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

实验报告

开课学期：2022 春季

课程名称：计算机网络

实验名称：协议栈之 IP、ICMP、UDP 协议实现

学生班级：计算机 10 班

学生学号：190111010

学生姓名：李诺舟

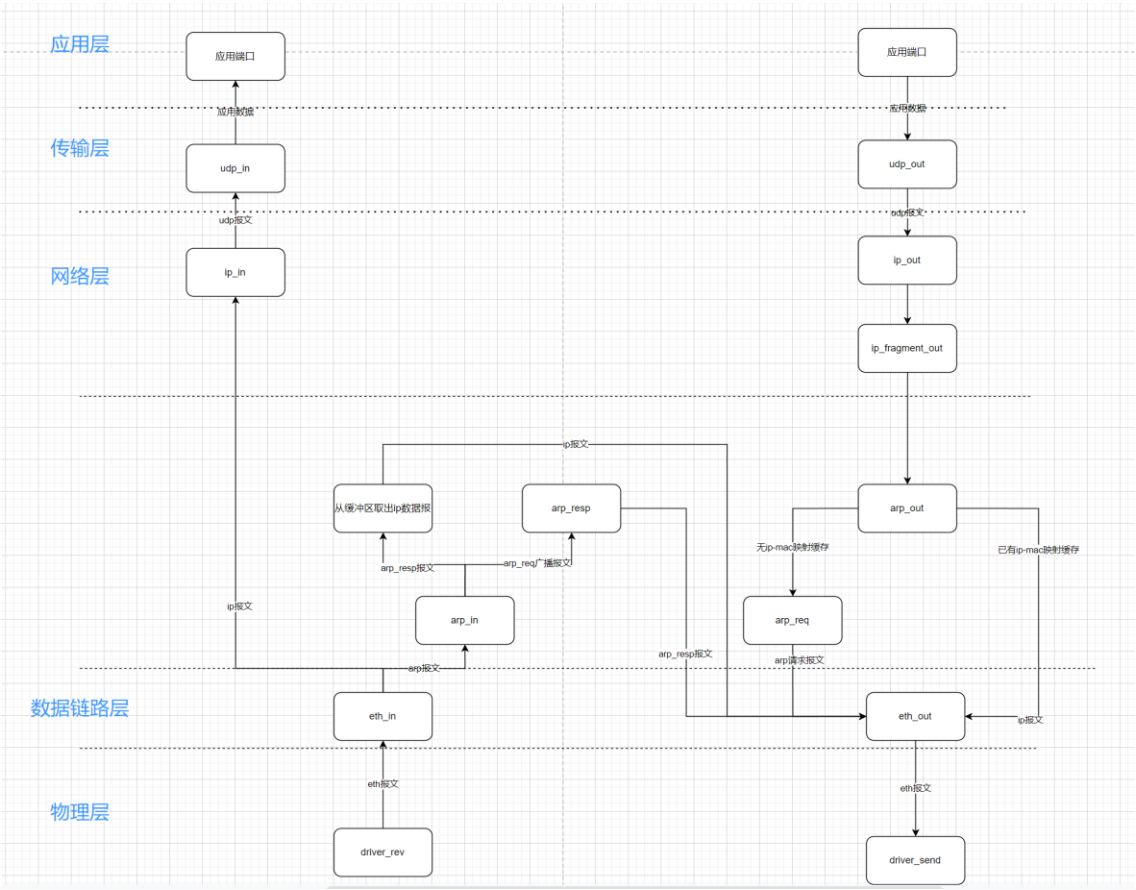
评阅教师：

报告成绩：

实验与创新实践教育中心制

2022 年 3 月

一、 实验详细设计

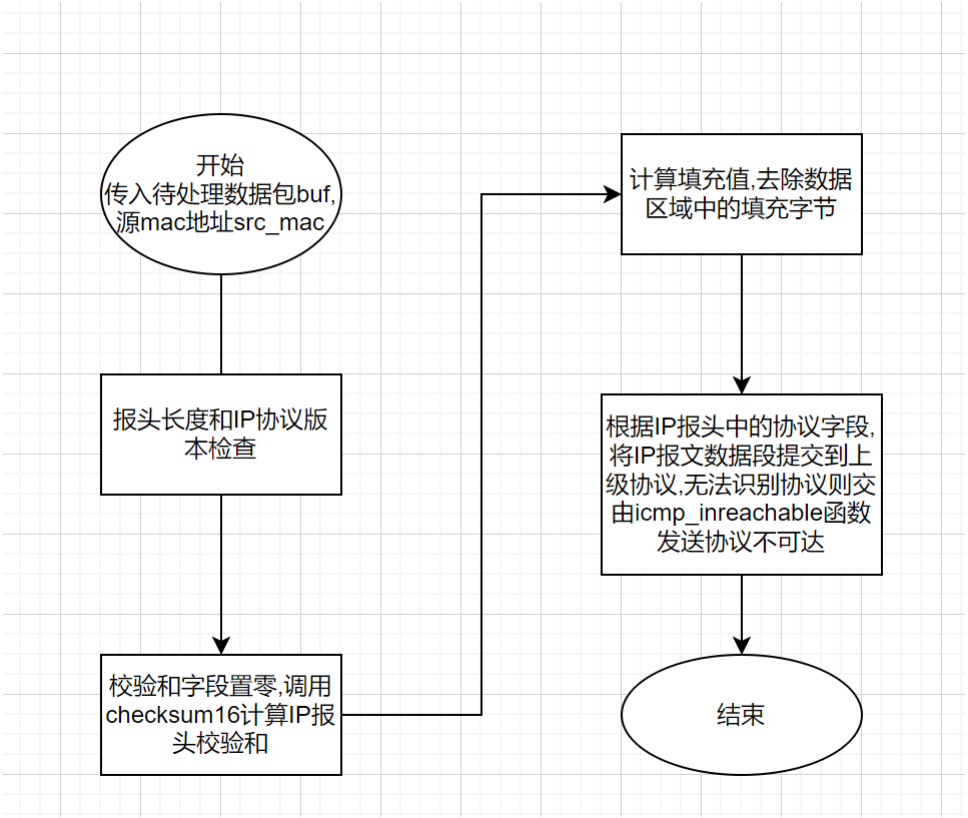


1. IP 协议详细设计

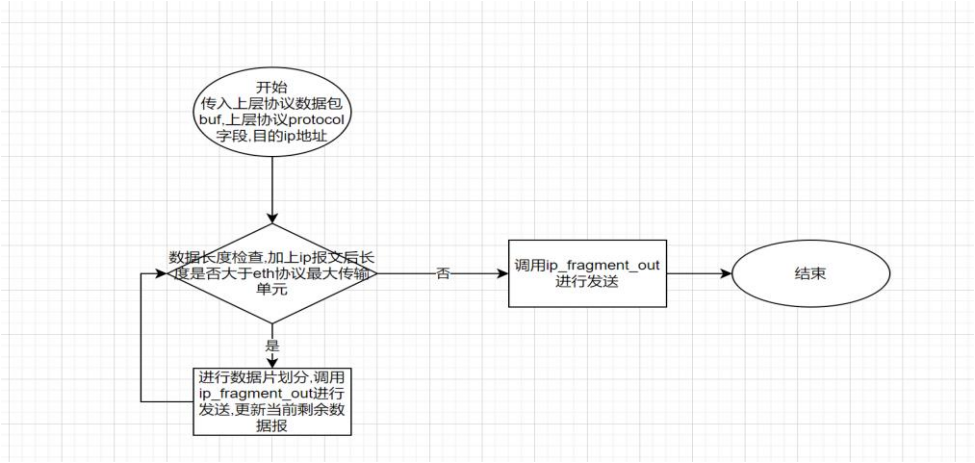
Ip 协议主要由 ip_in,ip_out 两个函数实现 ip 数据报的收发功能.收发过程中会调用 checksum16 函数计算 ip 头部的校验和.当要发送的 ip 数据包长度过长时,ip_in 会调用 ip_fragment_out 函数进行分片再发送.

整体架构:

ip_in:



ip_out:



2. ICMP 协议详细设计

当网络报文传输出现错误时,接受方就会发送 icmp 报文表示发生了传输错误.icmp 协议由 icmp_in 负责处理收到的 icmp 数据包,

```c
void icmp_in(buf_t *buf, uint8_t *src_ip)
{
    // TO-DO
    if(buf->len >= 8){
        icmp_hdr_t *hdr = buf->data;
        if(hdr->type == ICMP_TYPE_ECHO_REQUEST){
            icmp_resp(buf,src_ip);
        }
    }
}
```

icmp_unreachable 负责处理发送 icmp 不可达报文.

```c
void icmp_unreachable(buf_t *recv_buf, uint8_t *src_ip, icmp_code_t code)
{
    ip_hdr_t *err_ip_hdr = recv_buf->data;
    buf_init(&txbuf,err_ip_hdr->hdr_len*4+8);
    memcpy(txbuf.data,recv_buf->data,txbuf.len);
    buf_add_header(&txbuf,8);
    icmp_hdr_t *hdr = txbuf.data;
    hdr->type = ICMP_TYPE_UNREACH;
    hdr->code = code;
    hdr->checksum16 = 0;
    hdr->id16 = 0;
    hdr->seq16 = 0;
    hdr->checksum16 = checksum16(hdr,txbuf.len);
    ip_out(&txbuf,src_ip,NET_PROTOCOL_ICMP);
}
```

本次实验中 icmp_in 只实现了对 echo_request 类型的 icmp 报文处理.该处理会调用 icmp_resp 函数发送一个类型为 icmp_reply 的 icmp 数据报.

```c
static void icmp_resp(buf_t *req_buf, uint8_t *src_ip)
{
    // TO-DO
    //buf_init(&txbuf,req_buf->len);
    icmp_hdr_t *hdr = req_buf->data;
    hdr->type = ICMP_TYPE_ECHO_REPLY;
    hdr->code = 0;
    hdr->checksum16 = 0;
    hdr->checksum16 = checksum16(hdr,req_buf->len);
    ip_out(req_buf,src_ip,NET_PROTOCOL_ICMP);
}
```

3. UDP 协议详细设计

Udp 协议主要由 udp_in 函数负责处理收到的 udp 数据包,处理收到的数据包时,首先会将源 ip 地址保存在另外的内存中,防止进行 udp 校验和计算时错误修改源 ip 值.通过校验后,解析得到目的端口地址,在哈希表中找到对应的处理函数,调用处理函数.

```c
void udp_in(buf_t *buf, uint8_t *src_ip)
{
    // TO-DO
    printf("udp_in!\n");
    if(buf->len >= sizeof(udp_hdr_t)){
        uint8_t src_ip_temp[4];
        udp_hdr_t *hdr = buf->data;
        memcpy(src_ip_temp,src_ip,4);
        uint16_t src_checksum = hdr->checksum16;
        hdr->checksum16 = 0;
        hdr->checksum16 = udp_checksum(buf,src_ip,net_if_ip);
        if(hdr->checksum16 == src_checksum){
            uint16_t port = swap16(hdr->dst_port16);
            udp_handler_t* handler = (udp_handler_t*)map_get(&udp_table,&port);
            if(handler){
                buf_remove_header(buf,sizeof(udp_hdr_t));
                (*handler)(buf->data,buf->len,src_ip_temp,swap16(hdr->src_port16));
            }
            else{
                buf_add_header(buf,sizeof(ip_hdr_t));
                ip_hdr_t *ip_hdr = buf;
                memcpy(ip_hdr -> dst_ip,net_if_ip,4);
                ip_hdr -> hdr_len = 5;
                icmp_unreachable(buf,src_ip_temp,ICMP_CODE_PORT_UNREACH);
            }
        }
    }
}
```

处理函数 handler 会将得到的数据重新送回发送端,调用 udp_send,udp_send 会调用 udp_out 函数.udp_out 函数负责发送 udp 数据报,将 udp 报头填写好后会调用 ip_out 发送报文.

```c
void udp_out(buf_t *buf, uint16_t src_port, uint8_t *dst_ip, uint16_t dst_port)
{
    // TO-DO
    buf_add_header(buf,sizeof(udp_hdr_t));
    udp_hdr_t *hdr = buf->data;
    hdr->dst_port16 = swap16(dst_port);
    hdr->src_port16 = swap16(src_port);
    hdr->total_len16 = swap16(buf->len);
    hdr->checksum16 = 0;
    hdr->checksum16 = udp_checksum(buf,net_if_ip,dst_ip);
    ip_out(buf,dst_ip,NET_PROTOCOL_UDP);
}
```

1. IP 协议实验结果及分析

```
    Start 4: ip_test
4/6 Test #4: ip_test ..........................   Passed    0.01 sec
    Start 5: ip_frag_test
5/6 Test #5: ip_frag_test .....................   Passed    0.00 sec
```

Ip 协议运转正常,能够通过分片和未分片测试.

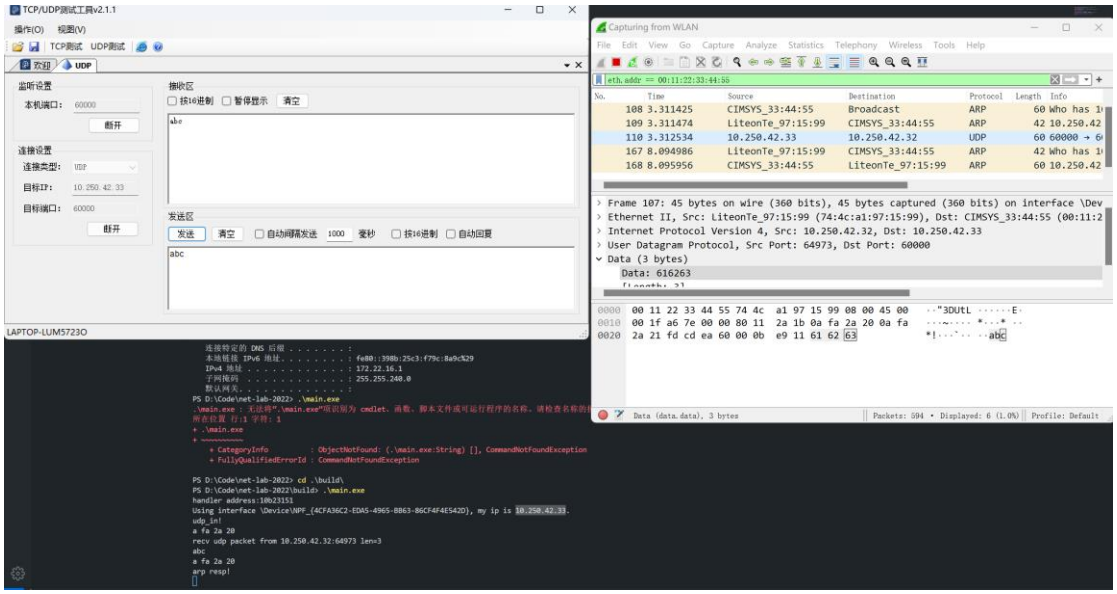
2. ICMP 协议实验结果及分析

```
    Start 6: icmp_test
6/6 Test #6: icmp_test ........................   Passed    0.01 sec
```

Icmp 协议运转正常,可以处理对应的传输错误和 icmp 类型的请求.

3. UDP 协议实验结果及分析

（本小节还需要分析你自己用 Wireshark 抓包工具捕获到的相关报文（包含 UDP 和 ARP 报文），解析报文内容）



Wireshark 抓包报文分析:

No.	Time	Source	Destination	Protocol	Length	Info
107	3.305139	10.250.42.32	10.250.42.33	UDP	45	64973 → 60000 Len=3
108	3.311425	CIMSYS_33:44:55	Broadcast	ARP	60	Who has 10.250.42.32? Tell 10.250.42.33
109	3.311474	LiteonTe_97:15:99	CIMSYS_33:44:55	ARP	42	10.250.42.32 is at 74:4c:a1:97:15:99
110	3.312534	10.250.42.33	10.250.42.32	UDP	60	60000 → 60000 Len=3
167	8.094986	LiteonTe_97:15:99	CIMSYS_33:44:55	ARP	42	Who has 10.250.42.33? Tell 10.250.42.32
168	8.095956	CIMSYS_33:44:55	LiteonTe_97:15:99	ARP	60	10.250.42.33 is at 00:11:22:33:44:55

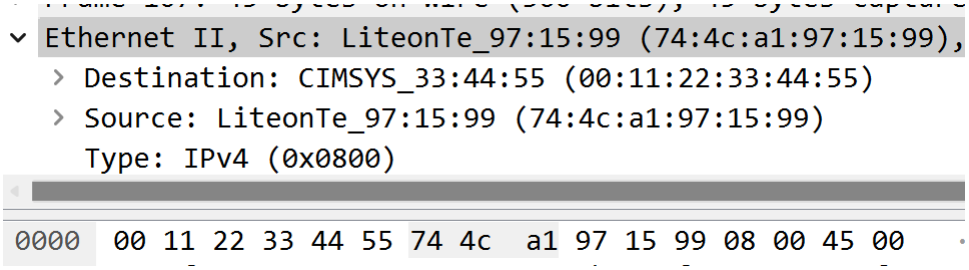
整个通信过程如上图所示,首先主机向测试平台发送 udp 报文,内容为 abc,在先前主机已经事先得到了测试平台 ip 与 mac 的映射关系,所以可以直接发送 udp 报文.测试平台收到 udp 报文后,将其重新发送回主机,但是此时测试平台并不知道主机 ip 对应的 mac 地址.于是测试平台先发送一个 arp_req 的广播报文,主机收到后发送 arp_resp 报文告诉测试平台主机的 mac 地址为 74:4c:a1:97:15:99,测试平台得到主机 ip 对应的 mac 地址后,将原先缓存的 ip 数据包发送给主机.此时 wireshark 抓包得到了该 udp 报文.发送完成后过一段时间,主机中关于测试平台的 mac-ip 映射关系失效,于是重新发送 arp_req 广播报文,更新 mac 地址.具体不同类型报文解析如下:

No107:

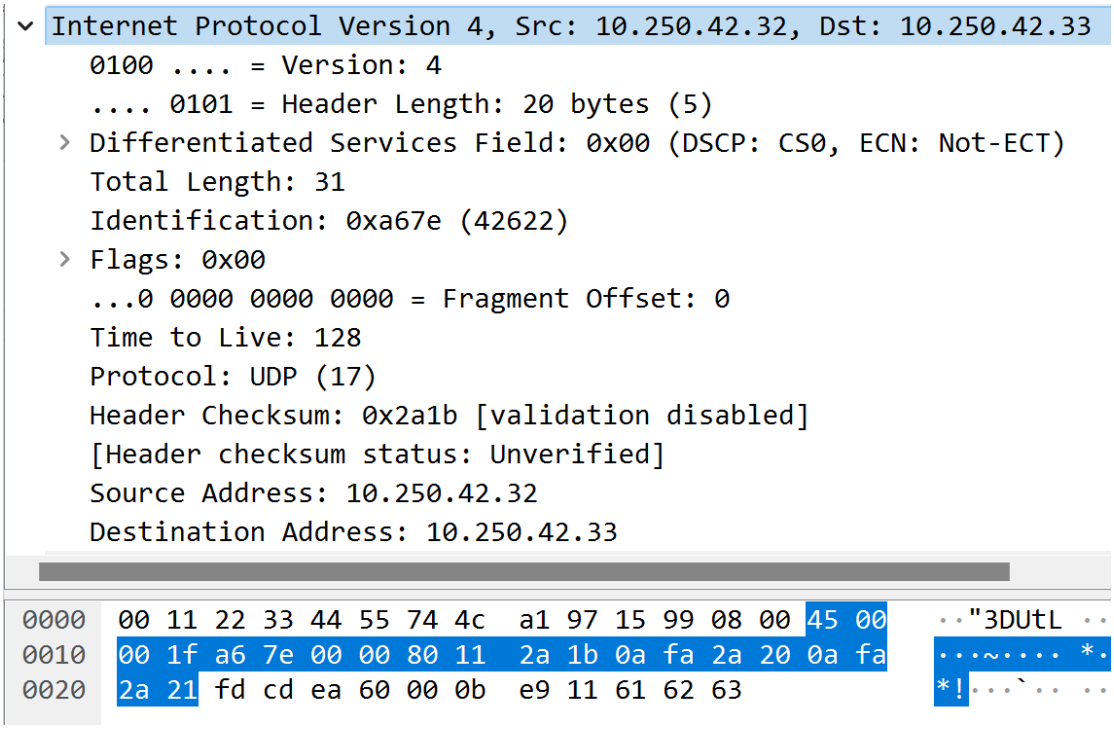
报文内容:

```
0000  00 11 22 33 44 55 74 4c  a1 97 15 99 08 00 45 00
0010  00 1f a6 7e 00 00 80 11  2a 1b 0a fa 2a 20 0a fa
0020  2a 21 fd cd ea 60 00 0b  e9 11 61 62 63
```

Eth 报头:


Ethernet II, Src: LiteonTe_97:15:99 (74:4c:a1:97:15:99),
Destination: CIMSYS_33:44:55 (00:11:22:33:44:55)
Source: LiteonTe_97:15:99 (74:4c:a1:97:15:99)
Type: IPv4 (0x0800)
0000 00 11 22 33 44 55 74 4c a1 97 15 99 08 00 45 00

Ip 报头:


Internet Protocol Version 4, Src: 10.250.42.32, Dst: 10.250.42.33
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 31
Identification: 0xa67e (42622)
Flags: 0x00
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 128
Protocol: UDP (17)
Header Checksum: 0x2a1b [validation disabled]
[Header checksum status: Unverified]
Source Address: 10.250.42.32
Destination Address: 10.250.42.33
0000 00 11 22 33 44 55 74 4c a1 97 15 99 08 00 45 00
0010 00 1f a6 7e 00 00 80 11 2a 1b 0a fa 2a 20 0a fa
0020 2a 21 fd cd ea 60 00 0b e9 11 61 62 63

Udp 报头:

```
User Datagram Protocol, Src Port: 64973, Dst Port: 60000
    Source Port: 64973
    Destination Port: 60000
    Length: 11
    Checksum: 0xe911 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 32]
  > [Timestamps]
    UDP payload (3 bytes)
> Data (3 bytes)
```

```
0000  00 11 22 33 44 55 74 4c  a1 97 15 99 08 00 45 00   ..
0010  00 1f a6 7e 00 00 80 11  2a 1b 0a fa 2a 20 0a fa   ...
0020  2a 21 fd cd ea 60 00 0b  e9 11 61 62 63            *!
```

数据:

```
Data (3 bytes)
    Data: 616263
    [Length: 3]
```

```
0000  00 11 22 33 44 55 74 4c  a1 97 15 99 08 00 45 00
0010  00 1f a6 7e 00 00 80 11  2a 1b 0a fa 2a 20 0a fa
0020  2a 21 fd cd ea 60 00 0b  e9 11 61 62 63
```

No108:

报文内容:

```
0000  ff ff ff ff ff ff 00 11  22 33 44 55 08 06 00 01
0010  08 00 06 04 00 01 00 11  22 33 44 55 0a fa 2a 21
0020  00 00 00 00 00 00 0a fa  2a 20 00 00 00 00 00 00
0030  00 00 00 00 00 00 00 00  00 00 00 00
```

Eth 报头:

```
Ethernet II, Src: CIMSYS_33:44:55 (00:11:22:33:44:55), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  > Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  > Source: CIMSYS_33:44:55 (00:11:22:33:44:55)
    Type: ARP (0x0806)
    Padding: 000000000000000000000000000000000000
```

```
0000  ff ff ff ff ff ff 00 11  22 33 44 55 08 06 00 01   ........ "3DU....
0010  08 00 06 04 00 01 00 11  22 33 44 55 0a fa 2a 21   ........ "3DU..*!
0020  00 00 00 00 00 00 0a fa  2a 20 00 00 00 00 00 00   ........ * ......
0030  00 00 00 00 00 00 00 00  00 00 00 00              ........ ....
```

Arp_req 报头:

```
Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: CIMSYS_33:44:55 (00:11:22:33:44:55)
    Sender IP address: 10.250.42.33
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.250.42.32
```

```
0000  ff ff ff ff ff ff 00 11  22 33 44 55 08 06 00 01   ······ ·"3DU···
0010  08 00 06 04 00 01 00 11  22 33 44 55 0a fa 2a 21   ······ ·"3DU··*!
0020  00 00 00 00 00 00 0a fa  2a 20 00 00 00 00 00 00   ······ *·······
0030  00 00 00 00 00 00 00 00  00 00 00 00              ······ ····
```

No110:

报文内容:

```
0000  00 11 22 33 44 55 74 4c  a1 97 15 99 08 06 00 01
0010  08 00 06 04 00 02 74 4c  a1 97 15 99 0a fa 2a 20
0020  00 11 22 33 44 55 0a fa  2a 21
```

Eth 报头:

```
Ethernet II, Src: LiteonTe_97:15:99 (74:4c:a1:97:15:99), Dst: CIMSYS_33:44:55 (00:11:22:33:44:55)
  > Destination: CIMSYS_33:44:55 (00:11:22:33:44:55)
  > Source: LiteonTe_97:15:99 (74:4c:a1:97:15:99)
    Type: ARP (0x0806)
```

```
0000  00 11 22 33 44 55 74 4c  a1 97 15 99 08 06 00 01   ··"3DUtL ······ ··
0010  08 00 06 04 00 02 74 4c  a1 97 15 99 0a fa 2a 20   ····· tL ····· *
0020  00 11 22 33 44 55 0a fa  2a 21                     ··"3DU·· *!
```

Arp_resp 报头:

```
Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: LiteonTe_97:15:99 (74:4c:a1:97:15:99)
    Sender IP address: 10.250.42.32
    Target MAC address: CIMSYS_33:44:55 (00:11:22:33:44:55)
    Target IP address: 10.250.42.33
```

```
0000  00 11 22 33 44 55 74 4c  a1 97 15 99 08 06 00 01   ··"3DUtL ······ ··
0010  08 00 06 04 00 02 74 4c  a1 97 15 99 0a fa 2a 20   ····· tL ····· *
0020  00 11 22 33 44 55 0a fa  2a 21                     ··"3DU·· *!
```

二、 实验中遇到的问题及解决方法

①本次实验前期在 wsl2 的 linux 平台上进行,在 vscode 插件安装使用上较为便捷.但是在 udp 实验调试阶段,使用 linux 的网络工具 cocat 进行 udp 通信时,cocat 可以成功连接测试平台端口和发送数据,但是测试平台的输入只有 arp 数据报,没有 udp 数据报,无法继续进行测试,于是最终转到 windows 平台上完成最后的测试

②在 ip 协议的实验中,实验提供的 ip 数据报头的 version 和 hdr_len 字段都是 8 位长度,这不符合 ipv4 协议,所以直接拿该结构体解析报头会发生错误,需要进行修改.

三、 实验收获和建议

通过本次实验,我对于 tcp 的五层模型有了更加深入地理解,对于网络栈协议的运作过程有了一个直观的体验.实验提供的测试机制很大程度上方便了调试的进行,但还是希望给出一个在 linux 上进行 udp 实验测试的教程加以补充,方便同学们自由选择实验环境.