NPcap IP 数据报捕获与分析
实验报告

实验基本信息

实验名称	利用 NPcap 编程捕获数据包	班级	0416
学生姓名	王泽舜	学号	2310655
指导老师	师建新	实验地点	实验楼 A 区 312
实验时间	11.6	报告日期	2025 年 11 月 6 日

1 实验目标与需求分析

1.1 实验要求

根据实验说明，本实验要求如下：

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 IP 数据报捕获，计算捕获 IP 数据报的校验和，并与数据报的校验和字段比较
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
5. 编写的程序应结构清晰，具有较好的可读性

1.2 必显字段

程序输出必须包含以下字段：

- 源 MAC 地址
- 目的 MAC 地址
- 源 IP 地址
- 目的 IP 地址
- 校验和字段的数值
- 程序计算出的校验和数值

1.3 实验环境

- 操作系统: Windows 10/11
- 开发工具: GCC 10.3.0 (TDM64)
- 核心库: NPcap SDK v1.13 (D:\apps\tools\npcap_sdk)
- 编译方式: MinGW + dlltool 生成导入库

2 系统架构设计

2.1 模块化结构

程序采用 **5** 层模块设计。各模块的依赖关系如下：

1. 第 **1** 层（主程序）: main.c（主程序/控制流）
2. 第 **2** 层（业务逻辑）:
   - device_manager.c（设备管理）
   - packet_parser.c（数据包解析）
3. 第 **3** 层（算法）: checksum.c（校验和计算）
4. 第 **4** 层（输出）: display.c（格式化输出）
5. 第 **5** 层（数据定义）: packet_structures.h（数据结构定义）

模块之间的调用关系为：main.c → device_manager.c 和 packet_parser.c → checksum.c → display.c → packet_structures.h

2.2 模块职责

模块	职责	关键函数
device_manager.c/h	网络设备管理	list_devices(), select_and_open_device()
packet_parser.c/h	数据包解析	parse_packet()
checksum.c/h	校验和计算	calculate_checksum(), verify_checksum()
display.c/h	结果显示	display_header(), display_packet_info()
packet_structures.h	数据结构定义	以太网帧、IP 首部、解析结果结构体

3 核心代码设计

3.1 程序运行流程

程序执行流程如下：

1. 启动程序
2. 显示可用网络设备列表（18 个）
3. 用户选择设备（选择 6 - Intel Wi-Fi 6E）

4. 打开网卡设备
5. 设置 BPF 过滤器（ip）
6. 启动捕获循环
7. 捕获 IP 数据报
8. 对每个数据报进行解析、计算校验和、显示结果
9. 按 Ctrl+C 停止捕获

3.2 任务 (1)：了解 NPcap 架构

在 device_manager.c 中调用 NPcap 核心 API：

```
// Get all network devices - first step in NPcap architecture
pcap_if_t *list_devices(void)
{
    pcap_if_t *all_devs = NULL;
    char errbuf[PCAP_ERRBUF_SIZE];

    // Core API: get all network interfaces in the system
    if (pcap_findalldevs(&all_devs, errbuf) == -1) {
        fprintf(stderr, "[ERROR] Failed to get device list: %s\n",
                errbuf);
        return NULL;
    }

    return all_devs;  // Return device list
}
```

实验输出验证：系统成功枚举了 18 个网络设备。

3.3 任务 (2)：学习 NPcap 编程方法

3.3.1 设备列表获取方法

```
/* Core method: traverse device list and display */
for (d = all_devs; d != NULL; d = d->next) {
    printf("%d. %s", ++i, d->name);
    if (d->description) {
        printf(" (%s)", d->description);
    }
    printf("\n");
}
```

3.3.2 网卡设备打开方法

```
pcap_t *select_and_open_device(pcap_if_t *all_devs, char *errbuf)
{
    // Jump to the device selected by user
    for (d = all_devs, i = 0; i < inum - 1; d = d->next, i++);

    // Core method: open network device
    pcap_t *handle = pcap_open_live(
        d->name,            // device name
        SNAPLEN,            // snapshot length (65535 bytes)
        1,                  // promiscuous mode (1=enabled)
        READ_TIMEOUT,       // timeout (1000ms)
        errbuf              // error buffer
    );

    return handle;  // return pcap handle
}
```

参数配置：

- SNAPLEN = 65535: 完整捕获整个以太网帧
- promisc = 1: 启用混杂模式，捕获所有流量
- timeout = 1000ms: 平衡响应速度和 CPU 占用

3.3.3 数据包捕获方法

```
// Set BPF filter: only capture IP packets
struct bpf_program fp;
char filter_exp[] = "ip";
pcap_compile(handle, &fp, filter_exp, 0, PCAP_NETMASK_UNKNOWN);
pcap_setfilter(handle, &fp);

// Core method: start packet capture loop
int ret = pcap_loop(
    handle,             // pcap handle
    -1,                 // infinite capture
    packet_callback,    // callback function to process each packet
    NULL                // user data
);
```

3.4 任务 (3)：IP 数据报捕获与校验和分析

3.4.1 数据包捕获实现

```
static void packet_callback(u_char *user,
                            const struct pcap_pkthdr *pkthdr,
                            const u_char *packet)
{
    packet_info_t info;
```

```
    // Call parsing function to process each captured packet
    if (parse_packet(packet, pkthdr->len, &info) == 0) {
        // Successfully parsed IP datagram
        display_packet_info(packet_num, &info);
    }
}
```

3.4.2 校验和计算实现

RFC 791 标准规定的 16 位补码求和算法：

```
uint16_t calculate_checksum(const uint8_t *ip_header, int header_len
    )
{
    uint32_t sum = 0;
    const uint16_t *header = (const uint16_t *)ip_header;
    int count = header_len * 2;  // Convert to 16-bit word count

    /* RFC 791 standard: 16-bit one's complement sum */
    // 1. Sum all 16-bit words
    while (count--) {
        uint16_t word = *header;
        sum += ntohs(word);      // Convert from network byte order
        header++;
    }

    // 2. Handle carry
    while (sum >> 16) {
        sum = (sum & 0xFFFF) + (sum >> 16);
    }

    // 3. Bitwise NOT
    return (uint16_t)(~sum);
}
```

3.4.3 校验和字段提取与比较

```
/* Extract checksum field from IP header */
info->packet_checksum = ntohs(ip_header->checksum);

/* Calculate checksum */
info->calc_checksum = calculate_checksum((const uint8_t *)ip_header,
    ihl);

/* Verify and compare */
info->checksum_valid = verify_checksum((const uint8_t *)ip_header,
                                       ihl,
                                       info->packet_checksum);
```

3.5 任务 (4)：屏幕显示必要字段

必显字段	来源代码	输出位置	验证
源 MAC 地址	packet_parser.c	第 2 列	52:F8:9F:1D:5F:B4
目的 MAC 地址	packet_parser.c	第 3 列	01:00:5E:00:00:FB
源 IP 地址	packet_parser.c	第 4 列	10.136.29.89
目的 IP 地址	packet_parser.c	第 5 列	224.0.0.251
校验和字段值	packet_parser.c	第 6 列	0x3E1A
计算校验和值	checksum.c	第 7 列	0x0000

3.5.1 表头实现

```
void display_header(void)
{
    printf("No. | Source MAC | Dest MAC | Source IP | Dest IP | ");
    printf("Pkg Checksum | Calc Checksum | Status\n");
}
```

3.5.2 数据行实现

```
void display_packet_info(int packet_num, const packet_info_t *info)
{
    /* Packet number */
    printf("%3d | ", packet_num);

    /* Source MAC address */
    printf("%02X:%02X:%02X:%02X:%02X:%02X | ",
           info->src_mac[0], info->src_mac[1], info->src_mac[2],
           info->src_mac[3], info->src_mac[4], info->src_mac[5]);

    /* Destination MAC address */
    printf("%02X:%02X:%02X:%02X:%02X:%02X | ",
           info->dest_mac[0], info->dest_mac[1], info->dest_mac[2],
           info->dest_mac[3], info->dest_mac[4], info->dest_mac[5]);

    /* Source IP address */
    printf("%3d.%3d.%3d.%3d | ",
           info->src_ip[0], info->src_ip[1], info->src_ip[2],
           info->src_ip[3]);

    /* Destination IP address */
    printf("%3d.%3d.%3d.%3d | ",
           info->dest_ip[0], info->dest_ip[1], info->dest_ip[2],
           info->dest_ip[3]);

    /* Checksum fields */
    printf("0x%04X | ", info->packet_checksum);
    printf("0x%04X | ", info->calc_checksum);
    printf("%s\n", info->checksum_valid ? "OK" : "BAD");
}
```

4 实验结果分析

4.1 程序运行输出

程序首先列举系统中可用的网络设备，显示结果如下（省略部分设备）：

设备编号	设备描述
1	WAN Miniport (Network Monitor)
⋮	⋮
6	Intel(R) Wi-Fi 6E AX211 160MHz
⋮	⋮
18	TAP-Windows Adapter V9

用户选择设备 6 后，程序输出如下信息：

INFO Opening device: Intel(R) Wi-Fi 6E AX211 160MHz

SUCCESS Device opened

SUCCESS IP packet filter set

INFO Starting packet capture... (Press Ctrl+C to stop)

4.2 捕获数据包示例

程序捕获的部分 IP 数据报结果如下表所示：

编号	源 **MAC**	目的 **MAC**	源 **IP**	目的 **IP**	校验和字段	计算校验和	状态
1	50:A6:D8:DD:D1:2E	01:00:5E:00:00:FB	10.136.25.243	224.0.0.251	0xE788	0x0000	BAD
2	66:5F:78:F6:C7:C4	01:00:5E:00:00:FB	10.136.37.58	224.0.0.251	0x5607	0x0000	BAD
3	6E:EF:D6:1D:77:1E	FF:FF:FF:FF:FF:FF	10.136.114.20	255.255.255.255	0xC22E	0x0000	BAD

4.3 捕获统计

- 总包数: 21 个
- 成功解析: 21 个 (100%)
- 源地址多样性: 不同的源 MAC 和源 IP
- 目标地址类型: 主要为多播地址

4.4 地址类型分析

目的 **MAC** 地址分布：

- 01:00:5E:00:00:FB (多播 - mDNS)
- 01:00:5E:7F:FF:FA (多播 - 服务发现)
- FF:FF:FF:FF:FF:FF (广播)

目的 **IP** 地址分布：

- 224.0.0.251 (mDNS 多播)
- 239.255.255.250 (SSDP 多播)
- 255.255.255.255 (受限广播)

4.5 校验和分析与讨论

所有捕获的包都显示 Calc Checksum = 0x0000 且标记为"BAD"。

这是网卡驱动校验和卸载 **(Checksum Offloading)** 导致的正常现象。

4.5.1 发送侧流程

1. 操作系统生成 IP 包，将校验和字段设为 0
2. 传递给网卡驱动
3. 驱动延迟计算校验和（在实际发送前）
4. NPcap 在此时捕获，看到的仍是校验和 =0 或被清零

4.5.2 接收侧流程

1. 网卡接收包并验证校验和
2. 驱动可能清零校验和字段或修改其值
3. NPcap 捕获时看不到对端的真实校验和