



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: 20 级计算机六班

学生学号: 200110632

学生姓名: 雷洵

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\_in()函数

先做 IP 报文合法性检测：

- a)若 buf 的长度小于 IP 数据报头的长度（20 字节），则直接 return；（丢弃报文）
- b)利用 ip\_hdr\_t 类型指针取出报文数据的 IP 数据报头
- c)对比 ip 版本号以及报头长度字段，若版本号不为 ipv4 或者 ip 报文长度大于 buf 的长度时，丢弃报文
- d)用一个 uint16 保存校验和，并将原本的校验和置零，将 hdr 指针再转化为 16 位无符号 int 类型的指针传入 checksum16 函数，报文长度则由 hdr 头乘以 4（ip 包头长度单位）计算后传入。若计算后的校验和与保存的校验和不同，则丢弃报文
- e)比较本地 ip 地址与目的 ip 是否相等（使用 memcmp 比较数组内容），不同则丢弃报文
- f)若收到的数据包长度长于 ip 数据包头记录的数据包总长度（需要将小段转换回大端），说明有填充字段，则使用函数 buf\_remove\_padding 将多余的部分截去

最后发送 ip 报文：

使用 buf\_remove\_header 去除 ip 报头，用一个 8 位无符号 int 取出上层协议字段，需要判断下协议类型，决定 net\_in 使用 ip 作为源地址参数或者 mac 作为源地址参数，例如 ICMP 报文就需要使用 ip 作为源地址参数，最后再调用 net\_in 函数将报文发往更上层的协议去处理。检测 net\_in 的返回值，若返回-1 则说明上层协议不可达，则需要使用头部拓展函数将报文的头部还原，再调用 icmp\_unreachable 函数报告

ip\_out()函数

维护一个全局变量 idcnt，其初值为 0

首先检测数据包长度是否大于 1480（一个 ip 数据包数据段所能承载最大的长度），若小于，则直接调用 ip\_fragment\_out 将整个数据包发送出去

若大于，则进行 ip 分片操作，初始化 offset 值为 0

首先申请一块 1480 长度的新数据块，用 memcpy 将原数据包 buf->data 字段的 1480 位数据拷入申请新数据块的 data 字段中，再调用 ip\_fragment\_out 函数将该分片发送，其中 id 为 idcnt 的值，标志位为 1，将偏移量自增 185（1480/8，以 8 字节为单位），最

后调用 `buf_remove_header` 移去原数据包已被发送的 1480 位数据  
 继续使用 `while` 进行条件判断，对大于 1480 的数据包进行相同的操作，每次都将在偏移量自增 185，直到剩余数据包不足 1480 长  
 若剩余数据包长度仍然大于 0，则将剩余数据拷入最后一个申请的数据包中，将其发送，标志位为 0

在讲数据包发送完成后，将全局变量 `idcnt` 自增 1

### `ip_fragment_out()` 函数

首先使用 `buf_add_header` 函数为数据报添加 20 字节的 IP 报头长度，同样使用 `ip_hdr_t` 读取和填充数据报头的内容，分别填入长度 5，版本号 4，`tos` 为 0，有效数据大小（小端存储，需要翻转），数据报标识符（小端存储，需要翻转），标志与分段拼接在一起后，也翻转存入，最后 `tll`、上层协议号，用 `memcpy` 拷贝目的 `ip` 和源 `ip`，校验和位置 0 后调用 `checksum16` 将 `hdr` 头转为 16 位无符号数指针传入，计算 20 位头部的校验和并传入，最后调用 `aro_out` 向下层发送报文

### `checksum16` 实现：

观察到传入的一个 16 位无符号 `int` 的指针，天然的是一个半字，但是代码框架中的基本 `len` 是以字节为单位的，所以使用 `while` 循环时（`len>1`），`data` 指针++（指向数据中下一个半字），`len` 需要自减 2，这样就将要检测报文的内容以 2 字节为单位相加了

自减 2 的结果 `len` 可能余下 1，也就是最后 1 字节（8 位）的数据。这时候再将 `data` 指针转成 8 位无符号 `int` 的指针类型，把这最后 8 个 `bit` 值加上

取出相加结果的高十六位（通过左移结果），判断其是否为 0，不为 0 则将其加上 `res` 的低 16 位（与上 `0x0000ffff` 以取得低位），直到取出的高位为 0

将最后的结果存为 16 位无符号 `int`，取反后再返回

## 2. ICMP 协议详细设计

### `icmp_resp` 函数：

在观察 `icmp` 响应报文的结构后发现数据部分和头部的部分参数取自请求报文，所以首先使用指针取出请求包的头部，再去掉请求报文的头部 8 位，将其 `data` 字段拷贝入 `txbuf` 的 `data` 字段中

为 `txbuf` 延长出 8 位的头部，填入类型 0 和代码 0，标识符和序列号采用之前取出的头部的数据，最后将校验和置 0 并计算新校验和，使用 `ip_out` 发送报文。

### `icmp_in` 函数：

首先判断收到 `buf` 包的长度，若小于 9 位则丢弃报文

用指针取出报头数据，只判断 `type` 字段是否为请求报文类型，如果是，则调用 `icmp_resp` 回复相应报文

### `icmp_unreachable` 函数：

观察差错报文的结构，发现要先填入数据段才行，指导书先填入头部的指导我不知道是不是不妥的，但是从框架代码实现来看，应该是先拷贝入数据段，也就是原缓冲区的前 28 位。再延长 8 位报文头部。填入类型 3 以及 `code`，将标识和序列号以及校验和置 0，再计算校验和并填入，调用 `ip_out` 发送报文

### 3. UDP 协议详细设计

udp\_checksum 函数:

首先使用一个指针将原本 udp 头中的数据取出, 便于填入伪头部的长度信息  
计算实际长度是需要翻转的, 但是填入时则可以直接填入。接着将报文头前移 20 个字节, 用 memcpy 拷贝出 ip 报头, 再回移留出伪头部的的位置并填入相关信息。  
最后调用已经在 ip 协议编写好的生成校验码, 其中长度为原长度+12。  
去掉伪头部, 用 memcpy 拷贝回原 ip 头部, 最后返回校验和即可。

udp\_in 函数:

首先判断报文长度是否小于 8 位, 丢弃长度不足的报文  
取出头部的校验和, 并重新计算校验和, 丢弃校验失败的报文  
将目的端口号翻转后, 得到正确的端口号数据, 调用 map\_get 在 udp\_table 中寻找合适的 udp\_handler, 其中端口传入的是端口号地址转 void 类型指针。  
判断拿到的 handler 是否为空, 如果为空则还原 ip 报头, 调用 icmp 不可达  
如果不为空, 则调用 udp\_open 打开端口功能, 然后解引用 handler 并填入参数, 调用回调函数。

udp\_out 函数:

增加 udp 头, 所有 16 位的数值都需要翻转后填入。最后计算校验和并使用 ip\_out 发送报文

## 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

```
PS E:\Network\lab4\net-lab-master\build> ctest -R ip_test
Test project E:/Network/lab4/net-lab-master/build
  Start 4: ip_test
1/1 Test #4: ip_test ..... Passed    0.06 sec

100% tests passed, 0 tests failed out of 1

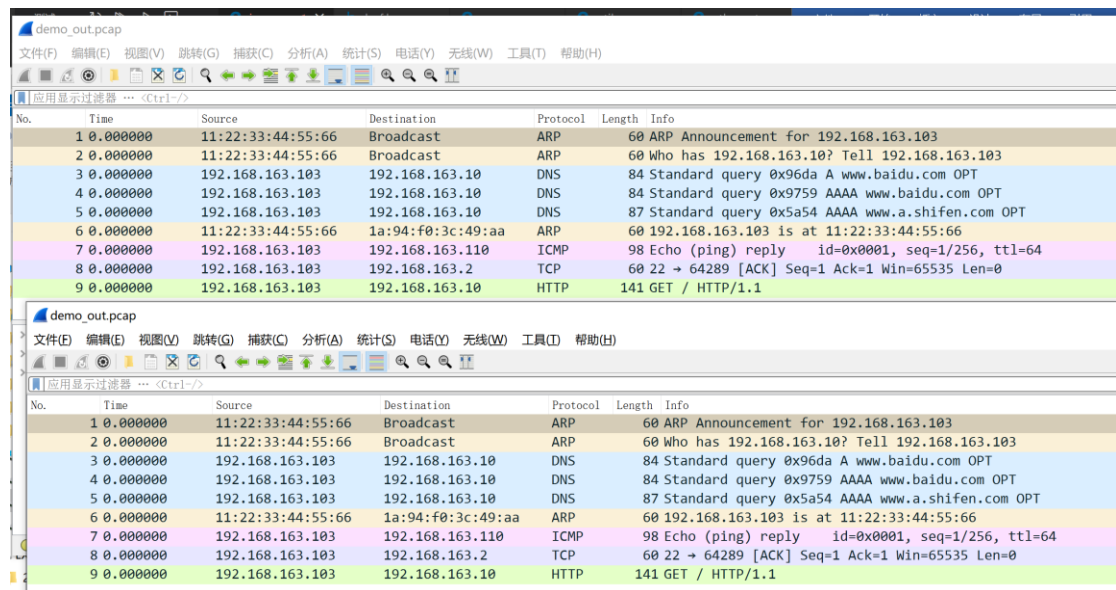
Total Test time (real) =  0.12 sec
PS E:\Network\lab4\net-lab-master\build>
```

```
PS E:\Network\lab4\net-lab-master\build> ctest -R ip_frag_test
Test project E:/Network/lab4/net-lab-master/build
  Start 5: ip_frag_test
1/1 Test #5: ip_frag_test ..... Passed    0.04 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =  0.09 sec
PS E:\Network\lab4\net-lab-master\build>
```

通过观察 out.pcap 可以看到 2ip 报文



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	11:22:33:44:55:66	Broadcast	ARP	60	ARP Announcement for 192.168.163.103
2	0.000000	11:22:33:44:55:66	Broadcast	ARP	60	Who has 192.168.163.10? Tell 192.168.163.103
3	0.000000	192.168.163.103	192.168.163.10	DNS	84	Standard query 0x96da A www.baidu.com OPT
4	0.000000	192.168.163.103	192.168.163.10	DNS	84	Standard query 0x9759 AAAA www.baidu.com OPT
5	0.000000	192.168.163.103	192.168.163.10	DNS	87	Standard query 0x5a54 AAAA www.a.shifen.com OPT
6	0.000000	11:22:33:44:55:66	1a:94:f0:3c:49:aa	ARP	60	192.168.163.103 is at 11:22:33:44:55:66
7	0.000000	192.168.163.103	192.168.163.110	ICMP	98	Echo (ping) reply id=0x0001, seq=1/256, ttl=64
8	0.000000	192.168.163.103	192.168.163.2	TCP	60	22 → 64289 [ACK] Seq=1 Ack=1 Win=65535 Len=0
9	0.000000	192.168.163.103	192.168.163.10	HTTP	141	GET / HTTP/1.1

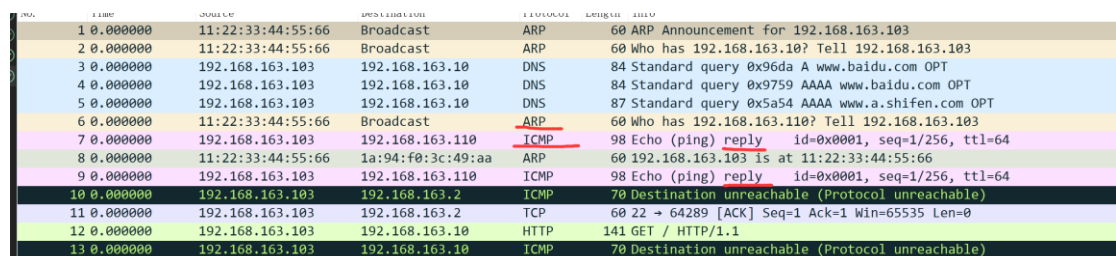
通过对比发现，各个 ipv4 报文的内容和范例都是相同的

## 2. ICMP 协议实验结果及分析

```
PS E:\Network\lab4\net-lab-master> cd .\build\
PS E:\Network\lab4\net-lab-master\build> ctest -R icmp_test
Test project E:/Network/lab4/net-lab-master/build
  Start 6: icmp_test
1/1 Test #6: icmp_test ..... Passed    0.06 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =  0.12 sec
PS E:\Network\lab4\net-lab-master\build>
```

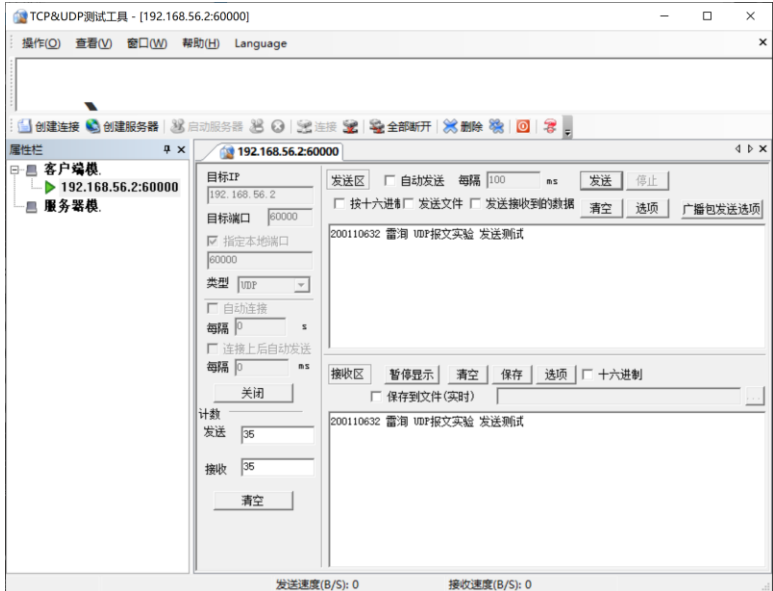


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	11:22:33:44:55:66	Broadcast	ARP	60	ARP Announcement for 192.168.163.103
2	0.000000	11:22:33:44:55:66	Broadcast	ARP	60	Who has 192.168.163.10? Tell 192.168.163.103
3	0.000000	192.168.163.103	192.168.163.10	DNS	84	Standard query 0x96da A www.baidu.com OPT
4	0.000000	192.168.163.103	192.168.163.10	DNS	84	Standard query 0x9759 AAAA www.baidu.com OPT
5	0.000000	192.168.163.103	192.168.163.10	DNS	87	Standard query 0x5a54 AAAA www.a.shifen.com OPT
6	0.000000	11:22:33:44:55:66	Broadcast	ARP	60	Who has 192.168.163.110? Tell 192.168.163.103
7	0.000000	192.168.163.103	192.168.163.110	ICMP	98	Echo (ping) reply id=0x0001, seq=1/256, ttl=64
8	0.000000	11:22:33:44:55:66	1a:94:f0:3c:49:aa	ARP	60	192.168.163.103 is at 11:22:33:44:55:66
9	0.000000	192.168.163.103	192.168.163.110	ICMP	98	Echo (ping) reply id=0x0001, seq=1/256, ttl=64
10	0.000000	192.168.163.103	192.168.163.2	ICMP	70	Destination unreachable (Protocol unreachable)
11	0.000000	192.168.163.103	192.168.163.2	TCP	60	22 → 64289 [ACK] Seq=1 Ack=1 Win=65535 Len=0
12	0.000000	192.168.163.103	192.168.163.10	HTTP	141	GET / HTTP/1.1
13	0.000000	192.168.163.103	192.168.163.10	ICMP	70	Destination unreachable (Protocol unreachable)

有 2 个 ping 相应报文，对应 in 文件中的一个 ping 请求，还有 2 个协议不可达报文

### 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)



```
PS E:\NetWork\lab4\net-lab-master\build> . "E:\Network\lab4\net-lab-master\build\main.exe"
Using interface \Device\NPF_{5F233060-71BD-4490-9770-82410CBF3DEC}, my ip is 192.168.56.2.
tcp open
tcp open
recv udp packet from 192.168.56.1:60000 len=35
200110632 雷洵 UDP报文实验 发送测试
```

正在捕获 VirtualBox Host-Only Network

No.	Time	Source	Destination	Protocol	Length	Info
5	57.932519	CIMSYS_33:44:55	Broadcast	ARP	60	ARP Announcement for 192.168.56.2
6	67.387280	192.168.56.1	192.168.56.2	UDP	77	60000 → 60000 Len=35
7	67.397629	CIMSYS_33:44:55	Broadcast	ARP	60	Who has 192.168.56.1? Tell 192.168.56.2
8	67.397649	0a:00:27:00:00:07	CIMSYS_33:44:55	ARP	42	192.168.56.1 is at 0a:00:27:00:00:07
9	67.410785	192.168.56.2	192.168.56.1	UDP	77	60000 → 60000 Len=35
10	71.953094	0a:00:27:00:00:07	CIMSYS_33:44:55	ARP	42	Who has 192.168.56.2? Tell 192.168.56.1
11	71.953143	CIMSYS_33:44:55	0a:00:27:00:00:07	ARP	60	192.168.56.2 is at 00:11:22:33:44:55

第一条 ARP 报文，是广播报文，初次运行 main.exe 时，arp 协议发送的广播，声明自己的地址

第二条报文是通过测试工具发送的报文，

- User Datagram Protocol, Src Port: 60000, Dst Port: 60000
  - Source Port: 60000
  - Destination Port: 60000
  - Length: 43
  - Checksum: 0xce63 [unverified]
  - [Checksum Status: Unverified]
  - [Stream index: 1]
  - [Timestamps]
  - UDP payload (35 bytes)

可以看到是向端口 6000 发送的

第三条报文是 arp 询问报文，main 收到了来自 192.168.56.1 的报文，调用了回复相同内容的函数，要发送回 192.168.56.1，但是 arp 表中没有该地址的信息，故发送广播进行查询。

第四条报文则是回复报文，从虚拟网卡发送往报文来源，测试程序由此接收到了相同内容的回复报文

第五条和第六条也是 arp 报文，该 mac 地址的设备询问 192.168.56.2 的 mac 地址，并得到了回复。

### 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）*

#### *在写 IP 协议遇到的问题*

在设计 ip 时，在 out.pcap 中发现发送报文的校验码总是不正确，多次尝试修改校验和函数后，发现输出仍然错误，再仔细对比了其他位，发现有 2 个 16 位数据（id 和总长）与 demo 中的范例输出正好相反，原来是没有将其小端存储，在修改相关代码调用 swap16 后，成功通过了第一个 ip 测试。

但是 ip 分片的测试依然没有通过，检查 log 后发现输出的数据大大长于范例输出，在仔细检查代码分片逻辑后，发现要拷贝进 data 字段的是数据包的数据段，而不是对整个 buf 的指针使用 memcpy，在修改后，能够通过第二个 ip 测试。

#### *在写 ICMP 协议遇到的问题*

在做 ICMP 实现时，首先发现不可达报文的数据段与 demo 不同，在检查代码后，发现要再 ip\_in 函数处还原头部字段

继续查看 out.pcap 时，发现有个 ARP 请求报文，请求的是一个 16 进制形似 mac 的东西，而 ping 回应则比 demo 少了一条对应地址的。

在打了大量断点调试后，发现问题出现在 icmp\_in 函数，其传入的 src\_ip 地址是错误的，原本还以为是测试用例错误，但是继续大量断点调试后，我最终把问题集中在 net\_in 上

查看注释发现 net\_in 的参数可以是 mac 地址也可以是 ip 地址，继续调试以及加断点，发现在跳入 icmp\_in 函数之前最后的一个 net\_in 函数是在 ip\_in 之中被调用的。终于找到了问题症结。由于没有判断协议类型而默认将 mac 地址作为源地址传入了，所以 icmp\_in 将错误的 ip 地址（实际为 mac 地址的前几位）发出，而表中没有该 ip，自然发出 arp 请求，又由于得不到回应，找不到正确发出的地址，所以少了一条 ping 响应报文。在修改代码，增加判断语句后就能跑过测试了。

#### *在写 UDP 协议遇到的问题*

由于 UDP 实验并没有自动化的测试脚本可以跑，所以我选择在 main 函数运行时加入一些打



印函数和断点调试来查看程序究竟调用了哪些函数，参数有没有出错。在编写校验和时，通过仔细调试对比相关数据，终于得到了正确的结果。

对于如何调用函数，在之前 debug 的时候观察到了 `net_in` 是采用 `handler` 来实现回调的，在观察了 `udp.h` 和 `udp.c` 的代码后，发现也有类似的构造，结果上其实是最终调用了消息回复函数。

## 四、实验收获和建议

*（实验过程中的收获、感受、问题、建议等）*

感觉自己 debug 能力又变得更强了。在做 `icmp` 的调试时，对源框架代码又有了全新的认识。对网络通信的各项逻辑有了更加深刻的理解。这部分代码量不大，但是却涉及整个代码框架以及网络通信的各个环节，需要对自己的代码实现以及代码框架都有更深入的理解，才不容易出错。

网络协议需要逻辑严谨，每一层的逻辑都必须自洽且不能出错，否则就会层层影响，导致查错时间大大增加。

这次的框架代码个人是十分喜欢的，可读性也比较强，学到了很多，谢谢老师和助教！