

基于单网口主机的 IP 数据报转发及收发

1 静态 IP 配置

1.1 网关查找

如果你尝试了多个可能的网关地址（如 `192.168.234.2` 和 `192.168.222.1`）但都无法连通，那么可以通过以下方法来找到正确的网关地址。

检查虚拟机网络配置

确保你的虚拟机网络配置正确，以下是一些检查步骤：

- **确认网络模式**：如果使用 VirtualBox 或 VMware，确保虚拟机的网络模式为 NAT 或桥接模式（根据你的网络需求来设置）。
 - **NAT 模式**通常分配的 IP 地址和网关在一个私有网络范围内，如 `192.168.x.x`。
 - **桥接模式**则会使虚拟机的 IP 地址在局域网范围内，并共享主机的网关。

查找当前网络的路由表

在 Debian 虚拟机上使用 `ip route` 命令查看路由表，其中默认网关会标记为 `default via`：

```
1 | ip route
```

如果配置正确，输出可能类似于：

```
1 | default via 192.168.x.x dev ens33
2 | 192.168.x.0/24 dev ens33 proto kernel scope link src 192.168.x.x
```

- **default via 192.168.x.x** 就是系统默认的网关。
- 如果没有 `default via` 条目，说明目前系统未设置网关，可能是网络接口配置不正确。

使用 `route` 命令

`route` 命令也可以查看当前路由信息：

```
1 | route -n
```

在输出中，`Gateway` 列中与 `0.0.0.0` 对应的 IP 地址即是默认网关地址。

```
aircraft@root:~$ ip route
default via 192.168.222.2 dev ens33
192.168.222.0/24 dev ens33 proto kernel scope link src 192.168.222.131
aircraft@root:~$ ping -c 4 192.168.222.2
PING 192.168.222.2 (192.168.222.2) 56(84) bytes of data.
64 bytes from 192.168.222.2: icmp_seq=1 ttl=128 time=0.747 ms
64 bytes from 192.168.222.2: icmp_seq=2 ttl=128 time=0.197 ms
64 bytes from 192.168.222.2: icmp_seq=3 ttl=128 time=0.199 ms
64 bytes from 192.168.222.2: icmp_seq=4 ttl=128 time=0.247 ms

--- 192.168.222.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3059ms
rtt min/avg/max/mdev = 0.197/0.347/0.747/0.231 ms
```

1.2 静态 IP 配置

好的，以下是如何配置 `ens33` 接口为静态 IP `192.168.222.131/24` 的方法。

将以下内容添加到 `/etc/network/interfaces` 文件中：

```
1 auto ens33
2 iface ens33 inet static
3     address 192.168.222.131
4     netmask 255.255.255.0
5     gateway 192.168.222.2
6     dns-nameservers 8.8.8.8 8.8.4.4
```

配置解释

- `auto ens33`：设置 `ens33` 接口在启动时自动激活。
- `iface ens33 inet static`：指定 `ens33` 接口使用静态 IP 配置。
- `address 192.168.222.131`：指定静态 IP 地址。
- `netmask 255.255.255.0`：指定子网掩码，这里是 `/24`。
- `gateway 192.168.222.2`：设置网关地址（需确认虚拟网络的网关，通常是 `192.168.222.1`，你可以在主机上查看网关或在虚拟机里测试）。
- `dns-nameservers 8.8.8.8 8.8.4.4`：设置 DNS 服务器，这里用的是 Google 的公共 DNS。

操作步骤

1. 编辑配置文件：

- 打开 `/etc/network/interfaces` 文件：

```
1 | sudo vim /etc/network/interfaces
```

2. 添加配置：将上述内容添加到文件中。

3. 重启网络服务：

- 保存并退出文件后，重启网络服务以应用新的配置：

```
1 | sudo systemctl restart networking
```

或者重启虚拟机：

```
1 | sudo reboot
```

验证配置

重启网络后，可以使用以下命令来检查 IP 是否正确配置：

```
1 | ip addr show ens33
```

如果配置正确，你应该能看到 `ens33` 接口显示为 `192.168.222.131/24`。

1.3 ssh 登入

根据你提供的输出信息，存在两个问题：

1. **SSH 密钥已经存在**：你尝试生成的 SSH 密钥对 `C:\Users\Lenovo\.ssh\id_rsa` 已经存在，因此系统询问你是否要覆盖它。
2. **ssh-copy-id 命令不可用**：在 Windows 的默认命令行中，`ssh-copy-id` 不是内置命令。

1.3.1 生成密钥

```
1 | ssh-keygen -t rsa -b 4096
```

如果你不再需要现有的密钥，可以选择覆盖它。输入 `y` 以确认覆盖。如果你希望保留现有密钥，可以选择另一个文件名：

```
1 | Enter file in which to save the key (C:\Users\Lenovo\.ssh\id_rsa):  
C:\Users\Lenovo\.ssh\id_rsa_new
```

这将生成一个新的密钥对。

1.3.2 2. 将公钥复制到远程主机

由于 `ssh-copy-id` 在 Windows 中不可用，你需要手动将公钥复制到远程主机。请按照以下步骤操作：

1. 显示公钥内容：

```
1 | type C:\Users\Lenovo\.ssh\id_rsa.pub
```

或者如果你生成了新密钥（如 `id_rsa_new.pub`），请替换文件名：

```
1 | type C:\Users\Lenovo\.ssh\id_rsa_new.pub
```

```
1 | ssh-rsa  
AAAAB3NzaC1yc2EAAAADAQABAAQDIsQxoyNqrJrO2RuQSmII3sXlMGzA7wsEsIhukUMNELhqULW/JDBN  
ZyryR4LtwONMYXu/F65zwa3IADqZuMrm/0lwAsiGyYcbrAe5Q6f1J9WuoNgn19+VDn+/z/6Ma+mgRzXhm8Z  
B+LMsxjBZ2tfS8GC+vcVQmUBW4037G4ik5k1Mjah8NAiIlvbpsGPB1QmbujhU1l6kiWn3tseyPecMVQmkKA  
DE96AY1KMZ6jERQeX8Eh2sM1TwAZ+Tdd0IWtqLvo8DTxNcA2m1Dgw2WkN2EfbDr3+OoqPQFHZU4EjG6tvEa  
hWKGUBhG0xCJr5j40GYUjynCRDMu2kAMGXB0TLNSRxKpV4LCa5vb2bTAN0/v3cjLIMgsCkDbnp6KYYjrTLd  
bubf44YC73APXPfevx8mQs3cbzmb8L1rX366ET4QV0T0TIwWjybbR23sbQU0tUsTfNKpmxkz9cWk8eUQQdq  
3sQVqN8dDB3JI0xUx1JP9F3ozjEjI+t50smSf5hz4E107W0B6+1Uu5WmfKHJ8nCAyqjCptEUS6a0kXeBsVo  
Em6xc0Uebs9x1U0dVA5/Ycd/yPCn6YXe9j7Hg3A1oxYc2dpkyu0fGMR4n1sYq9sK59VTkCYxxjyj0D1VzIM  
Fbu81xu6n69Gis3/Z3PNGqhCnK2otSRW+xMcItoPzGtzsXasVw== 2451752823@qq.com
```

2. 登录到远程主机（使用密码登录）：

```
1 | ssh aircraft@192.168.222.131
```

3. 创建或编辑 `~/.ssh/authorized_keys` 文件:

```
1 mkdir -p ~/.ssh
2 vim ~/.ssh/authorized_keys
```

如果 `nano` 不可用, 你可以使用 `vi` 或其他文本编辑器。确保将公钥粘贴到此文件中, 然后保存并退出编辑器。

4. 确保 `~/.ssh` 和 `authorized_keys` 文件的权限正确:

```
1 chmod 700 ~/.ssh
2 chmod 600 ~/.ssh/authorized_keys
```

如果无法通过 SSH 进入远程主机, 可以通过其他方式 (如直接访问虚拟机或使用另一个 SSH 会话) 来执行这些命令。

2 查找MAC地址

```
1 ip link show
```

3 三主机示例程序

3.1 发送主机

为了将 `#define` 常量放在一个配置文件中并在C程序中读取, 你可以使用以下步骤:

1. 创建一个配置文件: 首先创建一个文本文件 (例如 `config.txt`), 在其中定义你的常量。例如:

```
1 UDP_SRC_PORT=12345
2 UDP_DST_PORT=12345
3 DEST_MAC=00:0c:29:3e:1e:4c
```

2. 读取配置文件: 在C程序中使用 `fgets()` 函数逐行读取文件内容, 并使用 `sscanf()` 或 `strtok()` 等函数解析出具体的值。你需要在程序中动态定义这些变量。

3. 修改C代码: 需要包含必要的头文件并添加代码用于读取配置文件。

以下是具体的实现示例:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6 #include <sys/socket.h>
7 #include <netinet/ip.h>
8 #include <netinet/udp.h>
9 #include <netinet/ether.h>
10 #include <net/if.h>
11 #include <sys/ioctl.h>
12 #include <linux/if_packet.h>
13
14 #define BUFFER_SIZE 1518 // 以太网帧最大长度
15
16 // 全局变量
17 int UDP_SRC_PORT;
```

```

18 int UDP_DST_PORT;
19 unsigned char DEST_MAC[6];
20
21 void load_config(const char *filename) {
22     FILE *file = fopen(filename, "r");
23     if (!file) {
24         perror("Could not open config file");
25         exit(EXIT_FAILURE);
26     }
27
28     char line[256];
29     while (fgets(line, sizeof(line), file)) {
30         // 去掉换行符
31         line[strcspn(line, "\n")] = 0;
32
33         // 根据等号分割键值对
34         char key[32];
35         char value[32];
36         if (sscanf(line, "%[^]=%s", key, value) == 2) {
37             if (strcmp(key, "UDP_SRC_PORT") == 0) {
38                 UDP_SRC_PORT = atoi(value);
39             } else if (strcmp(key, "UDP_DST_PORT") == 0) {
40                 UDP_DST_PORT = atoi(value);
41             } else if (strcmp(key, "DEST_MAC") == 0) {
42                 sscanf(value, "%hhx:%hhx:%hhx:%hhx:%hhx:%hhx",
43                     &DEST_MAC[0], &DEST_MAC[1], &DEST_MAC[2],
44                     &DEST_MAC[3], &DEST_MAC[4], &DEST_MAC[5]);
45             }
46         }
47     }
48
49     fclose(file);
50 }
51
52 int main() {
53     // 先加载配置文件
54     load_config("config.txt");
55
56     // 后续代码（例如创建套接字、构造数据包等）...
57     int sockfd; // 套接字文件描述符
58     struct ifreq if_idx, if_mac; // 用于获取接口索引和MAC地址
59     struct sockaddr_ll socket_address; // 套接字地址结构
60     char buffer[BUFFER_SIZE]; // 数据缓冲区
61
62     // 创建原始套接字，允许捕获所有以太网数据
63     if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) == -1) {
64         perror("socket");
65         return 1; // 出现错误，退出
66     }
67
68     // ... 省略获取接口索引和MAC地址的代码
69
70     // 构造以太网头
71     struct ether_header *eh = (struct ether_header *) buffer;
72     // 用获取到的MAC地址设置源地址
73     memcpy(eh->ether_shost, if_mac.ifr_hwaddr.sa_data, ETH_ALEN);

```

```

74     memcpy(eh->ether_dhost, DEST_MAC, ETH_ALLEN); // 目标MAC地址
75     eh->ether_type = htons(0x0800); // 设置以太网类型字段 (0x0800表示IP协议)
76
77     // 后面的代码与之前相同
78
79     // 发送数据包
80     if (sendto(sockfd, buffer, sizeof(struct ether_header) + sizeof(struct iphdr) +
sizeof(struct udphdr) + strlen("Hello, this is a test message."), 0, (struct
sockaddr*)&socket_address, sizeof(struct sockaddr_ll)) < 0) {
81         perror("sendto"); // 发送失败
82         return 1;
83     }
84
85     close(sockfd); // 关闭套接字
86     return 0; // 正常结束
87
88 }

```

• 发送端代码

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <arpa/inet.h>
6  #include <sys/socket.h>
7  #include <netinet/ip.h>
8  #include <netinet/udp.h>
9  #include <netinet/ether.h>
10 #include <net/if.h>
11 #include <sys/ioctl.h>
12 #include <linux/if_packet.h>
13
14 // 定义目标MAC地址
15 #define DEST_MAC0 0x00
16 #define DEST_MAC1 0x0c
17 #define DEST_MAC2 0x29
18 #define DEST_MAC3 0x3e
19 #define DEST_MAC4 0x1e
20 #define DEST_MAC5 0x4c
21
22 #define ETHER_TYPE 0x0800 // 以太网类型字段 (0x0800表示IP协议)
23 #define BUFFER_SIZE 1518 // 以太网帧最大长度
24 #define UDP_SRC_PORT 12345 // 源UDP端口
25 #define UDP_DST_PORT 12345 // 目的UDP端口
26
27 // 计算校验和的函数
28 unsigned short checksum(void *b, int len) {
29     unsigned short *buf = b;
30     unsigned int sum = 0;
31     unsigned short result;
32
33     // 对每两个字节进行求和
34     for (sum = 0; len > 1; len -= 2)
35         sum += *buf++;
36     // 如果有剩下的字节, 加上它

```

```

37     if (len == 1)
38         sum += *(unsigned char *)buf;
39     // 处理溢出
40     sum = (sum >> 16) + (sum & 0xFFFF);
41     sum += (sum >> 16);
42     result = ~sum; // 取反作为校验和
43     return result;
44 }
45
46 int main() {
47     int sockfd; // 套接字文件描述符
48     struct ifreq if_idx, if_mac; // 用于获取接口索引和MAC地址
49     struct sockaddr_ll socket_address; // 套接字地址结构
50     char buffer[BUFFER_SIZE]; // 数据缓冲区
51
52     // 创建原始套接字, 允许捕获所有以太网数据
53     if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) == -1) {
54         perror("socket");
55         return 1; // 出现错误, 退出
56     }
57
58     // 获取接口索引
59     memset(&if_idx, 0, sizeof(struct ifreq));
60     strncpy(if_idx.ifr_name, "eth0", IFNAMSIZ-1); // eth0 为目标网络接口名称
61     if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0) {
62         perror("SIOCGIFINDEX"); // 获取索引失败
63         return 1;
64     }
65
66     // 获取接口 MAC 地址
67     memset(&if_mac, 0, sizeof(struct ifreq));
68     strncpy(if_mac.ifr_name, "eth0", IFNAMSIZ-1);
69     if (ioctl(sockfd, SIOCGIFHWADDR, &if_mac) < 0) {
70         perror("SIOCGIFHWADDR"); // 获取MAC地址失败
71         return 1;
72     }
73
74     // 构造以太网头
75     struct ether_header *eh = (struct ether_header *) buffer;
76     memcpy(eh->ether_shost, if_mac.ifr_hwaddr.sa_data, ETH_ALEN); // 设置源MAC地址
77     eh->ether_dhost[0] = DEST_MAC0; // 设置目标MAC地址
78     eh->ether_dhost[1] = DEST_MAC1;
79     eh->ether_dhost[2] = DEST_MAC2;
80     eh->ether_dhost[3] = DEST_MAC3;
81     eh->ether_dhost[4] = DEST_MAC4;
82     eh->ether_dhost[5] = DEST_MAC5;
83     eh->ether_type = htons(ETHER_TYPE); // 设置以太网类型字段
84
85     // 构造 IP 头
86     struct iphdr *iph = (struct iphdr *) (buffer + sizeof(struct ether_header));
87     iph->ihl = 5; // IP头长度
88     iph->version = 4; // IPv4
89     iph->tos = 0; // 服务类型
90     iph->tot_len = htons(sizeof(struct iphdr) + sizeof(struct udphdr) +
91     strlen("Hello, this is a test message.")); // 总长度
92     iph->id = htonl(54321); // 标识符

```

```

92     iph->frag_off = 0; // 不分片
93     iph->ttl = 255; // 生存时间
94     iph->protocol = IPPROTO_UDP; // 协议类型为UDP
95     iph->check = 0; // 校验和开始为0
96     iph->saddr = inet_addr("192.168.1.2"); // 源IP地址
97     iph->daddr = inet_addr("192.168.1.3"); // 目的IP地址
98     iph->check = checksum((unsigned short *)iph, sizeof(struct iphdr)); // 计算并设置
    校验和
99
100    // 构造 UDP 头
101    struct udphdr *udph = (struct udphdr *) (buffer + sizeof(struct ether_header) +
    sizeof(struct iphdr));
102    udph->source = htons(UDP_SRC_PORT); // 源端口
103    udph->dest = htons(UDP_DST_PORT); // 目的端口
104    udph->len = htons(sizeof(struct udphdr) + strlen("Hello, this is a test
    message.")); // UDP报文长度
105    udph->check = 0; // UDP 校验和可选
106
107    // 填充数据内容
108    char *data = (char *) (buffer + sizeof(struct ether_header) + sizeof(struct
    iphdr) + sizeof(struct udphdr));
109    strcpy(data, "Hello, this is a test message."); // 数据内容
110
111    // 设置 socket 地址结构
112    socket_address.sll_ifindex = if_idx.ifr_ifindex; // 设置接口索引
113    socket_address.sll_halen = ETH_ALEN; // MAC地址长度
114    memcpy(socket_address.sll_addr, eh->ether_dhost, ETH_ALEN); // 设置目标MAC地址
115
116    // 发送数据包
117    if (sendto(sockfd, buffer, sizeof(struct ether_header) + sizeof(struct iphdr) +
    sizeof(struct udphdr) + strlen("Hello, this is a test message."), 0, (struct
    sockaddr*)&socket_address, sizeof(struct sockaddr_ll)) < 0) {
118        perror("sendto"); // 发送失败
119        return 1;
120    }
121
122    close(sockfd); // 关闭套接字
123    return 0; // 正常结束
124 }

```

• 打印数据

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <arpa/inet.h>
6  #include <sys/socket.h>
7  #include <netinet/ip.h>
8  #include <netinet/udp.h>
9  #include <netinet/ether.h>
10 #include <net/if.h>
11 #include <sys/ioctl.h>
12 #include <linux/if_packet.h>
13 #include <time.h>

```



```

14
15 #define BUFFER_SIZE 1518 // 以太网帧最大长度
16
17 void print_packet_info(const unsigned char *buffer, int size) {
18     struct ether_header *eh = (struct ether_header *) buffer; // 以太网头
19     struct iphdr *iph = (struct iphdr *) (buffer + sizeof(struct ether_header)); // IP
    头
20
21     // 获取当前时间
22     time_t now;
23     time(&now);
24     char *time_str = ctime(&now); // 获取当前时间
25     time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符
26
27     // 打印信息
28     printf("Received Packet:\n");
29     printf("Time: %s\n", time_str);
30     printf("Source MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh->ether_shost));
31     printf("Destination MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh->ether_dhost));
32     printf("Source IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->saddr));
33     printf("Destination IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->daddr));
34
35     printf("TTL: %d\n", iph->ttl);
36     printf("-----\n");
37 }
38
39 int main() {
40     int sockfd; // 套接字文件描述符
41     struct sockaddr_saddr; // 套接字地址结构
42     unsigned char *buffer = (unsigned char *) malloc(BUFFER_SIZE); // 数据缓冲区
43     socklen_t saddr_len = sizeof(saddr);
44
45     // 创建原始套接字, 允许捕获所有以太网数据
46     if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) == -1) {
47         perror("Socket creation failed");
48         return 1; // 出现错误, 退出
49     }
50
51     // 捕获数据包
52     while (1) {
53         int data_size = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, &saddr, &saddr_len);
54
55         if (data_size < 0) {
56             perror("Recvfrom error");
57             return 1; // 出现错误, 退出
58         }
59
60         // 解析并打印数据包信息
61         print_packet_info(buffer, data_size);
62     }
63
64     close(sockfd); // 关闭套接字
65     free(buffer); // 释放缓冲区
66     return 0; // 正常结束

```

3.2 说明

- **配置文件**: 在配置文件中定义的键值对可以灵活修改。
- **load_config函数**: 打开配置文件并逐行读取, 解析出所需的值并赋值给全局变量。
- **MAC地址处理**: 读取并解析MAC地址字符串, 存储到 `unsigned char DEST_MAC[6]` 数组中。
- **动态变量**: 这些变量代替了原来的 `#define` 常量, 因此可以在运行时根据配置文件的内容进行调整。

通过这种方法, 你可以更灵活地管理配置信息, 而无需在源代码中硬编码常量。

这段代码是一个用C语言编写的网络程序, 主要用于通过原始套接字发送一个UDP数据包。以下是代码的详细解释:

3.2.1 变量声明

```
1 int sockfd; // 套接字文件描述符
2 struct ifreq if_idx, if_mac; // 用于获取接口索引和MAC地址
3 struct sockaddr_ll socket_address; // 套接字地址结构
4 char buffer[BUFFER_SIZE]; // 数据缓冲区
```

- **sockfd**: 用于保存创建的套接字的文件描述符。
- **ifreq**: 一个结构体, 用于存储网络接口的信息, 包括索引和MAC地址。
- **sockaddr_ll**: 套接字地址结构, 用于原始套接字的地址。
- **buffer**: 用于存储构造的数据包。

3.2.2 创建原始套接字

```
1 if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) == -1) {
2     perror("socket");
3     return 1; // 出现错误, 退出
4 }
```

- 使用 `socket()` 函数创建一个原始套接字, 允许捕获所有以太网数据包。 `AF_PACKET` 是协议族, `SOCK_RAW` 是套接字类型, `htons(ETH_P_ALL)` 表示捕获所有以太网类型。

3.2.3 获取接口索引

```
1 memset(&if_idx, 0, sizeof(struct ifreq));
2 strncpy(if_idx.ifr_name, "eth0", IFNAMSIZ-1);
3 if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0) {
4     perror("SIOCGIFINDEX"); // 获取索引失败
5     return 1;
6 }
```

- 使用 `ioctl()` 获取指定网络接口 (这里是 `eth0`) 的索引。

3.2.4 获取接口 MAC 地址

```
1 memset(&if_mac, 0, sizeof(struct ifreq));
2 strncpy(if_mac.ifr_name, "eth0", IFNAMSIZ-1);
3 if (ioctl(sockfd, SIOCGIFHWADDR, &if_mac) < 0) {
4     perror("SIOCGIFHWADDR"); // 获取MAC地址失败
5     return 1;
6 }
```

- 再次使用 `ioctl()` 获取指定接口的MAC地址。

3.2.5 构造以太网头

```
1 struct ether_header *eh = (struct ether_header *) buffer;
2 memcpy(eh->ether_shost, if_mac.ifr_hwaddr.sa_data, ETH_ALEN); // 设置源MAC地址
3 eh->ether_dhost[0] = DEST_MAC0; // 设置目标MAC地址
4 // 省略其他目标MAC地址部分
5 eh->ether_type = htons(ETHER_TYPE); // 设置以太网类型字段
```

- 在缓冲区中构造以太网头，设置源MAC地址和目标MAC地址。

3.2.6 构造 IP 头

```
1 struct iphdr *iph = (struct iphdr *) (buffer + sizeof(struct ether_header));
2 iph->ihl = 5; // IP头长度
3 iph->version = 4; // IPv4
4 // 省略其他IP头设置
5 iph->check = checksum((unsigned short *)iph, sizeof(struct iphdr)); // 计算并设置校验和
```

- 在缓冲区中构造IP头，设置源和目的IP地址、协议类型等，并计算校验和。

3.2.7 构造 UDP 头

```
1 struct udphdr *udph = (struct udphdr *) (buffer + sizeof(struct ether_header) +
2 sizeof(struct iphdr));
3 udph->source = htons(UDP_SRC_PORT); // 源端口
4 udph->dest = htons(UDP_DST_PORT); // 目的端口
5 udph->len = htons(sizeof(struct udphdr) + strlen("Hello, this is a test message.)); //
6 // UDP报文长度
7 udph->check = 0; // UDP 校验和可选
```

- 在缓冲区中构造UDP头，设置源和目的端口等。

3.2.8 填充数据内容

```
1 char *data = (char *) (buffer + sizeof(struct ether_header) + sizeof(struct iphdr) +
2 sizeof(struct udphdr));
3 strcpy(data, "Hello, this is a test message."); // 数据内容
```

- 将要发送的消息内容填充到数据部分。

3.2.9 设置 socket 地址结构

```
1 socket_address.sll_ifindex = if_idx.ifr_ifindex; // 设置接口索引
2 socket_address.sll_halen = ETH_ALEN; // MAC地址长度
3 memcpy(socket_address.sll_addr, eh->ether_dhost, ETH_ALEN); // 设置目标MAC地址
```

- 设置要发送数据包的目标MAC地址和接口索引。

3.2.10 发送数据包

```
1 if (sendto(sockfd, buffer, sizeof(struct ether_header) + sizeof(struct iphdr) +
2 sizeof(struct udphdr) + strlen("Hello, this is a test message."), 0, (struct
3 sockaddr*)&socket_address, sizeof(struct sockaddr_ll)) < 0) {
4     perror("sendto"); // 发送失败
5     return 1;
6 }
```

- 使用 `sendto()` 函数发送构造好的数据包。如果发送失败，打印错误信息并退出。

全部代码

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <arpa/inet.h>
6  #include <sys/socket.h>
7  #include <netinet/ip.h>
8  #include <netinet/udp.h>
9  #include <netinet/ether.h>
10 #include <net/if.h>
11 #include <sys/ioctl.h>
12 #include <linux/if_packet.h>
13 #include <time.h>
14
15 #define BUFFER_SIZE 1518 // 以太网帧最大长度
16 #define ETHER_TYPE 0x0800 // 以太网类型字段 (0x0800表示IP协议)
17
18 // 全局变量
19 int UDP_SRC_PORT;
20 int UDP_DST_PORT;
21 char *S_ADDR;
22 char *D_ADDR;
23 unsigned char DEST_MAC[6];
24
25 void load_config(const char *filename) {
26     FILE *file = fopen(filename, "r");
27     if (!file) {
28         perror("Could not open config file");
29         exit(EXIT_FAILURE);
30     }
31
32     char line[256];
33     while (fgets(line, sizeof(line), file)) {
34         // 去掉换行符
35         line[strcspn(line, "\n")] = 0;
36
37         // 根据等号分割键值对
38         char key[32];
39         char value[32];
40         if (sscanf(line, "%[^]=%s", key, value) == 2) {
41             if (strcmp(key, "UDP_SRC_PORT") == 0) {
42                 UDP_SRC_PORT = atoi(value);
43             } else if (strcmp(key, "UDP_DST_PORT") == 0) {
44                 UDP_DST_PORT = atoi(value);
45             } else if (strcmp(key, "DEST_MAC") == 0) {
46                 sscanf(value, "%hhx:%hhx:%hhx:%hhx:%hhx:%hhx",
47                     &DEST_MAC[0], &DEST_MAC[1], &DEST_MAC[2],
48                     &DEST_MAC[3], &DEST_MAC[4], &DEST_MAC[5]);
49             } else if (strcmp(key, "S_ADDR") == 0) {
50                 S_ADDR = strdup(value); // 分配内存并复制源地址
51             } else if (strcmp(key, "D_ADDR") == 0) {
52                 D_ADDR = strdup(value); // 分配内存并复制目的地址
53             }
54         }
55     }
56 }

```

```

57     fclose(file);
58 }
59
60 void print_packet_info(const unsigned char *buffer) {
61     struct ether_header *eh = (struct ether_header *)buffer;           // 以
    以太网头
62     struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header)); // IP
    头
63
64     // 获取当前时间
65     time_t now;
66     time(&now);
67     char *time_str = ctime(&now);           // 获取当前时间
68     time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符
69
70     // 打印信息
71     printf("Received Packet:\n");
72     printf("Time: %s\n", time_str);
73     printf("Source MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh-
    >ether_shost));
74     printf("Destination MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh-
    >ether_dhost));
75     printf("Source IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->saddr));
76     printf("Destination IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph-
    >daddr));
77     printf("TTL: %d\n", iph->ttl);
78     printf("-----\n");
79 }
80
81 // 计算校验和的函数
82 unsigned short checksum(void *b, int len) {
83     unsigned short *buf = b;
84     unsigned int sum = 0;
85     unsigned short result;
86
87     // 对每两个字节进行求和
88     for (sum = 0; len > 1; len -= 2)
89         sum += *buf++;
90     // 如果有剩下的字节，加上它
91     if (len == 1)
92         sum += *(unsigned char *)buf;
93     // 处理溢出
94     sum = (sum >> 16) + (sum & 0xFFFF);
95     sum += (sum >> 16);
96     result = ~sum; // 取反作为校验和
97     return result;
98 }
99
100 int main() {
101     // 先加载配置文件
102     load_config("config.txt");
103
104     // 后续代码（例如创建套接字、构造数据包等）...
105     int sockfd;           // 套接字文件描述符
106     struct ifreq if_idx, if_mac; // 用于获取接口索引和MAC地址
107     struct sockaddr_ll socket_address; // 套接字地址结构

```

```

108     char buffer[BUFFER_SIZE];           // 数据缓冲区
109
110     // 创建原始套接字, 允许捕获所有以太网数据
111     if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) == -1) {
112         perror("socket");
113         return 1; // 出现错误, 退出
114     }
115
116     // 获取接口索引
117     memset(&if_idx, 0, sizeof(struct ifreq));
118     strncpy(if_idx.ifr_name, "eth0", IFNAMSIZ - 1); // eth0 为目标网络接口名称
119     if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0) {
120         perror("SIOCGIFINDEX"); // 获取索引失败
121         return 1;
122     }
123
124     // 获取接口 MAC 地址
125     memset(&if_mac, 0, sizeof(struct ifreq));
126     strncpy(if_mac.ifr_name, "eth0", IFNAMSIZ - 1);
127     if (ioctl(sockfd, SIOCGIFHWADDR, &if_mac) < 0) {
128         perror("SIOCGIFHWADDR"); // 获取MAC地址失败
129         return 1;
130     }
131
132     // 构造以太网头
133     struct ether_header *eh = (struct ether_header *)buffer;
134     memcpy(eh->ether_shost, if_mac.ifr_hwaddr.sa_data, ETH_ALEN); // 设置源MAC地址
135     memcpy(eh->ether_dhost, DEST_MAC, ETH_ALEN);                 // 目标MAC地址
136     eh->ether_type = htons(ETHER_TYPE);                           // 设置以太网类型字段
137
138     // 构造 IP 头
139     struct iphdr *iph = (struct iphdr *) (buffer + sizeof(struct ether_header));
140     iph->ihl = 5;
141
142     // IP头长度
143     iph->version = 4;
144
145     // IPv4
146     iph->tos = 0;
147
148     // 服务类型
149     iph->tot_len = htons(sizeof(struct iphdr) + sizeof(struct udphdr) +
150     strlen("Hello, this is a test message.")); // 总长度
151     iph->id = htonl(54321);
152
153     // 标识符
154     iph->frag_off = 0;
155
156     // 不分片
157     iph->ttl = 255;
158
159     // 生存时间
160     iph->protocol = IPPROTO_UDP;
161
162     // 协议类型为UDP
163     iph->check = 0;
164
165     // 校验和开始为0
166     iph->saddr = inet_addr(S_ADDR);
167
168     // 源IP地址
169     iph->daddr = inet_addr(D_ADDR);
170
171     // 目的IP地址
172     iph->check = checksum((unsigned short *)iph, sizeof(struct iphdr));
173
174     // 计算并设置校验和

```

```

152
153     // 构造 UDP 头
154     struct udphdr *udph = (struct udphdr *)(buffer + sizeof(struct ether_header) +
sizeof(struct iphdr));
155     udph->source = htons(UDP_SRC_PORT);
    // 源端口
156     udph->dest = htons(UDP_DST_PORT);
    // 目的端口
157     udph->len = htons(sizeof(struct udphdr) + strlen("Hello, this is a test
message.")); // UDP报文长度
158     udph->check = 0;
    // UDP 校验和可选
159
160     // 填充数据内容
161     char *data = (char *)(buffer + sizeof(struct ether_header) + sizeof(struct iphdr)
+ sizeof(struct udphdr));
162     strcpy(data, "Hello, this is a test message."); // 数据内容
163
164     // 设置 socket 地址结构
165     socket_address.sll_ifindex = if_idx.ifr_ifindex; // 设置接口索引
166     socket_address.sll_halen = ETH_ALEN; // MAC地址长度
167     memcpy(socket_address.sll_addr, eh->ether_dhost, ETH_ALEN); // 设置目标MAC地址
168
169     print_packet_info(buffer); // 添加分号
170
171     // 发送数据包
172     if (sendto(sockfd, buffer, sizeof(struct ether_header) + sizeof(struct iphdr) +
sizeof(struct udphdr) + strlen("Hello, this is a test message."), 0, (struct sockaddr
*)&socket_address, sizeof(struct sockaddr_ll)) < 0) {
173         perror("sendto"); // 发送失败
174         return 1;
175     }
176
177     close(sockfd); // 关闭套接字
178     free(S_ADDR); // 释放动态分配的内存
179     free(D_ADDR); // 释放动态分配的内存
180     return 0; // 正常结束
181 }
182

```

- 检查网络接口设备以及MAC地址

```

1 ip link show
2
3 2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
default qlen 1000
4     link/ether 00:0c:29:77:75:72 brd ff:ff:ff:ff:ff:ff
5
6

```

- 可以多次发送的版本

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>

```

```

6  #include <sys/socket.h>
7  #include <netinet/ip.h>
8  #include <netinet/udp.h>
9  #include <netinet/ether.h>
10 #include <net/if.h>
11 #include <sys/ioctl.h>
12 #include <linux/if_packet.h>
13 #include <time.h>
14
15 #define BUFFER_SIZE 1518 // 以太网帧最大长度
16 #define ETHER_TYPE 0x0800 // 以太网类型字段 (0x0800表示IP协议)
17
18 // 全局变量
19 int UDP_SRC_PORT;
20 int UDP_DST_PORT;
21 char *S_ADDR;
22 char *D_ADDR;
23 unsigned char DEST_MAC[6];
24
25 void load_config(const char *filename) {
26     FILE *file = fopen(filename, "r");
27     if (!file) {
28         perror("Could not open config file");
29         exit(EXIT_FAILURE);
30     }
31
32     char line[256];
33     while (fgets(line, sizeof(line), file)) {
34         // 去掉换行符
35         line[strcspn(line, "\n")] = 0;
36
37         // 根据等号分割键值对
38         char key[32];
39         char value[256]; // 扩大值的大小以支持更长的字符串
40         if (sscanf(line, "%[^=]=%s", key, value) == 2) {
41             if (strcmp(key, "UDP_SRC_PORT") == 0) {
42                 UDP_SRC_PORT = atoi(value);
43             } else if (strcmp(key, "UDP_DST_PORT") == 0) {
44                 UDP_DST_PORT = atoi(value);
45             } else if (strcmp(key, "DEST_MAC") == 0) {
46                 sscanf(value, "%hhx:%hhx:%hhx:%hhx:%hhx:%hhx",
47                     &DEST_MAC[0], &DEST_MAC[1], &DEST_MAC[2],
48                     &DEST_MAC[3], &DEST_MAC[4], &DEST_MAC[5]);
49             } else if (strcmp(key, "S_ADDR") == 0) {
50                 S_ADDR = strdup(value); // 分配内存并复制源地址
51             } else if (strcmp(key, "D_ADDR") == 0) {
52                 D_ADDR = strdup(value); // 分配内存并复制目的地址
53             }
54         }
55     }
56
57     fclose(file);
58 }
59
60 void print_packet_info(const unsigned char *buffer) {

```



```

61     struct ether_header *eh = (struct ether_header *)buffer; // 以太网头
62     struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header)); // IP头
63     struct udphdr *udph = (struct udphdr *)(buffer + sizeof(struct ether_header) +
        sizeof(struct iphdr)); // UDP头
64     char *data = (char *)(buffer + sizeof(struct ether_header) + sizeof(struct iphdr)
        + sizeof(struct udphdr)); // 数据部分
65
66     // 获取当前时间
67     time_t now;
68     time(&now);
69     char *time_str = ctime(&now); // 获取当前时间
70     time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符
71
72     // 打印信息
73     printf("Sended Packet:\n");
74     printf("Time: %s\n", time_str);
75     printf("Source MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh->ether_shost));
76     printf("Destination MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh->ether_dhost));
77     printf("Source IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->saddr));
78     printf("Destination IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->daddr));
79     printf("TTL: %d\n", iph->ttl);
80     printf("Source Port: %d\n", ntohs(udph->source)); // 源端口
81     printf("Destination Port: %d\n", ntohs(udph->dest)); // 目的端口
82     printf("Data: %s\n", data); // 打印数据
83     printf("-----\n");
84 }
85
86
87 // 计算校验和的函数
88 unsigned short checksum(void *b, int len) {
89     unsigned short *buf = b;
90     unsigned int sum = 0;
91     unsigned short result;
92
93     // 对每两个字节进行求和
94     for (sum = 0; len > 1; len -= 2)
95         sum += *buf++;
96     // 如果有剩下的字节, 加上它
97     if (len == 1)
98         sum += *(unsigned char *)buf;
99     // 处理溢出
100    sum = (sum >> 16) + (sum & 0xFFFF);
101    sum += (sum >> 16);
102    result = ~sum; // 取反作为校验和
103    return result;
104 }
105
106 int main() {
107     // 先加载配置文件
108     load_config("config.txt");
109

```

```

110 // 创建原始套接字, 允许捕获所有以太网数据
111 int sockfd;
112 struct ifreq if_idx, if_mac; // 用于获取接口索引和MAC地址
113 struct sockaddr_ll socket_address; // 套接字地址结构
114 char buffer[BUFFER_SIZE]; // 数据缓冲区
115 char data[256]; // 用于存储用户输入的数据
116
117 if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) == -1) {
118     perror("socket");
119     return 1; // 出现错误, 退出
120 }
121
122 // 获取接口索引
123 memset(&if_idx, 0, sizeof(struct ifreq));
124 strncpy(if_idx.ifr_name, "ens33", IFNAMSIZ - 1); // eth0 为目标网络接口名称
125 if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0) {
126     perror("SIOCGIFINDEX"); // 获取索引失败
127     return 1;
128 }
129
130 // 获取接口 MAC 地址
131 memset(&if_mac, 0, sizeof(struct ifreq));
132 strncpy(if_mac.ifr_name, "ens33", IFNAMSIZ - 1);
133 if (ioctl(sockfd, SIOCGIFHWADDR, &if_mac) < 0) {
134     perror("SIOCGIFHWADDR"); // 获取MAC地址失败
135     return 1;
136 }
137
138 while (1) {
139     printf("Enter message to send (type 'exit' to quit): ");
140     fgets(data, sizeof(data), stdin);
141     data[strcspn(data, "\n")] = 0; // 去掉换行符
142
143     if (strcmp(data, "exit") == 0) {
144         break; // 输入"exit", 退出循环
145     }
146
147     // 构造以太网头
148     struct ether_header *eh = (struct ether_header *)buffer;
149     memcpy(eh->ether_shost, if_mac.ifr_hwaddr.sa_data, ETH_ALEN); // 设置源MAC地址
150     memcpy(eh->ether_dhost, DEST_MAC, ETH_ALEN); // 目标MAC地址
151     eh->ether_type = htons(ETHER_TYPE); // 设置以太网类
152
153     // 构造 IP 头
154     struct iphdr *iph = (struct iphdr *) (buffer + sizeof(struct ether_header));
155     iph->ihl = 5;
156
157     // IP头长度
158     iph->version = 4;
159
160     // IPv4
161     iph->tos = 0;
162
163     // 服务类型
164     iph->tot_len = htons(sizeof(struct iphdr) + sizeof(struct udphdr) +
165     strlen(data)); // 总长度
166     iph->id = htonl(54321);
167
168     // 标识符

```

```

160     iph->frag_off = 0;
161                                     // 不分片
162     iph->ttl = 255;
163                                     // 生存时间
164     iph->protocol = IPPROTO_UDP;
165                                     // 协议类型为UDP
166     iph->check = 0;
167                                     // 校验和开始为0
168     iph->saddr = inet_addr(S_ADDR);
169                                     // 源IP地址
170     iph->daddr = inet_addr(D_ADDR);
171                                     // 目的IP地址
172     iph->check = checksum((unsigned short *)iph, sizeof(struct iphdr));
173                                     // 计算并设置校验和
174
175     // 构造 UDP 头
176     struct udphdr *udph = (struct udphdr *)(buffer + sizeof(struct ether_header)
177 + sizeof(struct iphdr));
178     udph->source = htons(UDP_SRC_PORT);
179     // 源端口
180     udph->dest = htons(UDP_DST_PORT);
181     // 目的端口
182     udph->len = htons(sizeof(struct udphdr) + strlen(data)); // UDP报文长度
183     udph->check = 0;
184     // UDP 校验和可选
185
186     // 填充数据内容
187     char *payload = (char *)(buffer + sizeof(struct ether_header) + sizeof(struct
188 iphdr) + sizeof(struct udphdr));
189     strcpy(payload, data); // 数据内容
190
191     // 设置 socket 地址结构
192     socket_address.sll_ifindex = if_idx.ifr_ifindex; // 设置接口索引
193     socket_address.sll_halen = ETH_ALEN; // MAC地址长度
194     memcpy(socket_address.sll_addr, eh->ether_dhost, ETH_ALEN); // 设置目标MAC地址
195
196     // 发送数据包
197     if (sendto(sockfd, buffer, sizeof(struct ether_header) + sizeof(struct iphdr)
198 + sizeof(struct udphdr) + strlen(data), 0, (struct sockaddr *)&socket_address,
199 sizeof(struct sockaddr_ll)) < 0) {
200         perror("sendto"); // 发送失败
201         continue; // 继续循环
202     }
203
204     // 打印信息
205     print_packet_info(buffer);
206 }
207
208 close(sockfd); // 关闭套接字
209 free(S_ADDR); // 释放动态分配的内存
210 free(D_ADDR); // 释放动态分配的内存
211 return 0; // 正常结束
212 }
213

```

4 路由转发程序

• 代码

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <arpa/inet.h>
5  #include <netinet/ip.h>
6  #include <netinet/if_ether.h>
7  #include <netinet/ether.h>
8  #include <sys/socket.h>
9  #include <unistd.h>
10 #include <linux/if_packet.h>
11 #include <net/if.h>
12 #include <sys/ioctl.h>
13 #include <time.h>
14
15 #define BUFFER_SIZE 65536 // 定义缓冲区大小
16
17 // 计算校验和函数
18 unsigned short checksum(void *b, int len) {
19     unsigned short *buf = b;
20     unsigned int sum = 0;
21     unsigned short result;
22
23     // 计算校验和
24     for (sum = 0; len > 1; len -= 2)
25         sum += *buf++;
26
27     // 如果有剩下的字节, 加上它
28     if (len == 1)
29         sum += *(unsigned char *)buf;
30
31     // 处理溢出
32     sum = (sum >> 16) + (sum & 0xFFFF);
33     sum += (sum >> 16);
34     result = ~sum; // 取反作为校验和
35     return result;
36 }
37
38 int main() {
39     int sockfd;
40     struct sockaddr saddr;
41     unsigned char *buffer = (unsigned char *)malloc(BUFFER_SIZE);
42
43     // 创建原始套接字
44     sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP));
45     if (sockfd < 0) {
46         perror("Socket creation failed");
47         return 1;
48     }
49
50     while (1) {
51         int saddr_len = sizeof(saddr);
52         // 接收数据包
```

```

53     int data_size = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, &saddr,
(socklen_t*)&saddr_len);
54     if (data_size < 0) {
55         perror("Recvfrom error");
56         return 1;
57     }
58
59     // 解析以太网头和 IP 头
60     struct ethhdr *eth_header = (struct ethhdr *)buffer;
61     struct iphdr *ip_header = (struct iphdr *)(buffer + sizeof(struct ethhdr));
62     char src_ip[INET_ADDRSTRLEN];
63     char dest_ip[INET_ADDRSTRLEN];
64
65     // 转换 IP 地址格式
66     inet_ntop(AF_INET, &(ip_header->saddr), src_ip, INET_ADDRSTRLEN);
67     inet_ntop(AF_INET, &(ip_header->daddr), dest_ip, INET_ADDRSTRLEN);
68
69     // 判断源和目的 IP 是否符合要求
70     if (strcmp(src_ip, "192.168.1.1") == 0 && strcmp(dest_ip, "192.168.1.3") ==
0) {
71         // 获取当前系统时间
72         time_t rawtime;
73         struct tm *timeinfo;
74         char time_str[100];
75
76         time(&rawtime);
77         timeinfo = localtime(&rawtime);
78
79         // 格式化时间字符串
80         strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S", timeinfo);
81
82         // 打印信息
83         printf("[%s] Captured packet from %s to %s\n", time_str, src_ip,
dest_ip);
84
85         // 修改 TTL
86         ip_header->ttl -= 1;
87         ip_header->check = 0; // 重置校验和
88         ip_header->check = checksum((unsigned short *)ip_header, ip_header->ihl *
4); // 计算新校验和
89
90         // 发送数据包到目的主机
91         struct ifreq ifr, ifr_mac;
92         struct sockaddr_ll dest;
93
94         // 获取网卡接口索引
95         memset(&ifr, 0, sizeof(ifr));
96         snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), "eth0");
97         if (ioctl(sockfd, SIOCGIFINDEX, &ifr) < 0) {
98             perror("ioctl");
99             return 1;
100         }
101
102         // 获取网卡接口 MAC 地址
103         memset(&ifr_mac, 0, sizeof(ifr_mac));
104         snprintf(ifr_mac.ifr_name, sizeof(ifr_mac.ifr_name), "eth0");

```

```

105         if (ioctl(sockfd, SIOCGIFHWADDR, &ifr_mac) < 0) {
106             perror("ioctl");
107             return 1;
108         }
109
110         // 设置目标 MAC 地址 (假设目标地址已知)
111         unsigned char target_mac[ETH_ALEN] = {0x00, 0x0c, 0x29, 0x48, 0xd3,
0xf7}; // 替换为实际的目标 MAC 地址
112         memset(&dest, 0, sizeof(dest));
113         dest.sll_ifindex = ifr.ifr_ifindex;
114         dest.sll_halen = ETH_ALEN;
115         memcpy(dest.sll_addr, target_mac, ETH_ALEN); // 复制目标 MAC 地址
116
117         // 构造新的以太网帧头
118         memcpy(eth_header->h_dest, target_mac, ETH_ALEN); // 目标 MAC 地址
119         memcpy(eth_header->h_source, ifr_mac.ifr_hwaddr.sa_data, ETH_ALEN); // 源
MAC 地址
120         eth_header->h_proto = htons(ETH_P_IP); // 以太网类型为 IP
121
122         printf("Interface name: %s, index: %d\n", ifr.ifr_name, ifr.ifr_ifindex);
123
124         // 发送数据包
125         if (sendto(sockfd, buffer, data_size, 0, (struct sockaddr *)&dest,
sizeof(dest)) < 0) {
126             perror("Sendto error");
127             return 1;
128         }
129         printf("Datagram forwarded.\n");
130     } else {
131         printf("Ignored packet from %s to %s\n", src_ip, dest_ip);
132     }
133 }
134
135 close(sockfd);
136 free(buffer);
137 return 0;
138 }
139

```

- 接收路由表的版本

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <arpa/inet.h>
5  #include <netinet/ip.h>
6  #include <netinet/if_ether.h>
7  #include <netinet/ether.h>
8  #include <sys/socket.h>
9  #include <unistd.h>
10 #include <linux/if_packet.h>
11 #include <net/if.h>
12 #include <sys/ioctl.h>
13 #include <time.h>
14 #include <netinet/udp.h>
15

```

```

16
17 #define BUFFER_SIZE 65536
18 #define MAX_ROUTES 100
19
20 typedef struct {
21     char src_ip[INET_ADDRSTRLEN];
22     char dest_ip[INET_ADDRSTRLEN];
23     unsigned char target_mac[ETH_ALEN];
24 } Route;
25
26
27 Route routing_table[MAX_ROUTES];
28 int route_count = 0;
29
30 unsigned short checksum(void *b, int len) {
31     unsigned short *buf = b;
32     unsigned int sum = 0;
33     unsigned short result;
34
35     • for (sum = 0; len > 1; len -= 2)
36     •     sum += *buf++;
37     • if (len == 1)
38     •     sum += *(unsigned char *)buf;
39     • sum = (sum >> 16) + (sum & 0xFFFF);
40     • sum += (sum >> 16);
41     • result = ~sum;
42     • return result;
43 }
44
45 // 加载路由表
46 void load_routing_table(const char *filename)
47 {
48     FILE *file = fopen(filename, "r");
49     if (!file)
50     {
51         perror("Could not open routing table file");
52         exit(EXIT_FAILURE);
53     }
54
55     • char line[256]; // 用于读取每一行
56     • while (route_count < MAX_ROUTES && fgets(line, sizeof(line), file))
57     • {
58     •     // 去掉换行符
59     •     line[strcspn(line, "\n")] = 0;
60
61     •     // 分割字符串
62     •     char *src_ip = strtok(line, " ");
63     •     char *dest_ip = strtok(NULL, " ");
64     •     char *mac_str = strtok(NULL, " ");
65
66     •     printf("Processing entry: SRC IP = %s, DEST IP = %s, MAC = %s\n", src_ip,
dest_ip, mac_str);
67
68     •     if (src_ip && dest_ip && mac_str)
69     •     {
70     •         // 复制源和目的 IP

```

```

71 |     strcpy(routing_table[route_count].src_ip, src_ip);
72 |     strcpy(routing_table[route_count].dest_ip, dest_ip);
73 |
74 |     // 解析 MAC 地址
75 |     if (sscanf(mac_str, "%hhx:%hhx:%hhx:%hhx:%hhx:%hhx",
76 |                &routing_table[route_count].target_mac[0],
77 |                &routing_table[route_count].target_mac[1],
78 |                &routing_table[route_count].target_mac[2],
79 |                &routing_table[route_count].target_mac[3],
80 |                &routing_table[route_count].target_mac[4],
81 |                &routing_table[route_count].target_mac[5]) == 6)
82 |     {
83 |         printf("Successfully added route: %s -> %s with MAC:
%02x:%02x:%02x:%02x:%02x:%02x\n",
84 |                routing_table[route_count].src_ip,
85 |                routing_table[route_count].dest_ip,
86 |                routing_table[route_count].target_mac[0],
87 |                routing_table[route_count].target_mac[1],
88 |                routing_table[route_count].target_mac[2],
89 |                routing_table[route_count].target_mac[3],
90 |                routing_table[route_count].target_mac[4],
91 |                routing_table[route_count].target_mac[5]);
92 |         route_count++;
93 |     }
94 |     else
95 |     {
96 |         fprintf(stderr, "Invalid MAC address format: %s\n", mac_str);
97 |     }
98 | }
99 | else
100 | {
101 |     fprintf(stderr, "Invalid routing table entry: %s\n", line);
102 | }
103 | }
104 | fclose(file);
105 | }
106 |
107 | // 打印路由表
108 | void print_routing_table()
109 | {
110 |     printf("Routing Table:\n");
111 |     printf("SRC IP\t\tDEST IP\t\tTARGET MAC\n");
112 |     for (int i = 0; i < route_count; i++)
113 |     {
114 |         printf("%s\t%s\t%02x:%02x:%02x:%02x:%02x:%02x\n",
115 |                routing_table[i].src_ip,
116 |                routing_table[i].dest_ip,
117 |                routing_table[i].target_mac[0], routing_table[i].target_mac[1],
118 |                routing_table[i].target_mac[2], routing_table[i].target_mac[3],
119 |                routing_table[i].target_mac[4], routing_table[i].target_mac[5]);
120 |     }
121 | }
122 |
123 | void print_packet_info(const unsigned char *buffer) {
124 |     struct ether_header *eh = (struct ether_header *)buffer;           // 以
    以太网头

```



```

125     struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header)); // IP
    头
126     struct udphdr *udph = (struct udphdr *)(buffer + sizeof(struct ether_header) +
sizeof(struct iphdr)); // UDP头
127     char *data = (char *)(buffer + sizeof(struct ether_header) + sizeof(struct iphdr)
+ sizeof(struct udphdr)); // 数据部分
128
129     • // 获取当前时间
130     • time_t now;
131     • time(&now);
132     • char *time_str = ctime(&now); // 获取当前时间
133     • time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符
134
135     • // 打印信息
136     • printf("Sended Packet:\n");
137     • printf("Time: %s\n", time_str);
138     • printf("Source MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh-
>ether_shost));
139     • printf("Destination MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh-
>ether_dhost));
140     • printf("Source IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->saddr));
141     • printf("Destination IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph-
>daddr));
142     • printf("TTL: %d\n", iph->ttl);
143     • printf("Source Port: %d\n", ntohs(udph->source)); // 源端口
144     • printf("Destination Port: %d\n", ntohs(udph->dest)); // 目的端口
145     • printf("Data: %s\n", data); // 打印数据
146     • printf("-----\n");
147 }
148
149 int main() {
150     int sockfd;
151     struct sockaddr saddr;
152     unsigned char *buffer = (unsigned char *)malloc(BUFFER_SIZE);
153
154     • // Load routing table from configuration file
155     • load_routing_table("config.txt");
156     •
157     • // Print routing table
158     • print_routing_table();
159
160     • sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP));
161     • if (sockfd < 0) {
162     •     perror("Socket creation failed");
163     •     return 1;
164     • }
165
166     • while (1) {
167     •     int saddr_len = sizeof(saddr);
168     •     int data_size = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, &saddr,
(socklen_t*)&saddr_len);
169     •     if (data_size < 0) {
170     •         perror("Recvfrom error");
171     •         return 1;
172     •     }
173

```

```

174 • struct ethhdr *eth_header = (struct ethhdr *)buffer;
175 • struct iphdr *ip_header = (struct iphdr *)(buffer + sizeof(struct ethhdr));
176 • char src_ip[INET_ADDRSTRLEN];
177 • char dest_ip[INET_ADDRSTRLEN];
178
179 • inet_ntop(AF_INET, &(ip_header->saddr), src_ip, INET_ADDRSTRLEN);
180 • inet_ntop(AF_INET, &(ip_header->daddr), dest_ip, INET_ADDRSTRLEN);
181
182 • for (int i = 0; i < route_count; i++) {
183 •     if (strcmp(src_ip, routing_table[i].src_ip) == 0 && strcmp(dest_ip,
routing_table[i].dest_ip) == 0) {
184 •         // 获取当前系统时间
185 •         time_t rawtime;
186 •         struct tm *timeinfo;
187 •         char time_str[100];
188
189 •         time(&rawtime);
190 •         timeinfo = localtime(&rawtime);
191 •         strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S", timeinfo);
192
193 •         // 打印信息
194 •         printf("[%s] Captured packet from %s to %s\n", time_str, src_ip,
dest_ip);
195 •         print_packet_info(buffer);
196
197 •         // 修改 TTL
198 •         ip_header->ttl -= 1;
199 •         ip_header->check = 0;
200 •         ip_header->check = checksum((unsigned short *)ip_header, ip_header-
>ihl * 4);
201
202 •         // 发送数据包到目的主机
203 •         struct ifreq ifr;
204 •         struct sockaddr_ll dest;
205
206 •         // 获取网卡接口索引
207 •         memset(&ifr, 0, sizeof(ifr));
208 •         snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), "ens33");
209 •         if (ioctl(sockfd, SIOCGIFINDEX, &ifr) < 0) {
210 •             perror("ioctl");
211 •             return 1;
212 •         }
213
214 •         memset(&dest, 0, sizeof(dest));
215 •         dest.sll_ifindex = ifr.ifr_ifindex;
216 •         dest.sll_halen = ETH_ALEN;
217 •         memcpy(dest.sll_addr, routing_table[i].target_mac, ETH_ALEN); // 使
用路由表中的目标MAC地址
218
219 •         // 构造新的以太网帧头
220 •         memcpy(eth_header->h_dest, routing_table[i].target_mac, ETH_ALEN);
221 •         memcpy(eth_header->h_source, ifr.ifr_hwaddr.sa_data, ETH_ALEN);
222 •         eth_header->h_proto = htons(ETH_P_IP);
223
224 •         if (sendto(sockfd, buffer, data_size, 0, (struct sockaddr *)&dest,
sizeof(dest)) < 0) {

```

```

225 |         perror("Sendto error");
226 |         return 1;
227 |     }
228 |     printf("Datagram forwarded to %s\n", routing_table[i].dest_ip);
229 |     break;
230 | } else {
231 |     // printf("Ignored packet from %s to %s\n", src_ip, dest_ip);
232 | }
233 | }
234 | }
235 |
236 | close(sockfd);
237 | free(buffer);
238 | return 0;
239 | }

```

5 接受主机程序

```

1 | #include <stdio.h>
2 | #include <stdlib.h>
3 | #include <string.h>
4 | #include <arpa/inet.h>
5 | #include <netinet/ip.h>
6 | #include <netinet/if_ether.h>
7 | #include <netinet/ether.h>
8 | #include <sys/socket.h>
9 | #include <unistd.h>
10 | #include <linux/if_packet.h>
11 | #include <net/if.h>
12 | #include <sys/ioctl.h>
13 | #include <time.h>
14 | #include <netinet/udp.h>
15 |
16 | #define PORT 54321
17 |
18 | void print_packet_info(const unsigned char *buffer) {
19 |     struct ether_header *eh = (struct ether_header *)buffer;           // 以太网头
20 |     struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header)); // IP头
21 |     struct udphdr *udph = (struct udphdr *)(buffer + sizeof(struct ether_header) +
22 | sizeof(struct iphdr)); // UDP头
23 |     char *data = (char *)(buffer + sizeof(struct ether_header) + sizeof(struct iphdr)
24 | + sizeof(struct udphdr)); // 数据部分
25 |
26 |     // 获取当前时间
27 |     time_t now;
28 |     time(&now);
29 |     char *time_str = ctime(&now); // 获取当前时间
30 |     time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符
31 |
32 |     // 打印信息
33 |     printf("Sended Packet:\n");
34 |     printf("Time: %s\n", time_str);

```

```

33     printf("Source MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh-
>ether_shost));
34     printf("Destination MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh-
>ether_dhost));
35     printf("Source IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->saddr));
36     printf("Destination IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->daddr));
37     printf("TTL: %d\n", iph->ttl);
38     printf("Source Port: %d\n", ntohs(udph->source)); // 源端口
39     printf("Destination Port: %d\n", ntohs(udph->dest)); // 目的端口
40     printf("Data: %s\n", data); // 打印数据
41     printf("-----\n");
42 }
43
44 int main() {
45     int sockfd;
46     struct sockaddr_in server_addr, client_addr;
47     socklen_t addr_len = sizeof(client_addr);
48     char buffer[1024];
49
50     // 创建 UDP 套接字
51     sockfd = socket(AF_INET, SOCK_DGRAM, 0);
52     if (sockfd < 0) {
53         perror("Socket creation failed");
54         return 1;
55     }
56
57     // 绑定套接字到端口
58     memset(&server_addr, 0, sizeof(server_addr));
59     server_addr.sin_family = AF_INET;
60     server_addr.sin_addr.s_addr = INADDR_ANY;
61     server_addr.sin_port = htons(PORT);
62
63     if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
64         perror("Bind failed");
65         return 1;
66     }
67
68     // 接收数据包
69     int recv_len = recvfrom(sockfd, buffer, sizeof(buffer) - 1, 0, (struct sockaddr
70                                                                    *)&client_addr,
&addr_len);
71     if (recv_len < 0) {
72         perror("Recvfrom failed");
73         return 1;
74     }
75
76     buffer[recv_len] = '\0';
77     printf("Received message: %s\n", buffer);
78
79     close(sockfd);
80     return 0;
81 }

```

- 可循环接收版本

```
1 #include <stdio.h>
```

```

2  #include <stdlib.h>
3  #include <string.h>
4  #include <arpa/inet.h>
5  #include <netinet/udp.h>
6  #include <sys/socket.h>
7  #include <unistd.h>
8  #include <time.h>
9
10 #define PORT 54321
11 #define BUFFER_SIZE 1024
12
13 int main() {
14     int sockfd;
15     struct sockaddr_in server_addr, client_addr;
16     socklen_t addr_len = sizeof(client_addr);
17     char buffer[BUFFER_SIZE];
18
19     // 创建 UDP 套接字
20     sockfd = socket(AF_INET, SOCK_DGRAM, 0);
21     if (sockfd < 0) {
22         perror("Socket creation failed");
23         return EXIT_FAILURE;
24     }
25
26     // 绑定套接字到端口
27     memset(&server_addr, 0, sizeof(server_addr));
28     server_addr.sin_family = AF_INET;
29     server_addr.sin_addr.s_addr = INADDR_ANY;
30     server_addr.sin_port = htons(PORT);
31
32     if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
33         perror("Bind failed");
34         close(sockfd);
35         return EXIT_FAILURE;
36     }
37
38     // 循环接收数据包
39     printf("Server is listening on port %d...\n", PORT);
40     while (1) {
41         // 清空接收缓冲区
42         memset(buffer, 0, sizeof(buffer));
43
44         // 接收数据包
45         int recv_len = recvfrom(sockfd, buffer, sizeof(buffer) - 1, 0, (struct
sockaddr *)&client_addr, &addr_len);
46         if (recv_len < 0) {
47             perror("Recvfrom failed");
48             break; // 发生错误时退出循环
49         }
50
51         // 获取当前时间
52         time_t now = time(NULL);
53         char *time_str = ctime(&now); // 获取当前时间
54         if (time_str) {
55             time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符
56         }

```

```

57
58     buffer[recv_len] = '\0'; // 确保字符串结束
59
60     // 打印信息
61     printf("Received Packet:\n");
62     printf("Time: %s\n", time_str);
63     printf("Sender: %s:%d\n", inet_ntoa(client_addr.sin_addr),
        ntohs(client_addr.sin_port));
64     printf("Message: %s\n", buffer);
65     printf("-----\n");
66 }
67
68 close(sockfd);
69 return EXIT_SUCCESS;
70 }
71

```

Note vim → gg + dG 删除所有内容

基于双网口主机的路由转发

1 双网口主机配置

过虚拟机管理工具（如VMware、VirtualBox等）在虚拟机设置中添加新的网络适配器。在添加之后，你需要重新启动虚拟机并检查网络接口列表，看看新的接口是否被识别。

硬件 选项

设备	摘要
内存	4 GB
处理器	4
硬盘 (SCSI)	40 GB
CD/DVD (SATA)	正在使用文件 E:\24qiu\CSNet\...
网络适配器	NAT
网络适配器 2	NAT
USB 控制器	存在
声卡	自动检测
显示	自动检测

设备状态

☐ 已连接(C)

☒ 启动时连接(O)

网络连接

☐ 桥接模式(B): 直接连接物理网络

☐ 复制物理网络连接状态(P)

☒ NAT 模式(N): 用于共享主机的 IP 地址

☐ 仅主机模式(H): 与主机共享的专用网络

☐ 自定义(U): 特定虚拟网络

VMnet0 (自动桥接) ▾

☐ LAN 区段(L):

▾

LAN 区段(S)... 高级(V)...

添加(A)... 移除(R)

确定 取消 帮助

```

aircraft@root:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:30:37:0c brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.222.10/24 brd 192.168.222.255 scope global ens33
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe30:370c/64 scope link
        valid_lft forever preferred_lft forever
3: ens37: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:30:37:16 brd ff:ff:ff:ff:ff:ff
    altname enp2s5
    inet 192.168.223.10/24 brd 192.168.223.255 scope global ens37
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe30:3716/64 scope link
        valid_lft forever preferred_lft forever
aircraft@root:~$

```

```

root@root:/home/aircraft/lab4# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:30:37:0c brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.222.10/24 brd 192.168.222.255 scope global ens33
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe30:370c/64 scope link
        valid_lft forever preferred_lft forever
3: ens37: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:30:37:16 brd ff:ff:ff:ff:ff:ff
    altname enp2s5
    inet 192.168.223.10/24 brd 192.168.223.255 scope global ens37
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe30:3716/64 scope link
        valid_lft forever preferred_lft forever
root@root:/home/aircraft/lab4#

```

```

1 auto ens33
2 iface ens33 inet static
3     address 192.168.222.10
4     netmask 255.255.255.0
5     gateway 192.168.222.2
6
7 auto ens37
8 iface ens37 inet static
9     address 192.168.223.10
10    netmask 255.255.255.0
11

```



```
source /etc/network/interfaces.d/*
```

```
# The loopback network interface
```

```
auto lo
```

```
iface lo inet loopback
```

```
auto ens33
```

```
iface ens33 inet static
```

```
    address 192.168.222.10
```

```
    netmask 255.255.255.0
```

```
    gateway 192.168.222.2
```

```
auto ens37
```

```
iface ens37 inet static
```

```
    address 192.168.223.10
```

```
    netmask 255.255.255.0
```

```
root@root:/home/aircraft# █
```

2 发送端主机配置

2.1 配置静态 IP

```
1 vim /etc/network/interfaces
```

- 主机1

```
1 auto ens33
2 iface ens33 inet static
3     address 192.168.222.20
4     netmask 255.255.255.0
5     gateway 192.168.222.10
```

- 主机2

```

1 auto ens33
2 iface ens33 inet static
3     address 192.168.223.20
4     netmask 255.255.255.0
5     gateway 192.168.223.10
6
7 auto ens37
8 iface ens37 inet static
9     address 192.168.222.30
10    netmask 255.255.255.0
11    gateway 192.168.222.10

```

2.2 修改路由表

- 主机1

路由主机已经配置了 `ens33` 和 `ens37`，并且客户主机需要通过路由主机访问 `192.168.223.0/24` 网络（通过 `ens37` 接口），你可以在客户主机上执行以下步骤：

1. 为客户主机配置 IP 地址（假设你使用的是 `192.168.222.x` 网络）：

```

1 sudo ip addr add 192.168.222.20/24 dev ens33
2 sudo ip link set ens33 up

```

2. 添加到 `192.168.223.0/24` 网络的路由：

```

1 sudo ip route add 192.168.223.0/24 via 192.168.222.10

```

这里的命令含义如下：

- `192.168.223.0/24` 是目标网络地址。
- `via 192.168.222.10` 是路由主机的 IP 地址，客户主机会将数据包发送到该地址以便通过路由主机访问 `192.168.223.0/24` 网络。

3. 确认路由是否成功添加：

```

1 ip route show

```

这将显示当前的路由表，确保新添加的路由条目出现在其中。

4. 测试连接：尝试 ping 一下 `192.168.223.x` 网络中的设备，确认路由是否生效。

- 主机2

```

1 sudo ip route add 192.168.222.0/24 via 192.168.223.10

```

2.3 添加默认网关

```

1 sudo route add default gw 192.168.223.10

```

3 发送主机（源主机） 192.168.222.20

这个程序将发送一个简单的 UDP 数据包到路由器：

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <arpa/inet.h>
5  #include <sys/socket.h>
6  #include <unistd.h>
7  #include <time.h>
8
9  #define DEST_IP "192.168.223.20"
10 #define DEST_PORT 12345
11 #define BUFFER_SIZE 1024
12
13 int main() {
14     int sockfd;
15     struct sockaddr_in dest_addr, local_addr;
16     char message[BUFFER_SIZE];
17     socklen_t addr_len = sizeof(local_addr);
18
19     // 创建 UDP 套接字
20     sockfd = socket(AF_INET, SOCK_DGRAM, 0);
21     if (sockfd < 0) {
22         perror("Socket creation failed");
23         return 1;
24     }
25
26     // 设置目的地址
27     memset(&dest_addr, 0, sizeof(dest_addr));
28     dest_addr.sin_family = AF_INET;
29     dest_addr.sin_port = htons(DEST_PORT);
30     inet_pton(AF_INET, DEST_IP, &dest_addr.sin_addr);
31
32     printf("Enter messages to send (type 'exit' to quit):\n");
33
34     while (1) {
35         printf("Message: ");
36         fgets(message, BUFFER_SIZE, stdin);
37
38         // 去掉换行符
39         message[strcspn(message, "\n")] = 0;
40
41         // 检查是否输入 "exit" 来退出
42         if (strcmp(message, "exit") == 0) {
43             printf("Exiting...\n");
44             break;
45         }
46
47         // 发送数据包
48         if (sendto(sockfd, message, strlen(message), 0, (struct sockaddr *)&dest_addr,
49             sizeof(dest_addr)) < 0) {
50             perror("Sendto failed");
51             return 1;
52         }
53     }
```

```

52
53     // 获取并打印当前时间
54     time_t now = time(NULL);
55     char *time_str = ctime(&now);
56     if (time_str) {
57         time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符
58     }
59
60     // 获取发送端的 IP 和端口
61     if (getsockname(sockfd, (struct sockaddr *)&local_addr, &addr_len) == -1) {
62         perror("getsockname failed");
63         return 1;
64     }
65
66     // 打印发送的 IP、端口、时间和数据
67     printf("Message sent at %s\n", time_str);
68     printf("From %s:%d to %s:%d\n", "192.168.222.20", ntohs(local_addr.sin_port),
DEST_IP, DEST_PORT);
69     printf("Data: %s\n", message);
70     printf("-----\n");
71 }
72
73 close(sockfd);
74 return 0;
75 }
76

```

Note：因为作过下一条路由配置，所以会把 192.168.223.20 的数据包传到下一条 192.168.222.10

4 路由转发 (192.168.222.10 -> 192.168.223.10)

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <arpa/inet.h>
5  #include <netinet/ip.h>
6  #include <netinet/if_ether.h>
7  #include <sys/socket.h>
8  #include <sys/ioctl.h>
9  #include <net/if.h>
10 #include <netpacket/packet.h>
11 #include <unistd.h>
12 #include <netinet/udp.h>
13 #include <time.h>
14
15 #include <netinet/ether.h> // 包含 ether_ntoa 函数的头文件
16
17 #define BUFFER_SIZE 65536
18
19 struct route_entry {
20     uint32_t dest;
21     uint32_t gateway;
22     uint32_t netmask;
23     char interface[IFNAMSIZ];
24 };
25

```

```

26 struct route_entry route_table[1]; // 只声明一个空的数组项
27 int route_table_size = 1; // 路由表大小
28
29 unsigned short checksum(unsigned short *buf, int nwords) {
30     unsigned long sum;
31     for (sum = 0; nwords > 0; nwords--)
32         sum += *buf++;
33     sum = (sum >> 16) + (sum & 0xffff);
34     sum += (sum >> 16);
35     return (unsigned short)(~sum);
36 }
37
38 struct route_entry *lookup_route(uint32_t dest_ip) {
39     for (int i = 0; i < route_table_size; i++) {
40         if ((dest_ip & route_table[i].netmask) == (route_table[i].dest &
41 route_table[i].netmask)) {
42             return &route_table[i];
43         }
44     }
45     return NULL;
46 }
47
48 void print_packet_info(const unsigned char *buffer, int data_size) {
49     struct ether_header *eth_header = (struct ether_header *)buffer; // 以太网头
50     struct iphdr *ip_header = (struct iphdr *)(buffer + sizeof(struct ether_header));
51     // IP头
52     char *data = (char *)(buffer + sizeof(struct ether_header) + ip_header->ihl * 4 +
53 sizeof(struct udphdr)); // 数据部分
54
55     // 获取当前时间
56     time_t now;
57     time(&now);
58     char *time_str = ctime(&now);
59     time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符
60
61     // 计算数据部分的大小
62     int data_length = data_size - (data - (char *)buffer); // 数据部分的长度
63
64     // 创建一个临时数组以存储数据
65     char temp[data_length + 1]; // +1 用于字符串终止符
66
67     // 将数据复制到临时数组
68     memcpy(temp, data, data_length);
69     temp[data_length] = '\0'; // 添加字符串结束符
70
71     // 打印信息
72     printf("----- Received Packet -----\n");
73     printf("Time: %s\n", time_str);
74     printf("Source MAC Address: %s\n", ether_ntoa((struct ether_addr *)eth_header->ether_shost));
75     printf("Destination MAC Address: %s\n", ether_ntoa((struct ether_addr *)eth_header->ether_dhost));
76     printf("Source IP Address: %s\n", inet_ntoa(*(struct in_addr *)&ip_header->saddr));

```

```

75     printf("Destination IP Address: %s\n", inet_ntoa(*(struct in_addr *)&ip_header-
>daddr));
76     printf("TTL: %d\n", ip_header->ttl);
77     printf("Data: %s\n", temp); // 打印数据内容
78
79     // 打印数据的十六进制表示
80     printf("Hex Data: ");
81     for (int i = 0; i < data_length; i++) {
82         printf("%02x ", (unsigned char)temp[i]);
83     }
84     printf("\n");
85     printf("-----\n");
86 }
87
88 void print_send_packet_info(const unsigned char *buffer, int data_size) {
89     struct ether_header *eth_header = (struct ether_header *)buffer; // 以太网头
90     struct iphdr *ip_header = (struct iphdr *)(buffer + sizeof(struct ether_header));
91     // IP头
92     char *data = (char *)(buffer + sizeof(struct ether_header) + ip_header->ihl * 4 +
sizeof(struct udphdr)); // 数据部分
93
94     // 获取当前时间
95     time_t now;
96     time(&now);
97     char *time_str = ctime(&now);
98     time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符
99
100    // 计算数据部分的大小
101    int data_length = data_size - (data - (char *)buffer); // 数据部分的长度
102
103    // 创建一个临时数组以存储数据
104    char temp[data_length + 1]; // +1 用于字符串终止符
105
106    // 将数据复制到临时数组
107    memcpy(temp, data, data_length);
108    temp[data_length] = '\0'; // 添加字符串结束符
109
110    // 打印信息
111    printf("----- Send Packet ----- \n");
112    printf("Time: %s\n", time_str);
113    printf("Source MAC Address: %s\n", ether_ntoa((struct ether_addr *)eth_header-
>ether_shost));
114    printf("Destination MAC Address: %s\n", ether_ntoa((struct ether_addr
*)eth_header->ether_dhost));
115    printf("Source IP Address: %s\n", inet_ntoa(*(struct in_addr *)&ip_header-
>saddr));
116    printf("Destination IP Address: %s\n", inet_ntoa(*(struct in_addr *)&ip_header-
>daddr));
117    printf("TTL: %d\n", ip_header->ttl);
118 }
119
120 int main() {
121     int sockfd;
122     struct sockaddr saddr;
123     unsigned char *buffer = (unsigned char *)malloc(BUFFER_SIZE);
124     socklen_t saddr_len = sizeof(saddr);

```

```

124
125 // 初始化路由表
126 route_table[0].dest = inet_addr("192.168.223.0");
127 route_table[0].gateway = inet_addr("192.168.223.10");
128 route_table[0].netmask = inet_addr("255.255.255.0");
129 strncpy(route_table[0].interface, "ens37", IFNAMSIZ);
130
131 sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP));
132 if (sockfd < 0) {
133     perror("Socket creation failed");
134     return 1;
135 }
136
137 while (1) {
138     int data_size = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, &saddr, &saddr_len);
139     if (data_size < 0) {
140         perror("Recvfrom error");
141         return 1;
142     }
143
144     struct iphdr *ip_header = (struct iphdr *)(buffer + sizeof(struct
ether_header));
145     struct route_entry *route = lookup_route(ip_header->daddr);
146     if (route == NULL) {
147         continue;
148     }
149     print_packet_info(buffer, data_size);
150
151     // 修改 TTL
152     ip_header->ttl -= 1;
153     ip_header->check = 0;
154     ip_header->check = checksum((unsigned short *)ip_header, ip_header->ihl * 4);
155     // inet_ntop(AF_INET, &(ip_header->saddr), "192.168.223.10",
INET_ADDRSTRLEN);
156
157     // 获取接口索引
158     struct ifreq ifr, ifr_mac;
159     struct sockaddr_ll dest;
160     memset(&ifr, 0, sizeof(ifr));
161     snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), route->interface);
162     if (ioctl(sockfd, SIOCGIFINDEX, &ifr) < 0) {
163         perror("ioctl SIOCGIFINDEX failed");
164         return 1;
165     }
166
167     // 获取接口的 MAC 地址
168     memset(&ifr_mac, 0, sizeof(ifr_mac));
169     snprintf(ifr_mac.ifr_name, sizeof(ifr_mac.ifr_name), route->interface);
170     if (ioctl(sockfd, SIOCGIFHWADDR, &ifr_mac) < 0) {
171         perror("ioctl SIOCGIFHWADDR failed");
172         return 1;
173     }
174
175     // 设置目标 MAC 地址 (替换为实际的目标 MAC 地址)
176     unsigned char target_mac[ETH_ALEN] = {0x00, 0x0c, 0x29, 0x17, 0xe2, 0xd9};
177     memset(&dest, 0, sizeof(dest));

```

```

178     dest.sll_ifindex = ifr.ifr_ifindex;
179     dest.sll_halen = ETH_ALEN;
180     memcpy(dest.sll_addr, target_mac, ETH_ALEN);
181     // 打印目标 MAC 地址
182     printf("Send to MAC: %02x:%02x:%02x:%02x:%02x:%02x\n",
183           target_mac[0], target_mac[1], target_mac[2],
184           target_mac[3], target_mac[4], target_mac[5]);
185
186     // 设置以太网帧头
187     struct ether_header *eth_header = (struct ether_header *)buffer;
188     memcpy(eth_header->ether_dhost, target_mac, ETH_ALEN);
189     memcpy(eth_header->ether_shost, ifr_mac.ifr_hwaddr.sa_data, ETH_ALEN);
190     eth_header->ether_type = htons(ETH_P_IP);
191
192     print_send_packet_info(buffer, data_size);
193     printf("Forwarding packet on interface: %s (index: %d)\n", ifr.ifr_name,
194           ifr.ifr_ifindex);
195
196     // 发送数据包
197     if (sendto(sockfd, buffer, data_size, 0, (struct sockaddr *)&dest,
198           sizeof(dest)) < 0) {
199         perror("Sendto error");
200         return 1;
201     }
202     printf("Packet forwarded to %s\n", inet_ntoa(*(struct in_addr *)&ip_header-
203           >daddr));
204     printf("-----\n");
205 }
206
207     close(sockfd);
208     free(buffer);
209     return 0;
210 }

```

5 接收主机程序 (192.168.223.20)

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <arpa/inet.h>
5  #include <sys/socket.h>
6  #include <unistd.h>
7  #include <time.h>
8
9  #define PORT 12345
10
11 void print_packet_info(const struct sockaddr_in *client_addr, const struct sockaddr_in
12 *server_addr, const char *buffer, int recv_len) {
13     // 获取当前时间
14     time_t now;
15     time(&now);
16     char *time_str = ctime(&now);
17     time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符

```



```

18 // 打印接收的包信息
19 printf("----- Received Packet -----\n");
20 printf("Time: %s\n", time_str);
21 printf("Source IP Address: %s\n", inet_ntoa(client_addr->sin_addr));
22 printf("Source Port: %d\n", ntohs(client_addr->sin_port));
23 printf("Destination IP Address: %s\n", inet_ntoa(server_addr->sin_addr));
24 printf("Destination Port: %d\n", ntohs(server_addr->sin_port));
25
26 printf("Data (hex): ");
27 for (int i = 0; i < recv_len; i++) {
28     printf("%02x ", (unsigned char)buffer[i]);
29 }
30 printf("\n-----\n");
31 }
32
33 int main() {
34     int sockfd;
35     struct sockaddr_in server_addr, client_addr;
36     socklen_t addr_len = sizeof(client_addr);
37     char buffer[1024];
38
39     // 创建 UDP 套接字
40     sockfd = socket(AF_INET, SOCK_DGRAM, 0);
41     if (sockfd < 0) {
42         perror("Socket creation failed");
43         return 1;
44     }
45
46     // 绑定套接字到端口
47     memset(&server_addr, 0, sizeof(server_addr));
48     server_addr.sin_family = AF_INET;
49     server_addr.sin_addr.s_addr = INADDR_ANY;
50     server_addr.sin_port = htons(PORT);
51
52     if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
53         perror("Bind failed");
54         close(sockfd);
55         return 1;
56     }
57
58     printf("Listening for UDP packets on port %d...\n", PORT);
59
60     while (1) {
61         // 接收数据包
62         int recv_len = recvfrom(sockfd, buffer, sizeof(buffer) - 1, 0, (struct
sockaddr *)&client_addr, &addr_len);
63         if (recv_len < 0) {
64             perror("Recvfrom failed");
65             close(sockfd);
66             return 1;
67         }
68
69         // 处理数据包信息
70         buffer[recv_len] = '\0'; // 确保数据末尾为 '\0' (仅适用于文本数据)
71         print_packet_info(&client_addr, &server_addr, buffer, recv_len);
72     }

```

```

73
74     close(sockfd);
75     return 0;
76 }
77

```

正在捕获 VMware Network Adapter VMnet8

文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(I) 帮助(H)

ip.src==192.168.222.20 and ip.dst==192.168.223.20

No.	Time	Source	Destination	Protocol	Length	Info
178	17.214450	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=8
179	17.214786	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=8
947	133.884431	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=7
948	133.884752	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=7

Total Length: 35
 Identification: 0x2f09 (12041)
 > 010. = Flags: 0x2, Don't fragment
 ...0 0000 0000 0000 = Fragment Offset: 0
 Time to Live: 63
 Protocol: UDP (17)
 Header Checksum: 0x0ce2 [validation disabled]
 [Header checksum status: Unverified]
 Source Address: 192.168.222.20
 Destination Address: 192.168.223.20
 [Stream index: 3]
 > User Datagram Protocol, Src Port: 60257, Dst Port: 12345
 > Data (7 bytes)
 Data: 7a68656e78696e
 [Length: 7]

Data (data.data), 7 byte(s)

分组: 1186 · Displayed: 4 (0.3%) 配置: Default

lab4 [SSH 192.168.222.10] [Administrator]

```

valid_lft forever preferred_lft forever
aircraft@root:~/lab4$
Source MAC Address: 0:c:29:3e:1d:f1
Destination MAC Address: 0:c:29:30:37:c
Source IP Address: 192.168.222.20
Destination IP Address: 192.168.223.20
TTL: 64
Data: zhenxin
Hex Data: 7a 68 65 6e 78 69 6e 00 00 00 00 00 00 00 00 00
Send to MAC: 00:0c:29:17:e2:d9
----- Send Packet -----
Time: Sun Nov 3 19:14:36 2024
Source MAC Address: 0:c:29:30:37:16
Destination MAC Address: 0:c:29:17:e2:d9
Source IP Address: 192.168.222.20
Destination IP Address: 192.168.223.20
TTL: 63
Forwarding packet on interface: ens37 (index: 3)
Packet forwarded to 192.168.223.20

```

SSH 192.168.222.10

正在捕获 VMware Network Adapter VMnet8

文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(I) 帮助(H)

ip.src==192.168.222.20 and ip.dst==192.168.223.20

No.	Time	Source	Destination	Protocol	Length	Info
178	17.214450	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=8
179	17.214786	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=8
947	133.884431	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=7
948	133.884752	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=7

> Frame 948: 60 bytes on wire (480 bits), 60 bytes captured
 Encapsulation type: Ethernet (1)
 Arrival Time: Nov 3, 2024 19:14:39.320551000 中国
 UTC Arrival Time: Nov 3, 2024 11:14:39.320551000
 Epoch Arrival Time: 1730632479.320551000
 [Time shift for this packet: 0.000000000 seconds]
 [Time delta from previous captured frame: 0.000321]
 [Time delta from previous displayed frame: 0.00032]
 [Time since reference or first frame: 133.88475200]
 Frame Number: 948
 Frame Length: 60 bytes (480 bits)
 Capture Length: 60 bytes (480 bits)
 [Frame is marked: False]

Search [Administrator]

```

Message sent at Sun Nov 3 19:11:31 2024
From 192.168.222.20:60257 to 192.168.223.20:12345
Data: nihaoma
-----
Message: hh
Message sent at Sun Nov 3 19:11:46 2024
From 192.168.222.20:60257 to 192.168.223.20:12345
Data: hh
-----
Message: zhendema
Message sent at Sun Nov 3 19:12:42 2024
From 192.168.222.20:60257 to 192.168.223.20:12345
Data: zhendema
-----
Message: zhenxin
Message sent at Sun Nov 3 19:14:39 2024
From 192.168.222.20:60257 to 192.168.223.20:12345
Data: zhenxin
-----
Message:

```

SSH 192.168.222.20

Search [Administrator]

```

root@root:/home/aircraft/lab4/yanshou3# ./reciept
Listening for UDP packets on port 12345...

```

SSH 192.168.222.30

Lab4 [SSH: 192.168.222.10] [Administrator]

```
valid_lft forever preferred_lft forever
aircraft@root:~/lab4$
```

Source MAC Address: 08:c:29:3e:5d:f1
Destination MAC Address: 08:c:29:30:37:c
Source IP Address: 192.168.222.20
Destination IP Address: 192.168.223.20
TTL: 64
Data: hhh
Hex Data: 68 68 68 00 00 00 00 00 00 00 00 00 00 00 00 00
Send to MAC: 08:c:29:30:37:c
Time: Sun Nov 3 19:20:55 2024
Source MAC Address: 08:c:29:30:37:16
Destination MAC Address: 08:c:29:17:e2:d9
Source IP Address: 192.168.222.20
Destination IP Address: 192.168.223.20
TTL: 63
Forwarding packet on interface: ens37 (index: 3)
Packet forwarded to 192.168.223.20

正在捕获 VMware Network Adapter VMnet8

ip.src==192.168.222.20 and ip.dst==192.168.223.20

No.	Time	Source	Destination	Protocol	Length	Info
178	17.214450	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=8
179	17.214786	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=8
947	133.884431	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=7
948	133.884752	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=7
3633	512.946755	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=3
3634	512.947100	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=3

[Coloring Rule Name: UDP]
[Coloring Rule String: udp]
Ethernet II, Src: VMware_30:37:16 (08:0c:29:30:37:16), Dst: VMware_17:e2:d9 (08:0c:29:17:e2:d9)
Destination: VMware_17:e2:d9 (08:0c:29:17:e2:d9)
Source: VMware_30:37:16 (08:0c:29:30:37:16)
Type: IPv4 (0x0800)
[Stream index: 4]
Padding: 00000000000000000000000000000000
Internet Protocol Version 4, Src: 192.168.222.20, Dst: 192.168.223.20
User Datagram Protocol, Src Port: 60257, Dst Port: 12345
Data (3 bytes)
Data: 686868
[Length: 3]

Message sent at Sun Nov 3 19:11:46 2024
From 192.168.222.20:60257 to 192.168.223.20:12345
Data: hh
Message: zhendema
Message sent at Sun Nov 3 19:12:42 2024
From 192.168.222.20:60257 to 192.168.223.20:12345
Data: zhendema
Message: zhenxin
Message sent at Sun Nov 3 19:14:39 2024
From 192.168.222.20:60257 to 192.168.223.20:12345
Data: zhenxin
Message: hhh
Message sent at Sun Nov 3 19:20:58 2024
From 192.168.222.20:60257 to 192.168.223.20:12345
Data: hhh
Message:

root@root:/home/aircraft/lab4/yanshou3# ./receipt
Listening for UDP packets on port 12345...

Lab4 [SSH: 192.168.222.10] [Administrator]

```
valid_lft forever preferred_lft forever
aircraft@root:~/lab4$
```

Source MAC Address: 08:c:29:3e:5d:f1
Destination MAC Address: 08:c:29:30:37:c
Source IP Address: 192.168.222.20
Destination IP Address: 192.168.223.20
TTL: 64
Data: hhh
Hex Data: 68 68 68 00 00 00 00 00 00 00 00 00 00 00 00 00
Send to MAC: 08:c:29:30:37:c
Time: Sun Nov 3 19:20:55 2024
Source MAC Address: 08:c:29:30:37:16
Destination MAC Address: 08:c:29:17:e2:d9
Source IP Address: 192.168.222.20
Destination IP Address: 192.168.223.20
TTL: 63
Forwarding packet on interface: ens37 (index: 3)
Packet forwarded to 192.168.223.20

正在捕获 VMware Network Adapter VMnet8

ip.src==192.168.222.20 and ip.dst==192.168.223.20

No.	Time	Source	Destination	Protocol	Length	Info
178	17.214450	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=8
179	17.214786	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=8
947	133.884431	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=7
948	133.884752	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=7
3633	512.946755	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=3
3634	512.947100	192.168.222.20	192.168.223.20	UDP	60	60257 → 12345 Len=3

[Coloring Rule Name: UDP]
[Coloring Rule String: udp]
Ethernet II, Src: VMware_3e:5d:f1 (08:0c:29:3e:5d:f1), Dst: VMware_30:37:0c (08:0c:29:30:37:0c)
Destination: VMware_30:37:0c (08:0c:29:30:37:0c)
Source: VMware_3e:5d:f1 (08:0c:29:3e:5d:f1)
Type: IPv4 (0x0800)
[Stream index: 3]
Padding: 00000000000000000000000000000000
Internet Protocol Version 4, Src: 192.168.222.20, Dst: 192.168.223.20
User Datagram Protocol, Src Port: 60257, Dst Port: 12345
Data (3 bytes)
Data: 686868
[Length: 3]

Message sent at Sun Nov 3 19:11:46 2024
From 192.168.222.20:60257 to 192.168.223.20:12345
Data: hh
Message: zhendema
Message sent at Sun Nov 3 19:12:42 2024
From 192.168.222.20:60257 to 192.168.223.20:12345
Data: zhendema
Message: zhenxin
Message sent at Sun Nov 3 19:14:39 2024
From 192.168.222.20:60257 to 192.168.223.20:12345
Data: zhenxin
Message: hhh
Message sent at Sun Nov 3 19:20:58 2024
From 192.168.222.20:60257 to 192.168.223.20:12345
Data: hhh
Message:

root@root:/home/aircraft/lab4/yanshou3# ./receipt
Listening for UDP packets on port 12345...