# 第一题

首先写代码把src\_ip和dst\_ip为空的数据包打印出来

```
Packet that the value of src_ip and dst_ip are blank:
protocol: ARP info: Who has 192.168.1.23? Tell 192.168.1.2
protocol: ARP info: Who has 192.168.1.1? Tell 192.168.1.2
protocol: 0xfffa info: Ethernet II
protocol: ARP info: Who has 192.168.1.15? Tell 192.168.1.1
protocol: ARP info: 192.168.1.15 is at 54:14:f3:c8:73:7d
protocol: DNS info: Standard query 0x9d95 A articles.zsxq.com
protocol: DNS info: Standard query 0xbfeb AAAA articles.zsxq.com
protocol: DNS info: Standard query 0x9d95 A articles.zsxq.com
protocol: DNS info: Standard query response 0x9d95 A articles.zsxq.com A 117.50.36.86 A 117.50.10.178
protocol: DNS info: Standard query response 0xbfeb AAAA articles.zsxq.com SOA ns3.dnsv4.com
protocol: 0xfffa info: Ethernet II
```

如图所示,有三种协议源IP和目的IP为空,ARP、DNS、0xFFFA,下面分别解释原因

首先,众所周知,目前计算机网络实际广泛采用的使用四层模型,而七层模型和五层模型一个是理论上的,一个是教学用的,下图是四层模型以及下面分析所用的协议所在位置

应用层	DNS
运输层	TCP UDP
网际层	IP ARP
网络接口层	0xFFFA

### ARP协议

ARP协议位于网际层,和IP在同一层,理论上是可以使用IP协议的,我们看一下ARP协议的结构(下图),可以看到也是存在源IP和目的IP

字段名称	长度 (字节)	描述
硬件类型 (HTYPE)	2	表示硬件地址类型,例如以太网为 0x0001 。
协议类型 (PTYPE)	2	表示协议地址类型,例如 IPv4 为 0x0800 。
硬件地址长度 (HLEN)	1	硬件地址的长度,例如以太网为 6。
协议地址长度 (PLEN)	1	协议地址的长度,例如 IPv4 为 4 。
操作码 (Opcode)	2	表示 ARP 请求或响应, 1 表示请求, 2 表示响应
源硬件地址 (SHA)	HLEN	发送方的硬件地址(MAC地址)。
源协议地址 (SPA)	PLEN	发送方的协议地址(IP 地址)。
目标硬件地址 (THA)	HLEN	目标硬件地址(在请求中为 0 , 响应中为解析到的
目标协议地址 (TPA)	PLEN	目标的协议地址 (目标 IP 地址) 。
填充	可变	如果硬件地址或协议地址长度不是2字节的整数倍,可

那为什么我们的结果中arp协议没有源IP和目的IP呢?

这里我们回头看下使用的过滤语法: [ip.src], 推测这个语法是对基于IP协议(包括本身)的协议进行过滤, 而ARP协议和IP协议是并列关系, ARP协议只是使用到了IP协议

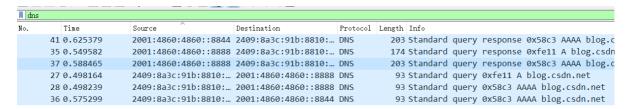
换一个语法进行验证: arp.src.proto\_ipv4,根据下图可以证明推测正确

arp. src. proto_ipv4					
No.	Tine	Source	Destination	Protocol	Length Info
	205 8.020989	Shenzhen_0a:d8:3c	Broadcast	ARP	42 Who has 192.168.1.23? Tell 192.168.1.2
	222 9.044488	Shenzhen_0a:d8:3c	Broadcast	ARP	42 Who has 192.168.1.1? Tell 192.168.1.2
	11 0.135544	vivoMobi_e6:fe:9b	Broadcast	ARP	42 Who has 192.168.1.1? Tell 192.168.1.102
		_			

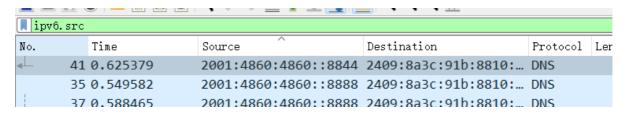
### DNS协议

DNS协议主要基于UDP,也可以使用TCP, 但二者都是基于IP协议的,但为什么这里源IP和目的IP为空呢?

下面在wireshark里过滤出dns协议来分析一下



如上图所示,可以观察到source和desitination都是ipv6的格式,推测 ip.src 这个语法过滤的是ipv4地址,虾米那用 ipv6.src 来验证,如下图所示



#### **OxFFFA**

这实际不是一个协议名,而是以太网类型码,属于最底层网络接口层,计算机网络中下层访问不到上层,没有IP的概念,自然没有源IP和目的IP

### 总结

通过列出src\_ip和dts\_ip都为空的数据包,得到以下三种协议没有源IP和目的IP:

- 1. ARP。原因: ARP协议并不是基于IP协议的,不能使用 ip.src 语法过滤,应该用 arp.src.proto\_ipv4
- 2. DNS。原因:本机抓取的DNS协议都是IPV6的,应使用 ipv6.src 来过滤
- 3. 0xFFFA。原因:IP在网际层,0xFFFA是以太网类型码,在网络接口层,该层在网际层之下,没有IP的概念

## 第二题

```
#include <iostream>
 1
    #include <stdio.h>
 2
 3
   #include <Windows.h>
 4
   #include <vector>
 5
   #include <sstream>
   #include <format>
 6
 7
   #include "rapidjson/document.h"
 8
   #include "rapidjson/writer.h"
    #include "rapidjson/prettywriter.h"
 9
10
    #include "rapidjson/stringbuffer.h"
11
12
    struct Packet {
13
        int frame_number = -1;
14
        std::string timestamp;
```

```
15
       std::string src_ip;
16
       int src_port = -1;
17
       std::string dst_ip;
18
       int dst_port = -1;
19
       std::string protocol;
20
        std::string info;
21
   };
22
23
24
   * 将tshark输出的一行数据解析成Packet结构体
25
   void parseLine(std::string line, Packet& packet);
26
27
28
    * 将Packet结构体转换成字符串格式
29
30
   std::string toString(Packet& packet);
31
32
33
   * 把端口字段的字符串转换成int
34
35
36
   int portToInt(std::string port);
37
38
39
   * 将Packet结构体转换成JSON对象
40
41
    rapidjson::Document toJSONDocument(const Packet& packet);
42
43
44
   * 将JSON对象转换成JSON字符串
45
   std::string toJSONString(const rapidjson::Document& document);
46
47
48
   int main() {
        SetConsoleOutputCP(CP_UTF8); // 解决控制台输出tashrk返回的中文时出现的乱码
49
    问题
50
        const char* command = "D:/wireshark/tshark.exe -r D:/temp/temp.pcap -T
51
    fields -e frame.number -e frame.time -e ip.src -e tcp.srcport -e ip.dst -e
    tcp.dstport -e _ws.col.Protocol -e _ws.col.Info";
52
        FILE* pipe = _popen(command, "r");
53
        if (!pipe) {
           std::cerr << "运行tshark失败!" << std::endl;
54
55
           return 1;
56
       }
57
58
        std::vector<Packet> packets;
59
60
        char buffer[4096];
        while (fgets(buffer, sizeof(buffer), pipe) != nullptr) {
61
            Packet packet;
62
63
           parseLine(buffer, packet);
64
           packets.push_back(packet);
65
        }
66
        std::cout << "Print in normal format: " << std::endl;</pre>
67
```

```
68
         for (auto& p : packets) {
 69
             std::cout << toString(p) << std::endl;</pre>
         }
 70
 71
 72
         std::cout << std::endl << "Print in JSON format: " << std::endl;</pre>
 73
         for (auto& p : packets) {
 74
             // 解析成JSON对象
 75
             rapidjson::Document json = toJSONDocument(p);
 76
             // 转换成字符串输出
 77
 78
             std::cout << toJSONString(json) << std::endl;</pre>
 79
         }
         _pclose(pipe);
 80
 81
         // 打印 源IP 和 目的IP 为空的数据包
 82
 83
         std::cout << std::endl << "Packet that the value of src_ip and dst_ip</pre>
     are blank:" << std::endl;</pre>
 84
         for (auto& p : packets) {
 85
             if (p.src_ip.empty() && p.dst_ip.empty()) {
 86
                  printf("protocol: %s\tinfo: %s\n", p.protocol.c_str(),
     p.info.c_str());
 87
             }
 88
         }
 89
     }
 90
 91
     void parseLine(std::string line, Packet& packet) {
 92
         if (line.back() == '\n') {
             line.pop_back();
 93
 94
         }
 95
 96
         std::stringstream ss(line);
 97
         std::string field;
 98
         std::vector<std::string> fields;
 99
         while (std::getline(ss, field, '\t')) {
100
             fields.push_back(field);
101
102
         }
103
         if (fields.size()) {
104
105
             packet.frame_number = std::stoi(fields[0]);
106
             packet.timestamp = fields[1];
             packet.src_ip = fields[2];
107
108
             packet.src_port = portToInt(fields[3]);
109
             packet.dst_ip = fields[4];
110
             packet.dst_port = portToInt(fields[5]);
             packet.protocol = fields[6];
111
112
             packet.info = fields[7];
113
         }
114
115
116
     std::string toString(Packet& packet) {
117
         std::string s;
         return std::format("frame_number: {0}\ttimestamp: {1}\tsrc_ip:
118
     {2}\tsrc_port: {3}\tdst_ip: {4}\tdst_port: {5}\tprotocol: {6}\tinfo: {7}",
119
             packet.frame_number,
120
             packet.timestamp,
```

```
121
             packet.src_ip,
122
             packet.src_port,
123
             packet.dst_ip,
124
             packet.dst_port,
125
             packet.protocol,
126
             packet.info
127
         );
128
     }
129
     int portToInt(std::string port) {
130
131
         try {
132
             return std::stoi(port);
133
         }
         catch (const std::exception e) {
134
135
             // 字符串转数字失败时,返回 -1
136
             return -1;
137
         }
138
     }
139
140
     rapidjson::Document toJSONDocument(const Packet& packet) {
         rapidjson::Document pktObj;
141
         rapidjson::Document::AllocatorType& allocator = pktObj.GetAllocator();
142
143
144
         pktObj.SetObject();
145
         pktObj.AddMember("frame_numer", packet.frame_number, allocator);
146
147
         pktObj.AddMember("timestamp",
     rapidjson::Value(packet.timestamp.c_str(), allocator);
148
         pktObj.AddMember("src_ip", rapidjson::Value(packet.src_ip.c_str(),
     allocator), allocator);
149
         pktObj.AddMember("src_port", packet.src_port, allocator);
         pktObj.AddMember("dst_ip", rapidjson::Value(packet.dst_ip.c_str(),
150
     allocator), allocator);
         pktObj.AddMember("dst_port", packet.dst_port, allocator);
151
         pktObj.AddMember("protocol", rapidjson::Value(packet.protocol.c_str(),
152
     allocator), allocator);
153
         pktObj.AddMember("info", rapidjson::Value(packet.info.c_str(),
     allocator), allocator);
154
155
         return pktObj;
156
157
     std::string toJSONString(const rapidjson::Document& document) {
158
159
         rapidjson::StringBuffer buffer;
160
         rapidjson::Writer<rapidjson::StringBuffer> writer(buffer);
161
         document.Accept(writer);
         return buffer.GetString();
162
163
     }
```