

实验2：IP数据报捕获与分析（利用NPcap编程捕获数据包）

郭坤昌 2012522 计算机科学与技术

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IP数据报捕获，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。

实验过程

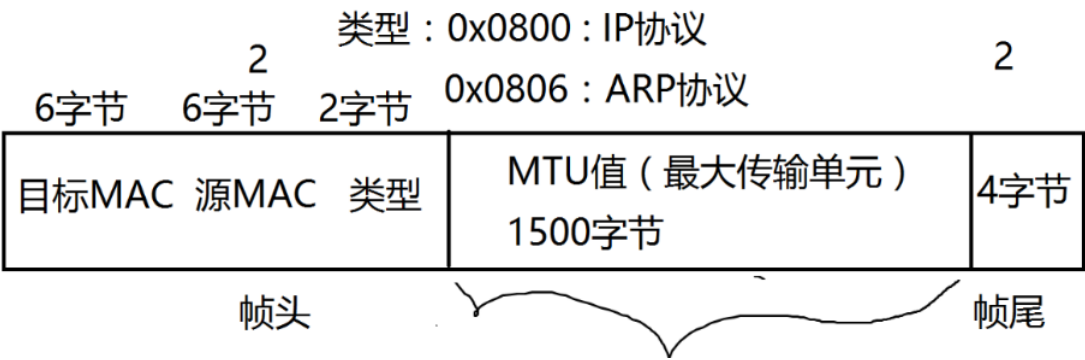
声明和定义

变量声明：

```
1  pcap_if_t* alldevs; //网卡列表
2  pcap_if_t* d;      //设备指针
3  pcap_addr_t* a;    //地址
4  pcap_t* adhandle;   //适配器句柄
5  struct pcap_pkthdr* header; //数据包头
6  struct tm ltime;    // 本地时间
7  time_t local_tv_sec; // 本地时间
8  const u_char* pkt_data; // 数据包数据
9  char errbuf[PCAP_ERRBUF_SIZE]; // 错误信息
10 char timestr[16];   // 时间字符串
11 int cnt = 0;        // 网卡计数
12 int res = 0;        // 捕获数据包计数
13 int i;              // 作为输入，选中的网卡序号 [1-cnt]
```

数据包头结构体：

- 帧结构



- IP数据包结构

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				

20字节

https://blog.csdn.net/weixin_14799090

20
字
节

根据帧结构和IP数据包结构，如下定义数据包头结构体：

数据包中的数据是连续的，因此需要进行字节对齐，使用`#pragma pack(1)`声明，定义完后恢复默认对齐方式

```

1  #pragma pack(1) // 进入字节对齐模式
2  typedef struct FrameHeader
3  {
4      BYTE dstMac[6]; // 目的MAC地址
5      BYTE srcMac[6]; // 源MAC地址
6      WORD type;      // 类型
7  };
8  typedef struct IPHeader
9  {
10     BYTE verLen;
11     BYTE tos;
12     WORD totalLen; // IP数据包总长度
13     WORD id; // 标识
14     WORD flagOffset;
15     BYTE ttl;
16     BYTE protocol;
17     WORD checksum; // 校验和
18     DWORD srcIP; // 目的IP
19     DWORD dstIP; // 源IP
20 };
21 typedef struct Data
22 {
23     struct FrameHeader fh;
24     struct IPHeader ih;
25 };
26 #pragma pack() // 恢复默认对齐方式

```

函数声明

```

1  /* 任意进制转十进制 */
2  int Atoi(string s, int radix);
3
4  /* 计算校验和 */
5  USHORT CheckSum(USHORT* buffer, int size);
6
7  /*解析数据包的帧首部和IP首部，得到类型、标识、序列号、校验和等*/
8  void parse(const u_char* pkt_data);

```

获取设备列表

使用pcap_findalldevs_ex函数获取设备列表。函数原型和参数含义为：

```

1  int pcap_findalldevs_ex(
2      char *source, // 获取列表的来源，希望获取本机网络接口时使用
        PCAP_SRC_IF_STRING
3      struct pcap_rmtauth *auth, // 获取远程设备网络接口列表时，如果需要
        认证则需要使用，此实验中为NULL即可
4      pcap_if_t **alldevs, // 函数返回后指向设备列表的第一个元素（类型
        为pcap_if_t）
5      char *errbuf // 错误信息返回缓冲区
6  );
7  // 调用错误返回-1，成功返回0

```

struct pcap_if_t的定义为：

```

1  struct pcap_if {
2      struct pcap_if *next;
3      char *name; // name to hand to "pcap_open_live()" */
4      char *description; // textual description of interface, or
        NULL */
5      struct pcap_addr *addresses;
6      bpf_u_int32 flags; // PCAP_IF_ interface flags */
7  };

```

获取设备列表：

```

1  if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf)
    == -1)
2  {
3      fprintf(stderr, "Error in pcap_findalldevs_ex: %s", errbuf);
4      exit(1);
5  }

```

显示结果为：

```

All device are listed here.
1  Name: rpcap://Device\NPF_{C292093B-F0CB-4A16-AD0C-EF9F2FF4F65E} Description: Network adapter 'WAN Miniport (Network Monitor)' on local host
2  Name: rpcap://Device\NPF_{16C4AB3F-4AE9-4B59-86E8-C5283D916305} Description: Network adapter 'WAN Miniport (IPv6)' on local host
3  Name: rpcap://Device\NPF_{12DEAFDF-4CA8-490C-ABC1-9B34EFA6AD12} Description: Network adapter 'WAN Miniport (IP)' on local host
4  Name: rpcap://Device\NPF_{1EDF97E5-D92D-484D-95B9-98E80196FC44} Description: Network adapter 'Bluetooth Device (Personal Area Network)' on local host
5  Name: rpcap://Device\NPF_{D90A1E99-1885-49CF-9173-35663F6E110F} Description: Network adapter 'Realtek 802.11n Wireless LAN 802.11ac PCI-E NIC' on local host
6  Name: rpcap://Device\NPF_{D83B3DA1-C8E4-4C7D-8822-41A2BAD8ABE6} Description: Network adapter 'VMware Virtual Ethernet Adapter for VMnet8' on local host
7  Name: rpcap://Device\NPF_{FF667A75-F495-4B59-B31F-EA8E1E172360} Description: Network adapter 'VMware Virtual Ethernet Adapter for VMnet1' on local host
8  Name: rpcap://Device\NPF_{8FCD9D89-B861-450E-8F50-B8F3EE98140D} Description: Network adapter 'Microsoft Wi-Fi Direct Virtual Adapter #2' on local host
9  Name: rpcap://Device\NPF_{8E778175-4F5C-411D-B110-1EE391C8D7EB} Description: Network adapter 'Microsoft Wi-Fi Direct Virtual Adapter' on local host
10 Name: rpcap://Device\NPF_Loopback Description: Network adapter 'Adapter for loopback traffic capture' on local host

```

打开对应网卡

打开网络接口可以使用pcap_open函数，原型为

```

1 pcap_t* pcap_open(
2     const char* source, // 打开网卡设备的名字
3     int snaplen, // 获取网络数据包的最大长度，最大为65536
4     int flags, // 打开网卡的模式，PCAP_OPENFLAG_PROMISCUOUS表示混杂模
        式，捕获所有流经该网卡的数据包
5     int read_timeout, // 捕获数据包的等待时间，若使用pcap_next_ex()没有
        捕获到数据包，则返回0
6     struct pcap_rmtauth* auth,
7     char* errbuf
8 );
9 // 调用出错时返回NULL

```

选择本机网卡，设置获取网络数据包的最大长度，等待时间为1s，以混合模式打开网卡，获得句柄管理打开的网卡

```

1 if ((adhandle = pcap_open(d->name, 65536, PCAP_OPENFLAG_PROMISCUOUS,
2     1000, NULL, errbuf)) == NULL)
3 {
4     fprintf(stderr, "Unable to open the adapter. %s is not supported
        by WinPcap\n", d->name);
5     /* 释放网卡列表 */
6     pcap_freealldevs(alldevs);
7     return -1;
8 }

```

捕获并解析数据包

使用函数pcap_next_ex捕获数据包。原型为：

```

1 int pcap_next_ex(
2     pcap_t* p, // 已打开的设备
3     struct pcap_pkthdr** pkt_header, // 报文头
4     const u_char** pkt_data // 报文内容
5 );
6 // 成功返回1，超时返回0，发生错误返回-1，获取到离线记录的最后一个报文返回-2

```

其中，报文头对应的结构体pcap_pkthdr

```

1 struct pcap_pkthdr
2 {
3     struct timeval ts; //ts是一个结构struct timeval，它有两个部分，
        第一部分是1900开始以来的秒数，第二部分是当前秒之后的毫秒数
4     bpf_u_int32 caplen; //表示抓到的数据长度
5     bpf_u_int32 len; //表示数据包的实际长度
6 }
7

```

捕获数据包，并打印时间、报文长度，并进一步解析帧的类型、IP数据包中的信息。

```

1 while ((res = pcap_next_ex(adhandle, &header, &pkt_data)) >= 0)
2 {
3     if (res == 0)
4         continue; /* 超时继续 */

```

```

5
6      /* 打印时间 */
7      local_tv_sec = header->ts.tv_sec;
8      localtime_s(&lttime, &local_tv_sec);
9      strftime(timestr, sizeof timestr, "%H:%M:%S", &lttime);
10     printf("[%s,%.6ld]\t", timestr, header->ts.tv_usec);
11
12     /* 打印数据包长度 */
13     printf("PKT LENGTH: %d\n", Atoi(to_string(header->len),16));
14
15     /* 解析数据包并打印相关信息 */
16     parse(pkt_data);
17     printf("\n-----\n");
18 }
19

```

分析过程在于根据数据包首部各部分定义，获取所需的信息，具体解析过程为：

```

1 void parse(const u_char* pkt_data)
2 {
3     struct Data* data = (struct Data*)pkt_data;
4     /* 打印序列号 */
5     printf("ID: % 04x\t\t", ntohs(data->ih.id));
6
7     /* 打印IP数据包长度 */
8     printf("IP PKT LENGTH: %d\n", Atoi(to_string(ntohs(data-
9 >ih.totalLen)), 16));
10
11     /* 打印目的Mac地址和源Mac地址 */
12     printf("DEST Mac: %02x:%02x:%02x:%02x:%02x:%02x\n SRC Mac:
13 %02x:%02x:%02x:%02x:%02x:%02x\n", data->fh.dstMac[0], data-
14 >fh.dstMac[1], data->fh.dstMac[2], data->fh.dstMac[3], data-
15 >fh.dstMac[4], data->fh.dstMac[5], data->fh.srcMac[0], data-
16 >fh.srcMac[1], data->fh.srcMac[2], data->fh.srcMac[3], data-
17 >fh.srcMac[4], data->fh.srcMac[5]);
18
19     /* 打印帧类型 */
20     printf("FRAME TYPE: %04x,", ntohs(data->fh.type));
21     switch (ntohs(data->fh.type))
22     {
23     case 0x0800:
24         printf("IPV4\n");
25         /* 打印校验和 */
26         printf("ORI CKECKSUM: %04x\n", ntohs(data->ih.checksum));
27         /* 打印计算得到的校验和 */
28         data->ih.checksum = 0;
29         printf("CAL CHECKSUM: %04x", ntohs(CheckSum((USHORT*)&data-
30 >ih, 20)));
31         break;
32     case 0x86dd:
33         printf("IPV6");
34         break;
35     case 0x0806:
36

```

```

29     printf("ARP ");
30     break;
31     case 0x8035:
32         printf("RARP");
33         break;
34     default:
35         printf("OTHER TYPE");
36         break;
37 }
38 }

```

计算校验和，需要首先将IP数据包首部中的校验和置0，计算方法为：

```

1  USHORT CheckSum(USHORT* buffer, int size)
2  {
3      unsigned long cksum = 0;
4      while (size > 1)
5      {
6          cksum += *buffer++;
7          size -= sizeof(USHORT);
8      }
9      if (size)
10     {
11         cksum += *(UCHAR*)buffer;
12     }
13     cksum = (cksum >> 16) + (cksum & 0xffff); //将高16bit与低16bit
相加
14     cksum += (cksum >> 16);
15     return (USHORT) (~cksum);
16 }

```

需要注意的是，数据在网络是大端表示，而在本地计算机（x86架构）中是小端表示，因此需要使用ntohs函数将网络中的数据以小端表示。

显示结果为：

```

[08:36:50, 999007] PKT LENGTH: 96 → 抓包时间
ID: 978e IP PKT LENGTH: 64 → IP数据包标识
DEST Mac: 28:39:26:e2:7d:6b
SRC Mac: 00:00:5e:00:01:0d
FRAME TYPE: 0800, IPV4 → 帧标识的类型
ORI CKECKSUM: 508e
CAL CHECKSUM: 508e

```

```

[08:36:50, 999007] PKT LENGTH: 918
ID: 978f IP PKT LENGTH: 898
DEST Mac: 28:39:26:e2:7d:6b
SRC Mac: 10:19:65:14:58:a0 → IP数据包的校验和
FRAME TYPE: 0800, IPV4
ORI CKECKSUM: 4f37 → 计算得到校验和
CAL CHECKSUM: 4f37

```

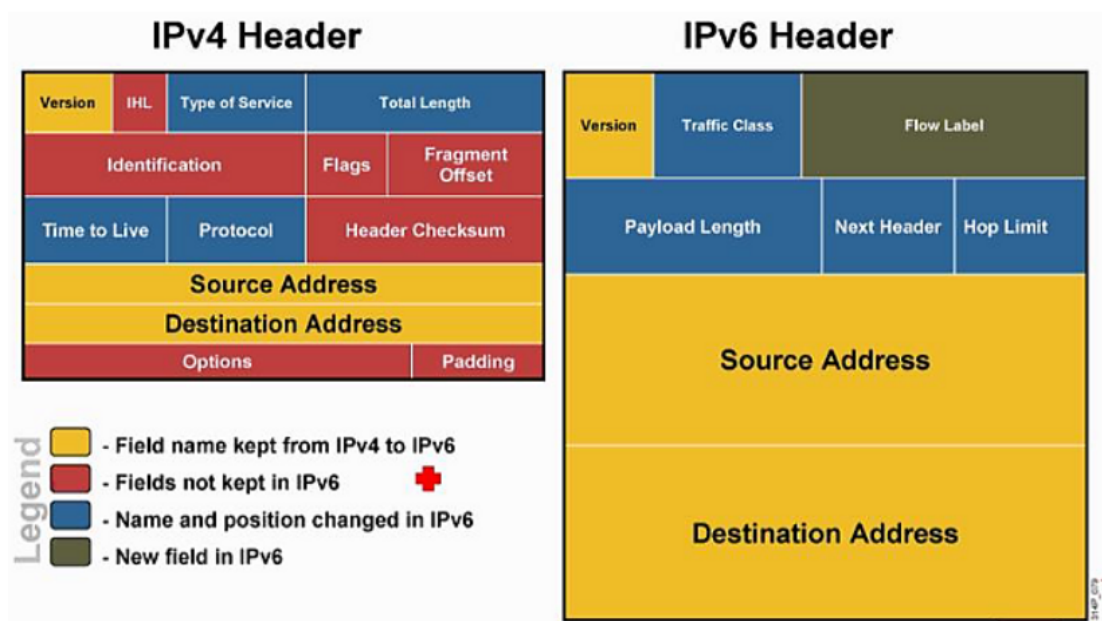
```

[08:36:50, 999007] PKT LENGTH: 291 → 数据包长度
ID: 9790 IP PKT LENGTH: 265 → IP数据包长度
DEST Mac: 28:39:26:e2:7d:6b → 目的Mac地址
SRC Mac: 10:19:65:14:58:a0 → 源Mac地址
FRAME TYPE: 0800, IPV4
ORI CKECKSUM: 5047
CAL CHECKSUM: 5047

```

实验中遇到的问题

- 数据包头的格式与报文的协议类型有关。



因此实验中预设定的包头结构体只对IPv4数据包进行解析是有效的。