网络技术与应用 第二次实验

- 姓名：秦泽斌
- 学号：2212005
- 专业：物联网工程

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。

一、NPcap架构

NPcap 是一个为 Windows 平台设计的高性能网络数据包捕获库，基于 WinPcap 项目开发。它提供了低级别的网络接口，允许应用程序直接访问网络数据包。NPcap 通常用于网络分析工具（如 Wireshark），可以捕获网络流量并进行分析。以下是 NPcap 的架构关键部分：

1. 驱动程序层

NPcap 的核心是其内核模式驱动程序。它直接与 Windows 网络堆栈交互，允许从网卡捕获网络数据包。该驱动程序基于 Windows NDIS（网络驱动接口规范）过滤驱动程序模型，这使它能够处理网络接口上的数据流量。

- **NDIS Filter**：NPcap 使用的是 NDIS 6.x 驱动程序模型。它将自己注册为网络适配器上的过滤器，能够捕获进入和离开的数据包，同时不影响正常的网络通信。
- **驱动优化**：NPcap 驱动程序经过优化，支持高吞吐量和低延迟的数据包捕获，适合需要高性能的网络应用。

2. 用户层 API

NPcap 提供了与应用程序交互的用户模式 API。通过这些 API，应用程序可以通过调用函数直接访问网络数据包，常见的库函数如 `pcap_open_live()`、`pcap_next()` 等。

- **Libpcap 兼容性**：NPcap 完全兼容 libpcap 库，用户可以使用与 WinPcap 相同的接口来捕获和发送网络数据包，简化了从其他平台移植代码的工作。
- **高级功能支持**：NPcap 支持 loopback 捕获（用于捕获 localhost 流量）、发送原始数据包、抓取 802.11 无线流量等功能，是 WinPcap 的增强版本。

3. 环回接口

NPcap 增强了对 Windows 环回接口（Loopback）的支持，这是 WinPcap 所缺少的功能。环回接口用于捕获本地主机上的网络流量，比如捕获来自 `127.0.0.1` 或本地应用之间的通信。

4. 安全性

NPcap 增加了对安全性的支持，可以选择安装为“非管理员模式”，允许非管理员用户也能捕获网络数据包。同时，它也支持基于用户的访问控制，防止未经授权的应用程序进行网络嗅探。

5. 性能改进

相比 WinPcap，NPcap 通过对内核驱动的优化以及对现代网络堆栈的支持，提升了数据包捕获和注入的性能。它可以充分利用多核 CPU 和现代网卡硬件加速特性。

6. 与 Wireshark 的集成

NPcap 与 Wireshark 深度集成，作为默认的捕获引擎，为用户提供精确的网络数据包捕获和分析能力。通过其提供的接口，用户可以在 Wireshark 中实时查看和分析捕获的数据流量。

7. 驱动模式

NPcap 支持两种主要的驱动安装模式：

- **混杂模式**：在这种模式下，NPcap 可以捕获到与目标网络接口相关的所有流量，而不仅仅是发往或来自本机的数据包。
- **非混杂模式**：只捕获发往或来自本机的数据包。

总体来说，NPcap 的架构设计主要关注高性能、扩展性以及对现代网络堆栈的支持，为开发人员和网络分析工具提供了强大的网络数据包捕获和发送能力。

二、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. 获取设备列表

使用 `pcap_findalldevs()` 函数可以获取系统中所有可用的网络设备列表。它会返回一个设备列表，包含每个设备的名称、描述以及相关的信息。

```
    pcap_if_t *alldevs;
    char errbuf[PCAP_ERRBUF_SIZE];

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

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

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

- `pcap_if_t` 是设备列表的链表结构，每个节点代表一个网络设备。

- `pcap_findalldevs()` 会返回所有网络设备的列表。
- `pcap_freealldevs()` 用于释放设备列表分配的内存。

2. 打开网卡设备

使用 `pcap_open_live()` 函数可以打开某个特定的网络设备进行数据包捕获。需要指定设备的名称、捕获的最大字节数、是否启用混杂模式、捕获超时时间以及错误缓冲区。

```
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    int snaplen = 65535; // 捕获最大字节数
    int promisc = 1;     // 启用混杂模式
    int timeout_ms = 1000; // 捕获超时，单位毫秒

    // 打开网卡设备
    handle = pcap_open_live("YOUR_DEVICE_NAME", snaplen, promisc, timeout_ms,
errbuf);
    if (handle == NULL) {
        fprintf(stderr, "Couldn't open device: %s\n", errbuf);
        return;
    }

    // 设置过滤规则（可选）
    struct bpf_program fp;
    char filter_exp[] = "tcp"; // 过滤 TCP 包
    if (pcap_compile(handle, &fp, filter_exp, 0, PCAP_NETMASK_UNKNOWN) == -1) {
        fprintf(stderr, "Couldn't parse filter %s: %s\n", filter_exp,
pcap_geterr(handle));
        return;
    }

    if (pcap_setfilter(handle, &fp) == -1) {
        fprintf(stderr, "Couldn't install filter %s: %s\n", filter_exp,
pcap_geterr(handle));
        return;
    }

    // 释放过滤器资源
    pcap_freecode(&fp);
```

- `pcap_open_live()` 用于打开设备，其中 `snaplen` 是捕获的数据包最大长度，`promisc` 表示是否启用混杂模式，`timeout_ms` 是捕获的超时时间。
- `pcap_compile()` 和 `pcap_setfilter()` 可以设置过滤规则，如只捕获 TCP 数据包。

3. 数据包捕获方法

使用 `pcap_loop()` 或 `pcap_next_ex()` 可以捕获数据包，`pcap_loop()` 是一个循环捕获函数，`pcap_next_ex()` 用于逐个捕获数据包。

- **使用 `pcap_loop()` 捕获多个数据包**

```
void packet_handler(u_char *user, const struct pcap_pkthdr *header, const
u_char *pkt_data) {
    printf("Captured a packet with length of [%d]\n", header->len);
    // 可以在这里对数据包进行处理
}

// 捕获 10 个数据包
pcap_loop(handle, 10, packet_handler, NULL);
```

  - `pcap_loop()` 第一个参数是 `pcap_t` 句柄，第二个参数是要捕获的数据包数量，第三个参数是处理函数，第四个参数是传递给处理函数的用户数据。

- **使用 `pcap_next_ex()` 捕获单个数据包**

```
struct pcap_pkthdr *header;
const u_char *pkt_data;
int res = pcap_next_ex(handle, &header, &pkt_data);
if (res == 1) {
    printf("Captured a packet with length of [%d]\n", header->len);
    // 在这里处理数据包
} else if (res == 0) {
    printf("Timeout occurred while capturing\n");
} else {
    printf("Error occurred: %s\n", pcap_geterr(handle));
}
```

  - `pcap_next_ex()` 用于捕获单个数据包，返回值为 1 表示成功捕获，0 表示超时，-1 表示发生错误。

三、NPcap编程

1. 头文件导入

```
#include<iostream>
#include<winSock2.h>
#include<iomanip>
#include<cstring>
#include<format>
#include<pcap.h>
```

- `iostream`: 用于标准输入输出。
- `winSock2.h`: 用于 Windows 套接字编程，定义了一些网络相关的类型和函数。
- `iomanip`: 用于格式化输出。
- `cstring`: 用于字符串操作。
- `format`: C++20 的新特性，用于格式化输出。
- `pcap.h`: WinPcap/NPcap 库的头文件，包含用于网络数据包捕获的函数和结构。

2. 库链接

```
#pragma comment(lib,"wpcap.lib")
#pragma comment(lib,"Packet.lib")
#pragma comment(lib,"WS2_32.lib")
```

- `wpcap.lib`: 用于链接 WinPcap/NPcap 库。
- `Packet.lib`: 用于链接 Packet.dll，提供底层的网络接口操作。
- `WS2_32.lib`: Windows 套接字库，用于网络通信。

3. 帧头和 IP 头结构体定义

使用 `#pragma pack(1)` 来确保结构体在内存中按 1 字节对齐，这对于处理网络数据包这种按字节序列传输的数据尤为重要。

```
typedef struct FrameHeader_t {
   BYTE DesMAC[6];    // 目标MAC地址
   BYTE SrcMAC[6];    // 源MAC地址
   WORD FrameType;    // 帧类型
} FrameHeader_t;

typedef struct IPHeader_t {
   BYTE Ver_HLen;     // 版本和首部长度
   BYTE TOS;          // 服务类型
   WORD TotalLen;     // 总长度
   WORD ID;           // 标识符
   WORD Flag_Segment;// 标识和分段偏移
   BYTE TTL;          // 生存时间
   BYTE Protocol;     // 协议类型
   WORD Checksum;     // 首部校验和
   ULONG SrcIP;       // 源IP地址
   ULONG DstIP;       // 目的IP地址
} IPHeader_t;

typedef struct Data_t {
   FrameHeader_t FrameHeader;
   IPHeader_t IPHeader;
} Data_t;
```

- `FrameHeader_t`: 表示以太网帧的头部，包含目标 MAC 地址、源 MAC 地址和帧类型。
- `IPHeader_t`: 表示 IP 数据包的头部，包含 IP 版本、总长度、源 IP、目的 IP 等信息。
- `Data_t`: 包含了一个以太网帧头和一个 IP 头，用于描述捕获到的 IP 数据包的结构。

4. 主函数逻辑

```
int main() {
    Data_t* IPPacket;              // 用于保存捕获到的数据包
    pcap_if_t* alldevs;            // 用于存储所有网络设备
    pcap_if_t* d;                  // 指向选择的网络设备
    int inum;                      // 用户选择的设备号
    int i = 0;                     // 计数器
    pcap_t* adhandle;              // pcap 捕获句柄
    struct pcap_pkthdr* header;    // 保存捕获的数据包头部信息
    const u_char* pkt_data;        // 捕获到的数据包内容
    char errbuf[PCAP_ERRBUF_SIZE];// 错误缓冲区
```

- `Data_t* IPPacket`: 指向捕获到的数据包。
- `pcap_if_t* alldevs`: 用于存储系统中所有网络设备的链表。
- `pcap_if_t* d`: 指向用户选择的网络设备。
- `pcap_t* adhandle`: pcap 的捕获句柄，用于打开并捕获数据包。

5. 获取设备列表并选择设备

```
if (pcap_findalldevs(&alldevs, errbuf) == -1) {
    cout << "error in findalldevs: " << errbuf << endl;
    return 0;
}
```

使用 `pcap_findalldevs()` 获取所有可用的网络设备，并将它们存储在 `alldevs` 中。如果函数返回 -1，表示获取设备列表失败，输出错误信息并退出。

```
for (d = alldevs; d; d = d->next) {
    cout << ++i << ". " << d->name << endl;
    if (d->description)
        cout << d->description << endl;
}
```

遍历设备列表，输出每个设备的名称和描述。

```
cout << "选择目标设备: " << endl;
cin >> inum;
for (d = alldevs, i = 0; i < inum - 1; d = d->next, i++);
```

用户输入一个设备编号，程序将 `d` 指向相应的设备。

6. 打开设备进行捕获

```
adhandle = pcap_open_live(d->name, 65536, 1, 1000, errbuf);
```

使用 `pcap_open_live()` 打开设备进行实时数据包捕获：

- `d->name`: 要打开的设备名。
- `65536`: 捕获的最大字节数。

- `1`：开启混杂模式，捕获所有流经此网卡的数据包。
- `1000`：捕获超时为 1000 毫秒。
- `errbuf`：错误缓冲区，用于保存出错信息。

7. 循环捕获数据包

```
int res;
while ((res = pcap_next_ex(adhandle, &header, &pkt_data)) >= 0) {
   if (res == 0)
      continue;
```

使用 `pcap_next_ex()` 循环捕获数据包：

- `pcap_next_ex()` 返回 1 表示成功捕获到数据包，0 表示超时，负值表示出错。

8. 解析和输出数据包内容

```
IPPacket = (Data_t*)pkt_data;
BYTE* desMac = IPPacket->FrameHeader.DesMAC;
BYTE* srcMac = IPPacket->FrameHeader.SrcMAC;
```

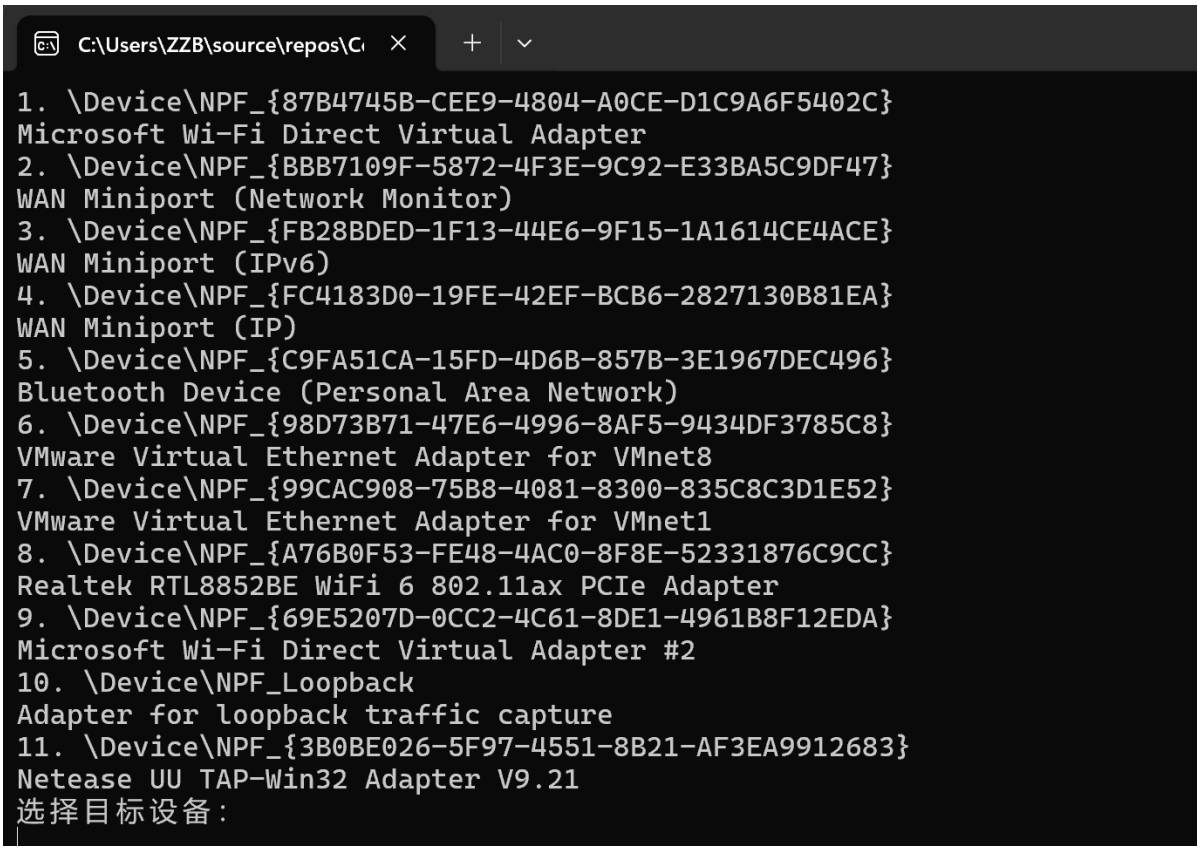
将捕获到的数据包转化为 `Data_t` 结构体，解析出源 MAC 地址和目的 MAC 地址。

```
string DesMAC = format("目的MAC地址: {:02x}:{:02x}:{:02x}:{:02x}:{:02x}:{:02x}",
desMac[0], desMac[1], desMac[2], desMac[3], desMac[4], desMac[5]);
string SrcMAC = format("源MAC地址: {:02x}:{:02x}:{:02x}:{:02x}:{:02x}:{:02x}",
srcMac[0], srcMac[1], srcMac[2], srcMac[3], srcMac[4], srcMac[5]);
cout << SrcMAC << "\t" << DesMAC
     << "\t类型: 0x" << hex << ntohs(IPPacket->FrameHeader.FrameType)
     << "\t总字段长度: " << dec << ntohs(IPPacket->IPHeader.TotalLen)
     << "\t捕获包长度: " << header->caplen << endl;
```

使用 `format()` 函数输出源 MAC 地址、目的 MAC 地址、帧类型、IP 数据包的总长度以及捕获的数据包长度。

四、程序效果展示

选择网卡界面如下：



为了方便演示，选择8.WIFI网卡，捕获情况如下：