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 IP、ICMP、UDP 协议实现

学生班级：6 班

学生学号：200110611

学生姓名：王志铭

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图文并茂地描述实验实现的所有功能和详细的设计方案及实验过程中的特色部分。）

1. IP 协议详细设计

IP 协议需要完成 IP 报文的发送和接受，对应了实验当中的 ip_in, ip_out, ip_fragment_out, checksum16. 这四个函数有互相依赖的关系，本文依此阐述设计过程。

i. Checksum16

IP 报文在发送和接受的时候都需要检验可能的报文损坏，因此需要引入校验和来进行检查。

校验和算法的步骤：

a. 将数据流分解成若干个 16 位的 uint，进行相加得到 sum

b. 将 sum 的高 16 位与低 16 位相加，直到 sum 的高 16 位为 0.

c. 将 sum 取反即可。

在算法的过程当中注意 swap。

ii. Ip_in

此函数模拟了接收一个 ip 报文的过程。主要流程如下：

a. 长度检查：长度不小于 IP header

b. 报头检查：

a) 版本号为 IPv4

b) 总长度字段小于等于收到的包长度

c. 检验和检查：

a) 暂存 header 的校验和

b) 重置 header 的 checksum16，计算此时 header 的 checksum16 是否与原来的相同

d. IP 地址检查：目的 IP 为本机 IP

e. 去除填充与报头

f. 向上传递数据包，如果不能识别协议就用 icmp_unreachable()返回不可达信息。

iii. Ip_fragment_out

此函数实现了一个 ip 片段的发送。主要设计流程如下：

a. 添加 ip 头，填写字段

b. 把 header 的 checksum16 置 0，计算校验和

c. 用 arp_out 将其发送出去。

iv. Ip_out

此函数实现了完整 ip 报文的发送。

如果 ip 报文的长度超过了最大负载，则需要将数据包拆分成若干个包，调用 ip_fragment_out 发送出去。

注意，最后一个分片的 MF = 0，其余 MF = 1.

2. ICMP 协议详细设计

本部分的设计主要包括了两个函数：icmp_in 和 icmp_resp、icmp_unreachable.

i. Icmp_in

实现输入 icmp 报文的处理。主要流程如下：

a. 报头检查：要求包长不小于 icmp 头部长

b. 查看报文是否是 icmp 的回显请求，如果是，则调用 icmp_resp 发送回应。

ii. Icmp_resp

此函数负责发送一个 icmp 回应。主要流程如下：

a. 初始化 txbuf

b. 填写报头和数据，其中数据可以直接复制接收报文

c. 调用 ip_out 发送

iii. Icmp_unreachable

此函数处理报文差错的情况，此处端口/协议不可达。主要流程如下：

a. 初始化 txbuf

b. 填写 icmp 报文和数据部分，其中数据指 ip 报文首部和前 8B

c. 调用 ip_out 发送

3. UDP 协议详细设计

Udp 协议的处理也比较简单，主要由 udp_checksum, udp_in, udp_out 组成。

i. Udp_checksum

Udp checksum 与 checksum 是类似的，只不过多了伪头部的概念。

a. 增加 udp 伪头部，由于伪头部覆盖了前面的 ip 头，所以要先保存 ip 头

b. 填写 udp 伪头部

c. 基于 udp 伪头部计算 checksum

d. 去除伪头部，恢复 ip 头

ii. Udp_out

a. 添加 udp 报头，并填写相关的字段

b. 计算 udp checksum，填入报头

c. 调用 ip_out 发送

iii. Udp_in

a. 包检查：要求报长不小于 udp 头长，以及 udp 首部给出的长度

b. 校验和检查：保存原校验和，然后重新调用 udp_checksum 计算，要求校验和一致。

c. 从 udp_table 查询端口对应的处理函数

a) 如果有，则去除 udp 报头，调用回调函数来处理

b) 如果没有，则发送 icmp_unreachable()

二、 实验结果截图及分析

（对你自己实验的测试结果进行评价）

1. IP 协议实验结果及分析

```
wangzhiming@MBP-2FVFJQMD6M-0302 build % ctest -R ip_frag_test
Test project /Users/wangzhiming/Desktop/net-stacks/build
    Start 5: ip_frag_test
1/1 Test #5: ip_frag_test .....................   Passed    0.01 sec

100% tests passed, 0 tests failed out of 1
```

```
wangzhiming@MBP-2FVFJQMD6M-0302 build % ctest -R ip_test
Test project /Users/wangzhiming/Desktop/net-stacks/build
    Start 4: ip_test
1/1 Test #4: ip_test ..........................   Passed    0.03 sec

100% tests passed, 0 tests failed out of 1
```

2. ICMP 协议实验结果及分析

```
wangzhiming@MBP-2FVFJQMD6M-0302 build % ctest -R icmp_test
Test project /Users/wangzhiming/Desktop/net-stacks/build
    Start 6: icmp_test
1/1 Test #6: icmp_test ........................   Passed    0.03 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   0.04 sec
```

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）

发送一个报文内容为“abc”的 udp 包，然后用 wireshark 抓包分析。

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

通信过程：

主机向测试机发送 udp 报文（内容为 abc），由于主机之前已经知道了测试机 ip 和 mac，所以可以直接发送 udp 报文。

测试机收到 udp 包之后，想要向主机 echo 一个一样的 udp 报，但是此时测试机不知道主机的 mac，于是他发起了 arp 广播，询问主机 ip 对应的 mac。主机收到后，回复一个 arp_resp，测试机得到主机的 mac 之后，将 buf_table 中缓存的 ip 包发给主机，于是 wireshark 捕获到第二个 udp 报文。

接下来分析第一个 udp 报：

从 ip 报头开始看起：

```
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
```

可以看到 ip 报头中装载了 ip 版本、ip 地址、长度、校验和、协议等信息，通过检查之后，ip 报体向上传递到传输层，交由 udp 处理。

```
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
```

可以看到，udp 报头中装载了端口号、校验和、长度等信息，而 udp 报文的数据内容为 61 62 63，即”abc”。

接下来分析 arp_req。

```
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
```

可以看到，此 arp req 报是基于 ipv4 协议的，类型为 request。

目标 mac 为 0，表明这是一个广播，向所有网络内主机寻找目标 ip 的 mac 地址。

而对于 arp resp：

```
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
```

可以看到，arp resp 报同样基于 IPv4 协议，操作类型为 reply。此时目标主机接收到 arp req，于是在 arp resp 当中填上了自己的 mac 地址，发送给测试机。

三、 实验中遇到的问题及解决方法

（包括设计过程中的错误及测试过程中遇到的问题）

1. 校验和剩余 8bit
   一开始没有很好地理解剩下的 8bit 要怎么处理，做到 udp 发送数据的时候就出了 bug。仔细理解之后，实际上原理还是类似的，给缺失的 8bit 补上 0，如 0xFF -> 0xFF00, 然后读进来变成 tmp = 0x00FF, 然后转换大小端，sum += swap(tmp)。
   在实现的时候用位运算重写了一种与上面等价的写法。

```
if (len == 1){
    uint8_t tmp = *data;
    sum += tmp << 8;
    break;
```

2. 校验和大小端 swap
   主要的问题是最后返回值的处理，一开始是直接返回，然后在外部 swap，但是后面发现这样做的话在后面检查和填写的时候相当混乱。
   直接在 checksum16()内部 swap，得到可以直接填入报头的校验和更清楚一些。
3. Ip_in 的协议过滤
   在 ip_in 的流程当中，最后需要判断协议是否可达。
   哪些协议可达？我一开始填写了 ICMP、UDP、TCP，但是在 test 时报错了。
   检查了 pcap 数据，发现问题在于 test 当中传了一个 TCP, 并且认为 TCP 是不可达的。这里出现了问题。在 test 当中只有 ICMP 和 UDP 是可达的。
4. Udp 伪头部拷贝问题
   Udp 的伪头部会覆盖一部分的 IP 头部，我们在处理这个的时候要格外小心，总的来说，多用深拷贝，否则很可能在修改 ip 地址的时候改变了指针所指的内容。
5. 初始值的问题
   有些字段的值可能是 0 之类的，但是不可以偷懒不赋值，实际上字段的初始值是不确定的，可能是 0 也可能是遗留下来的其他值，所以一定要显式地赋值。

四、 实验收获和建议

（实验过程中的收获、感受、问题、建议等）

通过这次实验，我对网络五层模型有了一个更好的理解，特别是层与层之间的交互过程，以及对一些细节协议如 icmp、arp 的理解也加深了。
做实验当中遇到了一些 bug，在 debug 的时候我发现除了要熟悉 wireshark 之外，阅读、解析日志的能力也十分重要，看日志能解决很多问题。

针对 ip_in 的协议过滤我提一个建议：

Icmp_test 当中认为只有 ICMP 和 UDP 是可达的。

我个人认为这样并不好，特别是我后面做了 TCP 的附加实验。做 TCP test 时要在这里加上 TCP 允许通过，在做 icmp_test 时又要删除。很不方便，希望能修改这个 test，用别的不可达协议测。