



哈爾濱工業大學 (深圳)
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期: 2023 春季
课程名称: 计算机网络
实验名称: 协议栈之 Eth、ARP 协议实现
学生班级: 5 班
学生学号: 200110513
学生姓名: 宗晴
评阅教师:
报告成绩:

实验与创新实践教育中心制

2023 年 3 月

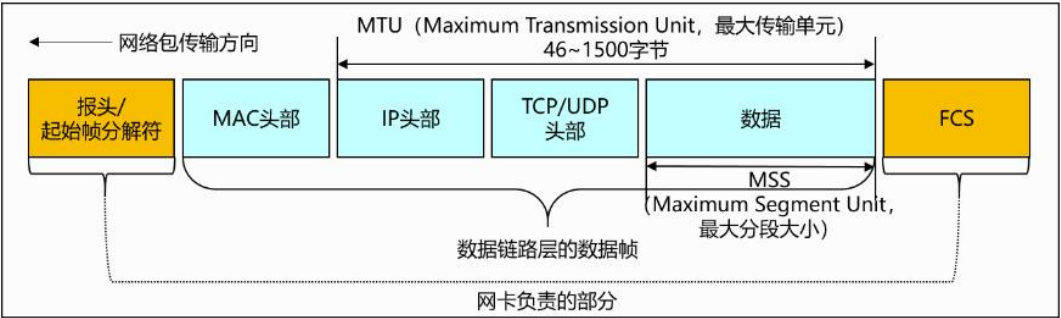
一、实验详细设计

注意不要完全照搬实验指导书上的内容，请根据你自己的设计方案来填写

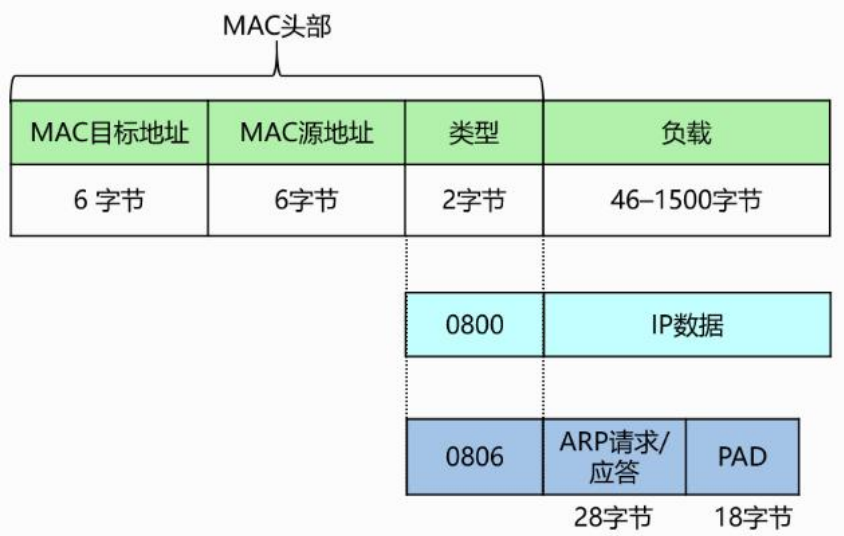
1. Eth 协议详细设计

来自物理线路的二进制数据包称作一个帧。以太网链路传输的数据帧称为以太帧，或者以太网数据帧。

在网线上传输的数据包如下图所示。其中，我们只需要关注 MAC 头部、IP 头部、TCP/UDP 头部、数据即可，其余部分（报头/起始帧分界符和 FCS）由网卡设备负责。



去除报头和 FCS，剩余的部分是数据链路层的数据帧，其帧格式如下图所示。



在该实验中，我们需要在给定的协议栈代码框架上，编写以太网数据链路层数据帧的发送和接收函数，使其能够发送和接收数据帧。

- (1) 完成以太网数据帧发送处理流程，即 `ethernet_out()` 函数

```

/**
 * @brief 处理一个要发送的数据包
 *
 * @param buf 要处理的数据包
 * @param mac 目标MAC地址
 * @param protocol 上层协议
 */
void ethernet_out(buf_t *buf, const uint8_t *mac, net_protocol_t protocol)
{
    // TO-DO
    // Step1 : 首先判断数据长度, 如果不足46则显式填充0, 填充可以调用buf_add_padding()函数来实现。
    if(buf->len < ETHERNET_MIN_TRANSPORT_UNIT)
        buf_add_padding(buf, 46 - buf->len);

    // Step2 : 调用buf_add_header()函数添加以太网包头。
    buf_add_header(buf, sizeof(ether_hdr_t));
    ether_hdr_t *hdr = (ether_hdr_t *)buf->data;

    // Step3 : 填写目的MAC地址。
    memcpy(hdr->dst, mac, NET_MAC_LEN);

    // Step4 : 填写源MAC地址, 即本机的MAC地址。
    memcpy(hdr->src, net_if_mac, NET_MAC_LEN);

    // Step5 : 填写协议类型 protocol。(大小端转换)
    hdr->protocol16 = swap16(protocol);

    // Step6 : 调用驱动层封装好的driver_send()发送函数, 将添加了以太网包头的数据帧发送到驱动层。
    driver_send(buf);
}

```

如上图所示, 首先需要判断数据长度, 如果不足 46, 即以太网最小传输单元, 则显式填充 0, 调用 `buf_add_padding()` 函数来实现填充。然后, 调用 `buf_add_header()` 函数添加以太网包头。用 `memcpy()` 函数填写目的 MAC 地址和源 MAC 地址, 即本机的 MAC 地址。然后填写协议类型 `protocol`, 此时需要调用 `swap16()` 函数从小端转化成大端。最后, 调用驱动层封装好的 `driver_send()` 发送函数, 将添加了以太网包头的数据帧发送到驱动层。

(2) 完成以太网数据帧接收处理流程, 即 `ethernet_in()` 函数

```

/**
 * @brief 处理一个收到的数据包
 *
 * @param buf 要处理的数据包
 */
void ethernet_in(buf_t *buf)
{
    // TO-DO
    // Step1 : 首先判断数据长度, 如果数据长度小于以太网头部长度, 则认为数据包不完整, 丢弃不处理。
    if(buf->len < sizeof(ether_hdr_t))
        return;

    // Step2 : 调用buf_remove_header()函数移除以太网包头。(在移除以太网包头前获取包的数据起始地址)
    ether_hdr_t *hdr = (ether_hdr_t *)buf->data;
    buf_remove_header(buf, sizeof(ether_hdr_t));

    // Step3 : 填写协议类型 protocol (大小端转换), 调用net_in()函数向上层传递数据包。
    net_protocol_t protocol = swap16(hdr->protocol16);
    net_in(buf, protocol, hdr->src);
}

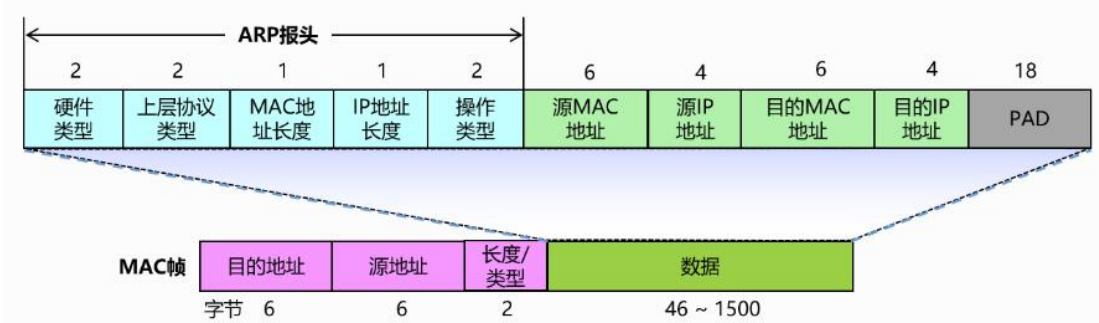
```

如上图所示, 首先判断数据长度, 如果数据长度小于以太网头部长度, 则认为数据包不完整, 丢弃不处理, 直接 return。否则, 调用 buf_remove_header()函数移除以太网包头。注意在移除以太网包头前需要获取包的数据起始地址, 因为后续需要使用到包头中的协议类型 protocol 和源 MAC 地址信息 src。然后, 填写协议类型 protocol, 此处需要将大端转换成小端。最后调用 net_in()函数向上层传递数据包。

2. ARP 协议详细设计

在 TCP/IP 的网络构造和网络通信中无需事先知道 MAC 地址究竟是什么, 只要确定了 IP 地址, 就可向这个目标地址发送 IP 数据报了。然而, 在数据链路层使用的是硬件地址 (MAC) 进行报文传输, IP 地址不能被物理网络所识别, 因此必须建立 IP 地址和 MAC 地址的映射关系, 这一过程称为 ARP (Address Resolution Protocol) 地址解析协议。

ARP 是一个独立的三层协议, 所以 ARP 报文在向数据链路层传输时不需要经过 IP 协议的封装, 而是直接生成自己的报文, 其中包括 ARP 报头, 到数据链路层后再对应的数据链路层 (如以太网协议) 进行封装。ARP 报文分为 ARP 请求和应答报文两种, 报文格式如下图所示:



在该实验中, 我们需要在完成协议栈之 eth 协议的基础上, 编写 ARP 报文的接收、发送和请求报文函数, 使其能够发送和接收 ARP 数据报文, 实现动态 ARP 表项。

(1) 完成 ARP 初始化, arp_init()函数已实现, 需实现 arp_req()函数

```

/**
 * @brief 发送一个arp请求
 *
 * @param target_ip 想要知道的目标的ip地址
 */
void arp_req(uint8_t *target_ip)
{
    // TO-DO
    // Step1 : 调用buf_init()对txbuf进行初始化。
    buf_init(&txbuf, sizeof(arp_pkt_t));

    // Step2 : 填写ARP报头。
    arp_pkt_t *arp = (arp_pkt_t *) txbuf.data;
    memcpy(arp, &arp_init_pkt, sizeof(arp_pkt_t));
    memcpy(arp->target_ip, target_ip, NET_IP_LEN);

    // Step3 : ARP操作类型为ARP_REQUEST, 注意大小端转换。
    arp->opcode16 = swap16(ARP_REQUEST);

    // Step4 : 调用ethernet_out函数将ARP报文发送出去。
    // 注意: ARP announcement或ARP请求报文都是广播报文, 其目标MAC地址应该是广播地址: FF-FF-FF-FF-FF-FF。
    uint8_t broadcast_mac[NET_MAC_LEN];
    for(int i = 0; i < NET_MAC_LEN; i ++ ) broadcast_mac[i] = 0xFF;
    ethernet_out(&txbuf, broadcast_mac, NET_PROTOCOL_ARP);
}

```

如上图所示，首先调用 `buf_init()` 对 `txbuf` 进行初始化。然后填写 ARP 报头，除 `arp_init_pkt` 中已经填充的部分，还需要将报头中的 `target_ip` 换成传入的 `target_ip`。接着设置 ARP 操作类型为 `ARP_REQUEST`，此处需注意大小端转换。最后，调用 `ethernet_out` 函数将 ARP 报文发送出去，由于 ARP announcement 或 ARP 请求报文都是广播报文，所以其目标 MAC 地址应该是广播地址：FF-FF-FF-FF-FF-FF。

(2) 完成 ARP 发送处理过程，即实现 `arp_out()` 函数

```

/**
 * @brief 处理一个要发送的数据包
 *
 * @param buf 要处理的数据包
 * @param ip 目标ip地址
 * @param protocol 上层协议
 */
void arp_out(buf_t *buf, uint8_t *ip)
{
    // TO-DO
    // Step1 : 调用map_get()函数, 根据IP地址来查找ARP表(arp_table)。
    uint8_t *mac = map_get(&arp_table, ip);

    // Step2 : 如果能找到该IP地址对应的MAC地址, 则将数据包直接发送给以太网层
    if(mac) ethernet_out(buf, mac, NET_PROTOCOL_IP);

    // Step3 : 如果没有找到对应的MAC地址, 则需要进一步判断arp_buf是否已经有包了
    else {
        // 如果有, 则说明正在等待该ip回应ARP请求, 此时不能再发送arp请求
        if(map_get(&arp_buf, ip)) return;
        else {
            // 如果没有包, 则调用map_set()函数将来自IP层的数据包缓存到arp_buf
            map_set(&arp_buf, ip, buf);
            // 调用arp_req()函数, 发一个请求目标IP地址对应的MAC地址的ARP request报文
            arp_req(ip);
        }
    }
}

```

如上图所示, 首先调用 `map_get()` 函数, 根据 IP 地址来查找 ARP 表(`arp_table`)。如果能找到该 IP 地址对应的 MAC 地址, 则将数据包直接发送给以太网层, 即调用 `ethernet_out` 函数直接发出去。如果没有找到对应的 MAC 地址, 则需要进一步判断 `arp_buf` 是否已经有包了, 如果有, 则说明正在等待该 ip 回应 ARP 请求, 此时不能再发送 arp 请求; 如果没有包, 则调用 `map_set()` 函数将来自 IP 层的数据包缓存到 `arp_buf`, 然后, 调用 `arp_req()` 函数, 发一个请求目标 IP 地址对应的 MAC 地址的 ARP request 报文。

(3) 完成 ARP 接收处理过程, 即实现 `arp_in()` 函数


```

/**
 * @brief 处理一个收到的数据包
 *
 * @param buf 要处理的数据包
 * @param src_mac 源mac地址
 */
void arp_in(buf_t *buf, uint8_t *src_mac)
{
    // TO-DO
    // Step1 : 首先判断数据长度, 如果数据长度小于ARP头部长度, 则认为数据包不完整, 丢弃不处理。
    if(buf->len < sizeof(arp_pkt_t)) return ;

    // Step2 : 接着, 做报头检查, 查看报文是否完整
    arp_pkt_t *arp = (arp_pkt_t *)buf->data;
    if (swap16(arp->hw_type16) != ARP_HW_ETHER || // 硬件类型
        swap16(arp->pro_type16) != NET_PROTOCOL_IP || // 上层协议类型
        arp->hw_len != NET_MAC_LEN || // MAC硬件地址长度
        arp->pro_len != NET_IP_LEN || // IP协议地址长度
        (swap16(arp->opcode16) != ARP_REQUEST && // 操作类型
         swap16(arp->opcode16) != ARP_REPLY)) return;
}

```

如上图所示, 首先判断数据长度, 如果数据长度小于 ARP 头部长度, 则认为数据包不完整, 丢弃不处理。接着, 做报头检查, 查看报文是否完整, 检测内容包括: ARP 报头的硬件类型、上层协议类型、MAC 硬件地址长度、IP 协议地址长度、操作类型, 检测该报头是否符合协议规定。

```

// Step3 : 调用map_set()函数更新ARP表项。
map_set(&arp_table, arp->sender_ip, arp->sender_mac);

// Step4 : 调用map_get()函数查看该接收报文的IP地址是否有对应的arp_buf缓存。
buf_t *arp_buf_item = map_get(&arp_buf, arp->sender_ip);

// 如果有, 则说明ARP分组队列里面有待发送的数据包。
if (arp_buf_item) {
    // 将缓存的数据包arp_buf再发送给以太网层
    ethernet_out(arp_buf_item, arp->sender_mac, NET_PROTOCOL_IP);
    // 将这个缓存的数据包删除掉
    map_delete(&arp_buf, arp->sender_ip);
}

// 否则, 还需要判断是否是请求本主机MAC地址的ARP请求报文
else{
    if (swap16(arp->opcode16) == ARP_REQUEST &&
        memcmp(arp->target_ip, net_if_ip, NET_IP_LEN) == 0) {
        // 回应一个响应报文
        arp_resp(arp->sender_ip, arp->sender_mac);
    }
}
}

```

然后调用 map_set()函数更新 ARP 表项。调用 map_get()函数查看该接收报文的 IP 地址是否有对应的 arp_buf 缓存。如果有, 则说明 ARP 分组队列里面有待发送的数据包。也就是上一次调用 arp_out()函数发送来自 IP 层的数据包时, 由于没有找到对应的 MAC

地址进而先发送的 ARP request 报文，此时收到了该 request 的应答报文。然后，将缓存的数据包 arp_buf 再发送给以太网层，即调用 ethernet_out() 函数直接发出去，接着调用 map_delete() 函数将这个缓存的数据包删除掉。

如果该接收报文的 IP 地址没有对应的 arp_buf 缓存，还需要判断接收到的报文是否为 ARP_REQUEST 请求报文，并且该请求报文的 target_ip 是本机的 IP，则认为是请求本主机 MAC 地址的 ARP 请求报文，则调用 arp_resp() 函数回应一个响应报文。

(4) 完成 ARP 响应包，即实现 arp_resp() 函数

```
/**
 * @brief 发送一个arp响应
 *
 * @param target_ip 目标ip地址
 * @param target_mac 目标mac地址
 */
void arp_resp(uint8_t *target_ip, uint8_t *target_mac)
{
    // TO-DO
    // Step1 : 首先调用buf_init()来初始化txbuf。
    buf_init(&txbuf, sizeof(arp_pkt_t));

    // Step2 : 接着，填写ARP报头首部。
    arp_pkt_t *arp = (arp_pkt_t *) txbuf.data;
    memcpy(arp, &arp_init_pkt, sizeof(arp_pkt_t));
    arp->opcode16 = swap16(ARP_REPLY);
    memcpy(arp->target_mac, target_mac, NET_MAC_LEN);
    memcpy(arp->target_ip, target_ip, NET_IP_LEN);

    // Step3 : 调用ethernet_out()函数将填充好的ARP报文发送出去。
    ethernet_out(&txbuf, arp->target_mac, NET_PROTOCOL_ARP);
}
```

如上图所示，首先调用 buf_init() 来初始化 txbuf。接着，填写 ARP 报头首部。此处除 arp_init_pkt 中已经填充的部分，还需要将报头中的 target_mac 和 target_ip 换成传入的 target_mac 和 target_ip。设置 ARP 操作类型为 ARP_REPLY，此处需注意大小端转换。最后，调用 ethernet_out() 函数将填充好的 ARP 报文发送出去。

二、实验结果截图及分析

1. Eth 协议实验结果及分析

eth_in 测试结果：


```

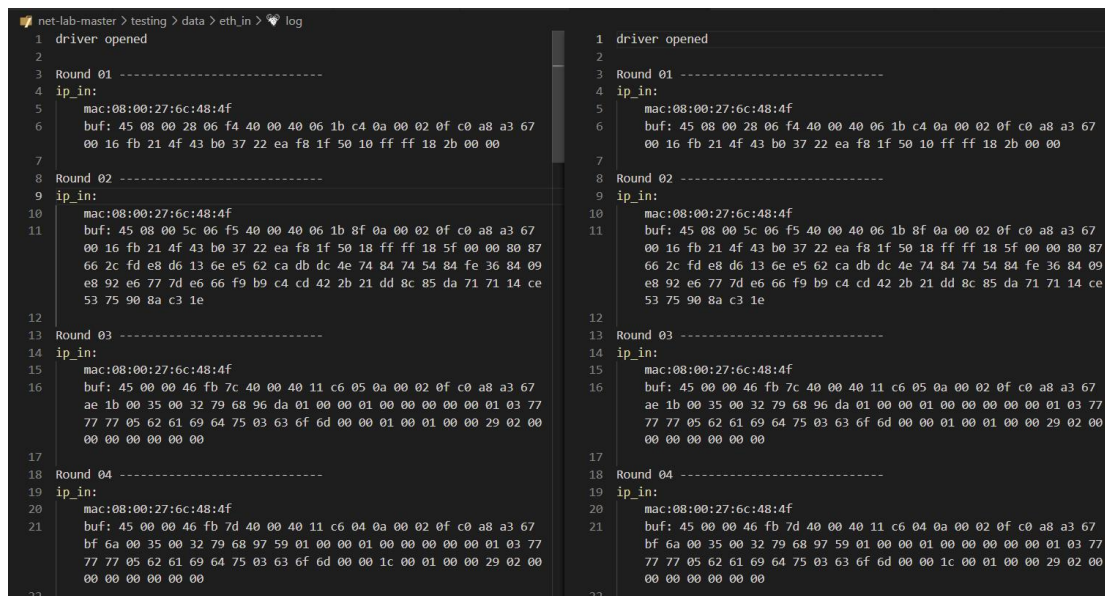
PS E:\lab2\net-lab-master> cd build
PS E:\lab2\net-lab-master\build> ctest -R eth_in
Test project E:/lab2/net-lab-master/build
  Start 1: eth_in
1/1 Test #1: eth_in ..... Passed    0.03 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =  0.04 sec
PS E:\lab2\net-lab-master\build> 

```

在 VSCode 工程目录下，将 testing/eth_in 目录下的 demo_log 和 log 这两个文件进行比对，发现二者一致：



```

net-lab-master > testing > data > eth_in > log
1 driver opened
2
3 Round 01 -----
4 ip_in:
5   mac:08:00:27:6c:48:4f
6   buf: 45 08 00 28 06 f4 40 00 00 06 1b c4 0a 00 02 0f c0 a8 a3 67
7   00 16 fb 21 4f 43 b0 37 22 ea f8 1f 50 10 ff ff 18 2b 00 00
8 Round 02 -----
9 ip_in:
10  mac:08:00:27:6c:48:4f
11  buf: 45 08 00 5c 06 f5 40 00 00 06 1b 8f 0a 00 02 0f c0 a8 a3 67
12  00 16 fb 21 4f 43 b0 37 22 ea f8 1f 50 18 ff ff 18 5f 00 00 80 87
13  66 2c fd e8 d6 13 6e e5 62 ca db dc 4e 74 84 74 54 84 fe 36 84 09
14  e8 92 e6 77 7d e6 66 f9 b9 c4 cd 42 2b 21 dd 8c 85 da 71 71 14 ce
15  53 75 90 8a c3 1e
16 Round 03 -----
17 ip_in:
18  mac:08:00:27:6c:48:4f
19  buf: 45 00 00 46 fb 7c 40 00 00 11 c6 05 0a 00 02 0f c0 a8 a3 67
20  ae 1b 00 35 00 32 79 68 96 da 01 00 00 01 00 00 00 00 01 03 77
21  77 77 05 62 61 69 64 75 03 63 6f 6d 00 00 01 00 01 00 00 29 02 00
22  00 00 00 00 00 00
23 Round 04 -----
24 ip_in:
25  mac:08:00:27:6c:48:4f
26  buf: 45 00 00 46 fb 7d 40 00 00 11 c6 04 0a 00 02 0f c0 a8 a3 67
27  bf 6a 00 35 00 32 79 68 97 59 01 00 00 01 00 00 00 00 00 01 03 77
28  77 77 05 62 61 69 64 75 03 63 6f 6d 00 00 1c 00 01 00 00 29 02 00
29  00 00 00 00 00 00

```

eth_out 测试结果：

```

PS E:\lab2\net-lab-master\build> ctest -R eth_out
Test project E:/lab2/net-lab-master/build
  Start 2: eth_out
1/1 Test #2: eth_out ..... Passed    0.02 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =  0.03 sec
PS E:\lab2\net-lab-master\build> 

```

在 VSCode 工程目录下，将 testing/eth_out 目录下的 demo_log 和 log 这两个文件进行比对，发现二者一致：

```

net-lab-master > testing > data > eth_out > log
1 driver opened
2
3 Round 01 -----
4
5 Round 02 -----
6
7 Round 03 -----
8
9 Round 04 -----
10
11 Round 05 -----
12
13 Round 06 -----
14
15 Round 07 -----
16
17 Round 08 -----
18
19 Round 09 -----
20
21 Round 10 -----
22
23 Round 11 -----
24
25 Round 12 -----
26
27 Round 13 -----
28
29 Round 14 -----
30
31 Round 15 -----
32
33 Round 16 -----
34

1 driver opened
2
3 Round 01 -----
4
5 Round 02 -----
6
7 Round 03 -----
8
9 Round 04 -----
10
11 Round 05 -----
12
13 Round 06 -----
14
15 Round 07 -----
16
17 Round 08 -----
18
19 Round 09 -----
20
21 Round 10 -----
22
23 Round 11 -----
24
25 Round 12 -----
26
27 Round 13 -----
28
29 Round 14 -----
30
31 Round 15 -----
32
33 Round 16 -----
34

```

2. ARP 协议实验结果及分析

arp_test 测试结果:

```

PS E:\lab2\net-lab-master\build> ctest -R arp_test
Test project E:/lab2/net-lab-master/build
  Start 3: arp_test
1/1 Test #3: arp_test ..... Passed    0.03 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =  0.04 sec
PS E:\lab2\net-lab-master\build>

```

在 VSCode 工程目录下, 将 testing/arp_test 目录下的 demo_log 和 log 这两个文件进行对比, 发现二者一致:

```

net-lab-master > testing > data > arp_test > log
1 driver opened
2 <===== arp table =====>
3 <===== arp buf =====>
4
5 Round 01 -----
6 <===== arp table =====>
7 <===== arp buf =====>
8 192.168.163.10 -> 45 00 00 46 fb 7c 40 00 40 11 77 67 c0 a8 a3 67 c0
a8 a3 0a ae 1b 00 35 00 32 79 68 96 da 01 00 00 01 00 00 00 00 01
03 77 77 77 05 62 61 69 64 75 03 63 6f 6d 00 00 01 00 01 00 00 29 02
00 00 00 00 00 00
9
10 Round 02 -----
11 <===== arp table =====>
12 192.168.163.10 -> 21:32:43:54:65:06
13 <===== arp buf =====>
14
15 Round 03 -----
16 <===== arp table =====>
17 192.168.163.10 -> 21:32:43:54:65:06
18 <===== arp buf =====>
19
20 Round 04 -----
21 ip_in:
22 mac:21:32:43:54:65:06
23 buf: 45 00 00 03 00 04 00 00 40 11 70 3c c0 a8 a3 67 c0 67

1 driver opened
2 <===== arp table =====>
3 <===== arp buf =====>
4
5 Round 01 -----
6 <===== arp table =====>
7 <===== arp buf =====>
8 192.168.163.10 -> 45 00 00 46 fb 7c 40 00 40 11 77 67 c0 a8 a3 67 c0
a8 a3 0a ae 1b 00 35 00 32 79 68 96 da 01 00 00 01 00 00 00 00 01
03 77 77 77 05 62 61 69 64 75 03 63 6f 6d 00 00 01 00 01 00 00 29 02
00 00 00 00 00 00
9
10 Round 02 -----
11 <===== arp table =====>
12 192.168.163.10 -> 21:32:43:54:65:06
13 <===== arp buf =====>
14
15 Round 03 -----
16 <===== arp table =====>
17 192.168.163.10 -> 21:32:43:54:65:06
18 <===== arp buf =====>
19
20 Round 04 -----
21 ip_in:
22 mac:21:32:43:54:65:06
23 buf: 45 00 00 03 00 04 00 00 40 11 70 3c c0 a8 a3 67 c0 67

```