62B4}
Description: VirtualBox Host-Only Ethernet Adapter #2

Device Name: \Device\NPF_{D480DCEF-51B3-4BF4-AFF1-336489229BEA}
Description: VirtualBox Host-Only Ethernet Adapter

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

Device Name: \Device\NPF_{95F8E752-DC52-4283-BB13-7ABB2C3C4156}
Description: Sangfor SSL VPN CS Support System VNIC

Device Name: \Device\NPF_{19794347-E819-4615-8673-CF0BCEB4ABB2}
Description: Realtek Gaming GbE Family Controller

```

输出的设备列表如上。

## 2. 网卡设备打开方法

pcap\_open\_live() 用于获取数据包捕获描述符以查看网络上的数据包。

device 是指定要打开的网络设备的字符串；在具有 2.2 或更高版本内核的 Linux 系统上，可以使用“any”或 NULL 的设备参数来捕获来自所有接口的数据包。

snaplen 指定要捕获的最大字节数。如果该值小于捕获的数据包的大小，则该数据包的第一个 snaplen 字节将被捕获并作为数据包数据提供。在大多数（如果不是全部）网络中，值 65535 应该足以捕获数据包中可用的所有数据。

promisc 指定是否将接口置于混杂模式。（注意，即使这个参数是假的，接口也可能在混杂模式下出于某些其他原因。）现在，这不适用于“任何”设备；如果提供了“any”或 NULL 的参数，则将忽略 promisc 标志。

to\_ms 指定读取超时（以毫秒为单位）。读取超时用于安排在看到数据包时读取不一定立即返回，但是在一次操作中，等待一段时间才能允许更多数据包到达并从操作系统内核读取多个数据包。

进行编程实现：

```
#include <pcap.h>
using namespace std;
#include <iostream>

int main() {
    char errbuf[PCAP_ERRBUF_SIZE]; //用以接受错误信息
    const char* deviceName = "\\Device\\NPF_{B0E297E9-9E9F-4609-AE0C-FDDB0C7DD7E6}"; //用IPv6 网络通信的 WAN Miniport 设备进行测试
    pcap_t* dev;

    dev = pcap_open_live(deviceName, 65535, 1, 1000, errbuf);

    if (dev == NULL) {
        cout << "Could not open" << deviceName;
        return 1;
    }
    else {
        cout << "you succeed";
    }
    // 成功打开网卡设备，可以开始捕获数据包

    pcap_close(dev);

    return 0;
}
```

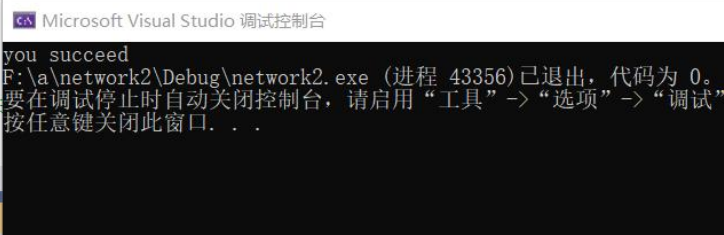
运行验证：

```
char errbuf[PCAP_ERRBUF_SIZE]; //用以接受错误信息
const char* deviceName = "\\Device\\NPF_{B0E297E9-9E9F-4609-AE0C-FDDB0C7DD7E6}"; //用IPv6 网络通信的 WAN Miniport 设备进行测试
pcap_t* dev;

dev = pcap_open_live(deviceName, 65535, 1, 1000, errbuf);

if (dev == NULL) {
    cout << "Could not open" << deviceName;
    return 1;
}
else {
    cout << "you succeed";
}
// 成功打开网卡设备，可以开始捕获数据包

pcap_close(dev);
```



Microsoft Visual Studio 调试控制台

```
you succeed
F:\a\network2\Debug\network2.exe (进程 43356)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”
按任意键关闭此窗口。...
```

### 3. 数据包捕获方法

```

#include <pcap.h>
#include <iostream>

int main() {
    int device_num = 0;
    pcap_if_t* alldevs;
    char err[PCAP_ERRBUF_SIZE]; // 用以接受错误信息

    if (pcap_findalldevs(&alldevs, err) == -1) {
        std::cerr << "Error in pcap_findalldevs: " << err << std::endl;
        return 1;
    }

    for (pcap_if_t* device = alldevs; device != nullptr; device = device->next) {
        std::cout << "Device Name: " << device->name << std::endl;
        device_num++;
        if (device->description)
            std::cout << "Description: " << device->description << std::endl;
        std::cout << std::endl;
    }
    std::cout << "Total devices: " << device_num << std::endl;

    int device_number;
    std::cout << "请输入要查找的设备: ";
    std::cin >> device_number;

    if (device_number >= device_num) {
        std::cerr << "无效的设备编号" << std::endl;
        pcap_freealldevs(alldevs);

        return 1;
    }

    pcap_if_t* device = alldevs;
    for (int i = 0; i < device_number; i++) {
        device = device->next;
    }

    pcap_t* handle = pcap_open_live(device->name, BUFSIZ, 1, 1000, err);
    while (true) {
        struct pcap_pkthdr* header;
        const u_char* packet;
        int result = pcap_next_ex(handle, &header, &packet);
        if (result == 1) {
            std::cout << "==== Got a packet with length of " << header->len << " =====" << std::endl;
            // 打印数据包的简要信息
            for (int i = 0; i < header->len; i++) {
                std::cout << std::hex << (int)packet[i] << " ";
            }
            std::cout << std::endl;
        }
        else if (result == 0) {
            // 未收到数据包, 继续捕获。
        }
        else if (result == -1) {
            std::cerr << "pcap_next_ex error: " << pcap_geterr(handle) << std::endl;
            break;
        }
        else if (result == -2) {
            std::cerr << "结束" << std::endl;
            break;
        }
    }

    pcap_freealldevs(alldevs);
    pcap_close(handle);
    return 0;
}

```



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。

捕获的数据包如下:

```
2 F:\a\network2\Release\network2.exe
===== Got a packet with length of d9 =====
1 0 5e 7f ff fa 0 50 56 c0 0 8 8 0 45 0 0 cb 46 65 0 0 1 11 73 19 c0 a8 4f 1 ef ff ff fa f0 46 7 6c 0 b7 7e 8b 4d 2d 53
45 41 52 43 48 20 2a 20 48 54 54 50 2f 31 2e 31 d a 48 4f 53 54 3a 20 32 33 39 2e 32 35 35 2e 32 35 35 2e 32 35 30 3a 31
39 30 30 d a 4d 41 4e 3a 20 22 73 73 64 70 3a 64 69 73 63 6f 76 65 72 22 d a 4d 58 3a 20 31 d a 53 54 3a 20 75 72 6e 3a
64 69 61 6c 2d 6d 75 6c 74 69 73 63 72 65 65 6e 2d 6f 72 67 3a 73 65 72 76 69 63 65 3a 64 69 61 6c 3a 31 d a 55 53 45 5
2 2d 41 47 45 4e 54 3a 20 4d 69 63 72 6f 73 6f 66 74 20 45 64 67 65 2f 31 31 38 2e 30 2e 32 30 38 38 2e 34 36 20 57 69 6
e 64 6f 77 73 d a d a
===== Got a packet with length of d9 =====
1 0 5e 7f ff fa 0 50 56 c0 0 8 8 0 45 0 0 cb 46 66 0 0 1 11 73 18 c0 a8 4f 1 ef ff ff fa f0 4b 7 6c 0 b7 95 ab 4d 2d 53
45 41 52 43 48 20 2a 20 48 54 54 50 2f 31 2e 31 d a 48 4f 53 54 3a 20 32 33 39 2e 32 35 35 2e 32 35 35 2e 32 35 30 3a 31
39 30 30 d a 4d 41 4e 3a 20 22 73 73 64 70 3a 64 69 73 63 6f 76 65 72 22 d a 4d 58 3a 20 31 d a 53 54 3a 20 75 72 6e 3a
64 69 61 6c 2d 6d 75 6c 74 69 73 63 72 65 65 6e 2d 6f 72 67 3a 73 65 72 76 69 63 65 3a 64 69 61 6c 3a 31 d a 55 53 45 5
2 2d 41 47 45 4e 54 3a 20 47 6f 6f 67 6c 65 20 43 68 72 6f 6d 65 2f 31 31 37 2e 30 2e 35 39 33 38 2e 31 35 32 20 57 69 6
e 64 6f 77 73 d a d a
===== Got a packet with length of 2a =====
ff ff ff ff ff ff 0 50 56 c0 0 8 8 6 0 1 8 0 6 4 0 1 0 50 56 c0 0 8 c0 a8 4f 1 0 0 0 0 0 0 c0 a8 4f 2
===== Got a packet with length of d9 =====
1 0 5e 7f ff fa 0 50 56 c0 0 8 8 0 45 0 0 cb 46 67 0 0 1 11 73 17 c0 a8 4f 1 ef ff ff fa f0 46 7 6c 0 b7 7e 8b 4d 2d 53
45 41 52 43 48 20 2a 20 48 54 54 50 2f 31 2e 31 d a 48 4f 53 54 3a 20 32 33 39 2e 32 35 35 2e 32 35 35 2e 32 35 30 3a 31
39 30 30 d a 4d 41 4e 3a 20 22 73 73 64 70 3a 64 69 73 63 6f 76 65 72 22 d a 4d 58 3a 20 31 d a 53 54 3a 20 75 72 6e 3a
64 69 61 6c 2d 6d 75 6c 74 69 73 63 72 65 65 6e 2d 6f 72 67 3a 73 65 72 76 69 63 65 3a 64 69 61 6c 3a 31 d a 55 53 45 5
2 2d 41 47 45 4e 54 3a 20 4d 69 63 72 6f 73 6f 66 74 20 45 64 67 65 2f 31 31 38 2e 30 2e 32 30 38 38 2e 34 36 20 57 69 6
e 64 6f 77 73 d a d a
===== Got a packet with length of d9 =====
1 0 5e 7f ff fa 0 50 56 c0 0 8 8 0 45 0 0 cb 46 68 0 0 1 11 73 16 c0 a8 4f 1 ef ff ff fa f0 4b 7 6c 0 b7 95 ab 4d 2d 53
45 41 52 43 48 20 2a 20 48 54 54 50 2f 31 2e 31 d a 48 4f 53 54 3a 20 32 33 39 2e 32 35 35 2e 32 35 35 2e 32 35 30 3a 31
39 30 30 d a 4d 41 4e 3a 20 22 73 73 64 70 3a 64 69 73 63 6f 76 65 72 22 d a 4d 58 3a 20 31 d a 53 54 3a 20 75 72 6e 3a
64 69 61 6c 2d 6d 75 6c 74 69 73 63 72 65 65 6e 2d 6f 72 67 3a 73 65 72 76 69 63 65 3a 64 69 61 6c 3a 31 d a 55 53 45 5
2 2d 41 47 45 4e 54 3a 20 47 6f 6f 67 6c 65 20 43 68 72 6f 6d 65 2f 31 31 37 2e 30 2e 35 39 33 38 2e 31 35 32 20 57 69 6
e 64 6f 77 73 d a d a
```

(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 源.cpp

运行结果:

```
Microsoft Visual Studio 调试控制台
=====
以太网类型 : 0800
IPv4
Mac源地址:
a8:64:f1:29:1c:92:
Mac目的地址:
00:00:5e:00:01:08:
版本: 4
IP协议首部长度: 20 Bytes
总长度: 67
生存时间:
协议号: 163
协议种类: 首部检验和: 0x0
源地址: 64.17.0.0
目的地址: 10.136.121.254
=====
以太网类型 : 0800
IPv4
Mac源地址:
00:00:5e:00:01:08:
Mac目的地址:
a8:64:f1:29:1c:92:
版本: 4
IP协议首部长度: 20 Bytes
总长度: 61
生存时间:
协议号: 41
协议种类: 首部检验和: 0x4000
源地址: 53.17.217.153
目的地址: 60.29.236.169
```

将捕获分析后的数据包和 wireshark 抓包进行对比,发现不同

77	1.019967	10.136.121.254	150.138.125.32	TCP	54	58374 → 7826 [ACK] Seq=139 Ack=133 Win=512 Len=0
104	1.772663	10.136.121.254	157.148.41.188	TCP	54	61622 → 36688 [FIN, ACK] Seq=655 Ack=41 Win=65496 Len=0
105	1.811640	157.148.41.188	10.136.121.254	TCP	60	36688 → 61622 [FIN, ACK] Seq=41 Ack=656 Win=68096 Len=0
106	1.811742	10.136.121.254	157.148.41.188	TCP	54	61622 → 36688 [ACK] Seq=656 Ack=42 Win=65496 Len=0
133	2.505130	10.136.121.254	47.106.155.75	TCP	55	61591 → 443 [ACK] Seq=1 Ack=1 Win=510 Len=1 [TCP segment of a reassembled PDU]
136	2.638467	47.106.155.75	10.136.121.254	TCP	78	443 → 61591 [ACK] Seq=1 Ack=2 Win=133 Len=0 Tsval=3028716078 Tsecr=583194601 SLE=1 SRE=2
137	2.673507	120.92.85.29	10.136.121.254	TCP	60	7826 → 59908 [ACK] Seq=1 Ack=1 Win=265 Len=0
139	2.673640	10.136.121.254	120.92.85.29	TCP	54	[TCP ACKed unseen segment] 59908 → 7826 [ACK] Seq=1 Ack=2 Win=512 Len=0
176	3.658116	101.91.133.30	10.136.121.254	TLSv1.2	238	Application Data
198	3.712964	10.136.121.254	101.91.133.30	TCP	54	57596 → 443 [ACK] Seq=1 Ack=185 Win=513 Len=0

(5) 编写的程序应结构清晰，具有较好的可读性。

## 四、实验问题及回答：

(1) wireshark 所捕获的数据包为什么和编程捕获的数据包不同。

数据包丢失或重复： 在高负载网络中，数据包可能会丢失或重复。Wireshark 通常会尽力重构数据包以提供尽可能准确的信息。你的自定义程序可能不具备这种重构功能，因此数据包看起来可能与 Wireshark 不同。

时间戳不同： Wireshark 通常会记录数据包捕获的时间戳，而你的自定义程序可能会采用不同的时间戳策略。这可能导致数据包在时间上不同步。

网络接口和捕获设置： 不同的网络接口和捕获设置（例如，混杂模式、数据包截取大小）可能导致捕获到不同的数据包。

(2) MAC 地址在数据包的哪个位置

我们所抓的数据包是 ip 数据包，属于网络层，而 mac 地址处于数据链路层中，因此需要到以太网帧中查看 mac 地址。Ip 数据包格式：

版本（4）	首部长度（4）	优先级与服务类型（8）	总长度（16）	
标识符（16）			标志（3）	段偏移量（13）
TTL（8）	协议号（8）		首部校验和（16）	
源地址（32）				
目标地址（32）				
可选项				
数据				

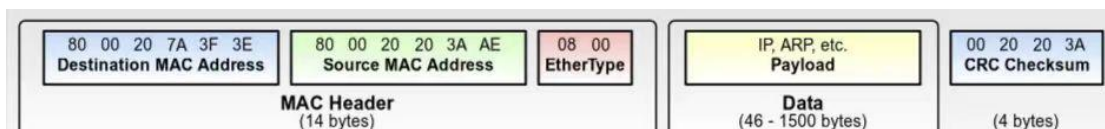
整个以太网帧由头部、数据和校验部分三部分组成，其中，头部由目的 mac 和源 mac 和 2 位的数据类型组成。数据是不定长的，可以由 46-1500 字节组成，这是因为：

以太网要求最小帧长度为 64 字节。这是因为在早期的以太网设计中，以太网帧必须足够长，以确保传输中的冲突检测（CSMA/CD 协议）正常工作。如果帧太短，冲突检测可能会出现问題，导致碰撞的正确检测不可靠。因此数据部分最短为 46 字节。

以太网的最大帧长度通常为 1516 字节。这个限制源于以太网的技术规格，它是在以太网标准 IEEE 802.3 中定义的。较小的帧长度可以更快地传输，而较大的帧长度可以减少协议开销。最大帧长度也有助于减少碰撞的机会。

MTU（最大传输单元）： 以太网帧长度的最大值（1500 字节）通常与网络协议栈中的网络层协议（如 IP）的最大传输单元（MTU）相匹配。这确保了数据包在从以太网传输到较大的网络时不需要进行分段，从而提高了性能。





### (3) 头部校验码的作用及实现

它是 IPv4 头部中的一个字段，用于检测头部信息在传输过程中是否发生了错误或损坏。以下是 IP 头部校验和的实现步骤：

构建 IP 头部：首先，构建完整的 IPv4 头部，包括版本、头部长度、服务类型、总长度、标识、标志位、片偏移、生存时间、协议、源 IP 地址和目标 IP 地址等字段。

将校验和字段设置为 0：在计算校验和之前，将 IP 头部的校验和字段设置为 0。

拆分头部为 16 位字：将 IPv4 头部拆分为多个 16 位字，每两个字节为一组。如果 IPv4 头部的长度不是 16 位字的整数倍，需要在最后一个 16 位字后添加零填充。

累加 16 位字：逐个累加所有 16 位字，包括源 IP 地址、目标 IP 地址和其他所有字段。

溢出位处理：如果在累加过程中发生了溢出，将其回卷到低位。也就是说，将溢出的位加到累加和的末尾。

取反：将累加和的结果按位取反。

将校验和写回头部：将取反后的校验和写回 IPv4 头部的校验和字段。

完成：现在，IP 头部的校验和字段包含了正确的校验和值，可以将 IPv4 数据包发送到网络上。

## 五、实验感悟：

Npcap 的使用：

Npcap 是一个 Windows 平台上的网络数据包捕获库，它提供了一种在程序中捕获数据包的方式。通过 Npcap，可以实现对网络流量的实时监控和分析。

通过编程捕获的数据包可以通过自定义分析逻辑进行处理，例如提取关键信息、识别协议、检测异常行为等。

可改进方向：在 Wireshark 中进行手动分析并记录结果，然后编程捕获相同流量，执行相同的分析逻辑，以进行对比