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`：

```
ip route
```

如果配置正确，输出可能类似于：

```
default via 192.168.x.x dev ens33
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：

```
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：

```
auto ens33
iface ens33 inet static
    address 192.168.222.131
    netmask 255.255.255.0
    gateway 192.168.222.2
    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：
     ```
     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：

```
sudo systemctl restart networking
```

或者重启虚拟机：

```
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：

```
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

```
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：

```
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**：

   ```
   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：

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

   ```
   ssh-rsa
   AAAAB3NzaC1yc2EAAAADAQABAAACAQDlsQxoyNqrJrO2RuQSmII3sXlMGzA7wsEsIhukUMNELhqULW/JDBN
   ZyryR4LtwONMYXu/F65zwa3IADqZuMrm/0lwAsiGyYcbrAe5Q6flJ9WuoNgn19+VDn+/z/6Ma+mgRzXhm8Z
   B+LMsxjBZ2tfS8GC+vcVQmUBW4O37G4ik5k1Mjah8NAiIlvhpsGPBlQmbujhUll6kiWn3tseyPecMVQmkkA
   DE96AY1KMZ6jERQeX8Eh2sM1TwAZ+TddOIWtqLvo8DTxNcA2m1Dgw2WkN2EfbDr3+OoqPQFHZU4EjG6tvEa
   hWKGUBhG0xCJr5j4OGYUjynCRDMu2kAMGXBOTLNSRxKpV4LCa5vb2bTAN0/v3cjLIMgsCkDbnp6KYYjrTLd
   bubf44YC73APXPfevx8mQs3cbzmb8L1rX366ET4QVOT0TIwWjybbR23sbQUOtUsTfNKpmxkz9cWk8eUQQdq
   3sQVqN8dDB3JI0xUxlJP9F3ozjEjI+t5OsmSf5hz4ElO7WOB6+1Uu5WmfKHJ8nCAyqjCptEUS6aOkXeBsVo
   Em6xc0Uebs9xlUOdvA5/Ycd/yPCn6YXe9j7Hg3AloxYc2dpkyu0fGMR4nlsYq9sK59VTkCYxxjyjODlVzIM
   Fbu8lxu6n69Gis3/Z3PNgqhCnK2otSRW+xMcItoPzGtzsXasVw== 2451752823@qq.com
   ```

2. **登录到远程主机**（使用密码登录）：

   ```
   ssh aircraft@192.168.222.131
   ```

3. **创建或编辑 `~/.ssh/authorized_keys` 文件**：

```
mkdir -p ~/.ssh
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**：

```
chmod 700 ~/.ssh
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

```
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：

```
UDP_SRC_PORT=12345
UDP_DST_PORT=12345
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：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <netinet/ether.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <linux/if_packet.h>

#define BUFFER_SIZE 1518 // 以太网帧最大长度

// 全局变量
int UDP_SRC_PORT;
```

```c
int UDP_DST_PORT;
unsigned char DEST_MAC[6];

void load_config(const char *filename) {
    FILE *file = fopen(filename, "r");
    if (!file) {
        perror("Could not open config file");
        exit(EXIT_FAILURE);
    }

    char line[256];
    while (fgets(line, sizeof(line), file)) {
        // 去掉换行符
        line[strcspn(line, "\n")] = 0;

        // 根据等号分割键值对
        char key[32];
        char value[32];
        if (sscanf(line, "%[^=]=%s", key, value) == 2) {
            if (strcmp(key, "UDP_SRC_PORT") == 0) {
                UDP_SRC_PORT = atoi(value);
            } else if (strcmp(key, "UDP_DST_PORT") == 0) {
                UDP_DST_PORT = atoi(value);
            } else if (strcmp(key, "DEST_MAC") == 0) {
                sscanf(value, "%hhx:%hhx:%hhx:%hhx:%hhx:%hhx",
                       &DEST_MAC[0], &DEST_MAC[1], &DEST_MAC[2],
                       &DEST_MAC[3], &DEST_MAC[4], &DEST_MAC[5]);
            }
        }
    }

    fclose(file);
}

int main() {
    // 先加载配置文件
    load_config("config.txt");

    // 后续代码（例如创建套接字、构造数据包等）...
    int sockfd; // 套接字文件描述符
    struct ifreq if_idx, if_mac; // 用于获取接口索引和MAC地址
    struct sockaddr_ll socket_address; // 套接字地址结构
    char buffer[BUFFER_SIZE]; // 数据缓冲区

    // 创建原始套接字，允许捕获所有以太网数据
    if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) == -1) {
        perror("socket");
        return 1; // 出现错误，退出
    }

    // ... 省略获取接口索引和MAC地址的代码

    // 构造以太网头
    struct ether_header *eh = (struct ether_header *) buffer;
    // 用获取到的MAC地址设置源地址
    memcpy(eh->ether_shost, if_mac.ifr_hwaddr.sa_data, ETH_ALEN);
```

```
    memcpy(eh->ether_dhost, DEST_MAC, ETH_ALEN); // 目标MAC地址
    eh->ether_type = htons(0x0800); // 设置以太网类型字段（0x0800表示IP协议）

    // 后面的代码与之前相同

    // 发送数据包
    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) {
        perror("sendto"); // 发送失败
        return 1;
    }

    close(sockfd); // 关闭套接字
    return 0; // 正常结束

}
```

- **发送端代码**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <netinet/ether.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <linux/if_packet.h>

// 定义目标MAC地址
#define DEST_MAC0 0x00
#define DEST_MAC1 0x0c
#define DEST_MAC2 0x29
#define DEST_MAC3 0x3e
#define DEST_MAC4 0x1e
#define DEST_MAC5 0x4c

#define ETHER_TYPE 0x0800        // 以太网类型字段（0x0800表示IP协议）
#define BUFFER_SIZE 1518         // 以太网帧最大长度
#define UDP_SRC_PORT 12345       // 源UDP端口
#define UDP_DST_PORT 12345       // 目的UDP端口

// 计算校验和的函数
unsigned short checksum(void *b, int len) {
    unsigned short *buf = b;
    unsigned int sum = 0;
    unsigned short result;

    // 对每两个字节进行求和
    for (sum = 0; len > 1; len -= 2)
        sum += *buf++;
    // 如果有剩下的字节，加上它
```

```
    if (len == 1)
        sum += *(unsigned char *)buf;
    // 处理溢出
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    result = ~sum; // 取反作为校验和
    return result;
}

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

    // 创建原始套接字，允许捕获所有以太网数据
    if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) == -1) {
        perror("socket");
        return 1; // 出现错误，退出
    }

    // 获取接口索引
    memset(&if_idx, 0, sizeof(struct ifreq));
    strncpy(if_idx.ifr_name, "eth0", IFNAMSIZ-1); // eth0 为目标网络接口名称
    if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0) {
        perror("SIOCGIFINDEX");  // 获取索引失败
        return 1;
    }

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

    // 构造以太网头
    struct ether_header *eh = (struct ether_header *) buffer;
    memcpy(eh->ether_shost, if_mac.ifr_hwaddr.sa_data, ETH_ALEN); // 设置源MAC地址
    eh->ether_dhost[0] = DEST_MAC0; // 设置目标MAC地址
    eh->ether_dhost[1] = DEST_MAC1;
    eh->ether_dhost[2] = DEST_MAC2;
    eh->ether_dhost[3] = DEST_MAC3;
    eh->ether_dhost[4] = DEST_MAC4;
    eh->ether_dhost[5] = DEST_MAC5;
    eh->ether_type = htons(ETHER_TYPE); // 设置以太网类型字段

    // 构造 IP 头
    struct iphdr *iph = (struct iphdr *) (buffer + sizeof(struct ether_header));
    iph->ihl = 5; // IP头长度
    iph->version = 4; // IPv4
    iph->tos = 0; // 服务类型
    iph->tot_len = htons(sizeof(struct iphdr) + sizeof(struct udphdr) +
strlen("Hello, this is a test message.")); // 总长度
    iph->id = htonl(54321); // 标识符
```

```
    iph->frag_off = 0; // 不分片
    iph->ttl = 255; // 生存时间
    iph->protocol = IPPROTO_UDP; // 协议类型为UDP
    iph->check = 0; // 校验和开始为0
    iph->saddr = inet_addr("192.168.1.2"); // 源IP地址
    iph->daddr = inet_addr("192.168.1.3"); // 目的IP地址
    iph->check = checksum((unsigned short *)iph, sizeof(struct iphdr)); // 计算并设置校验和

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

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

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

    // 发送数据包
    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) {
        perror("sendto"); // 发送失败
        return 1;
    }

    close(sockfd); // 关闭套接字
    return 0; // 正常结束
}
```

- **打印数据**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <netinet/ether.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <linux/if_packet.h>
#include <time.h>
```

```c

#define BUFFER_SIZE 1518 // 以太网帧最大长度

void print_packet_info(const unsigned char *buffer, int size) {
    struct ether_header *eh = (struct ether_header *) buffer; // 以太网头
    struct iphdr *iph = (struct iphdr *) (buffer + sizeof(struct ether_header)); // IP 头

    // 获取当前时间
    time_t now;
    time(&now);
    char *time_str = ctime(&now); // 获取当前时间
    time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符

    // 打印信息
    printf("Received Packet:\n");
    printf("Time: %s\n", time_str);
    printf("Source MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh->ether_shost));
    printf("Destination MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh->ether_dhost));
    printf("Source IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->saddr));
    printf("Destination IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->daddr));

    printf("TTL: %d\n", iph->ttl);
    printf("-------------------------------\n");
}

int main() {
    int sockfd; // 套接字文件描述符
    struct sockaddr saddr; // 套接字地址结构
    unsigned char *buffer = (unsigned char *) malloc(BUFFER_SIZE); // 数据缓冲区
    socklen_t saddr_len = sizeof(saddr);

    // 创建原始套接字，允许捕获所有以太网数据
    if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) == -1) {
        perror("Socket creation failed");
        return 1; // 出现错误，退出
    }

    // 捕获数据包
    while (1) {
        int data_size = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, &saddr, &saddr_len);

        if (data_size < 0) {
            perror("Recvfrom error");
            return 1; // 出现错误，退出
        }

        // 解析并打印数据包信息
        print_packet_info(buffer, data_size);
    }

    close(sockfd); // 关闭套接字
    free(buffer); // 释放缓冲区
    return 0; // 正常结束
```

```
65 }
```

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

```c
int sockfd; // 套接字文件描述符
struct ifreq if_idx, if_mac; // 用于获取接口索引和MAC地址
struct sockaddr_ll socket_address; // 套接字地址结构
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

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

- 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

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

- 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

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

- 再次使用`ioctl()`获取指定接口的MAC地址。

3.2.5 构造以太网头

```
struct ether_header *eh = (struct ether_header *) buffer;
memcpy(eh->ether_shost, if_mac.ifr_hwaddr.sa_data, ETH_ALEN); // 设置源MAC地址
eh->ether_dhost[0] = DEST_MAC0; // 设置目标MAC地址
// 省略其他目标MAC地址部分
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

```
struct iphdr *iph = (struct iphdr *) (buffer + sizeof(struct ether_header));
iph->ihl = 5; // IP头长度
iph->version = 4; // IPv4
// 省略其他IP头设置
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

```
struct udphdr *udph = (struct udphdr *) (buffer + sizeof(struct ether_header) +
sizeof(struct iphdr));
udph->source = htons(UDP_SRC_PORT); // 源端口
udph->dest = htons(UDP_DST_PORT); // 目的端口
udph->len = htons(sizeof(struct udphdr) + strlen("Hello, this is a test message.")); //
UDP报文长度
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

```
char *data = (char *) (buffer + sizeof(struct ether_header) + sizeof(struct iphdr) +
sizeof(struct udphdr));
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

```
socket_address.sll_ifindex = if_idx.ifr_ifindex; // 设置接口索引
socket_address.sll_halen = ETH_ALEN; // MAC地址长度
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

```
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) {
    perror("sendto"); // 发送失败
    return 1;
}
```

- 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

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <netinet/ether.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <linux/if_packet.h>
#include <time.h>

#define BUFFER_SIZE 1518  // 以太网帧最大长度
#define ETHER_TYPE 0x0800 // 以太网类型字段（0x0800表示IP协议）

// 全局变量
int UDP_SRC_PORT;
int UDP_DST_PORT;
char *S_ADDR;
char *D_ADDR;
unsigned char DEST_MAC[6];

void load_config(const char *filename) {
    FILE *file = fopen(filename, "r");
    if (!file) {
        perror("Could not open config file");
        exit(EXIT_FAILURE);
    }

    char line[256];
    while (fgets(line, sizeof(line), file)) {
        // 去掉换行符
        line[strcspn(line, "\n")] = 0;

        // 根据等号分割键值对
        char key[32];
        char value[32];
        if (sscanf(line, "%[^=]=%s", key, value) == 2) {
            if (strcmp(key, "UDP_SRC_PORT") == 0) {
                UDP_SRC_PORT = atoi(value);
            } else if (strcmp(key, "UDP_DST_PORT") == 0) {
                UDP_DST_PORT = atoi(value);
            } else if (strcmp(key, "DEST_MAC") == 0) {
                sscanf(value, "%hhx:%hhx:%hhx:%hhx:%hhx:%hhx",
                       &DEST_MAC[0], &DEST_MAC[1], &DEST_MAC[2],
                       &DEST_MAC[3], &DEST_MAC[4], &DEST_MAC[5]);
            } else if (strcmp(key, "S_ADDR") == 0) {
                S_ADDR = strdup(value); // 分配内存并复制源地址
            } else if (strcmp(key, "D_ADDR") == 0) {
                D_ADDR = strdup(value); // 分配内存并复制目的地址
            }
        }
    }

```

```c
    fclose(file);
}

void print_packet_info(const unsigned char *buffer) {
    struct ether_header *eh = (struct ether_header *)buffer;                    // 以太网头
    struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header)); // IP头

    // 获取当前时间
    time_t now;
    time(&now);
    char *time_str = ctime(&now);          // 获取当前时间
    time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符

    // 打印信息
    printf("Received Packet:\n");
    printf("Time: %s\n", time_str);
    printf("Source MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh->ether_shost));
    printf("Destination MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh->ether_dhost));
    printf("Source IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->saddr));
    printf("Destination IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->daddr));
    printf("TTL: %d\n", iph->ttl);
    printf("--------------------------------\n");
}

// 计算校验和的函数
unsigned short checksum(void *b, int len) {
    unsigned short *buf = b;
    unsigned int sum = 0;
    unsigned short result;

    // 对每两个字节进行求和
    for (sum = 0; len > 1; len -= 2)
        sum += *buf++;
    // 如果有剩下的字节，加上它
    if (len == 1)
        sum += *(unsigned char *)buf;
    // 处理溢出
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    result = ~sum; // 取反作为校验和
    return result;
}

int main() {
    // 先加载配置文件
    load_config("config.txt");

    // 后续代码（例如创建套接字、构造数据包等）...
    int sockfd;                          // 套接字文件描述符
    struct ifreq if_idx, if_mac;         // 用于获取接口索引和MAC地址
    struct sockaddr_ll socket_address; // 套接字地址结构
```

```
    char buffer[BUFFER_SIZE];          // 数据缓冲区

    // 创建原始套接字，允许捕获所有以太网数据
    if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) == -1) {
        perror("socket");
        return 1; // 出现错误，退出
    }

    // 获取接口索引
    memset(&if_idx, 0, sizeof(struct ifreq));
    strncpy(if_idx.ifr_name, "eth0", IFNAMSIZ - 1); // eth0 为目标网络接口名称
    if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0) {
        perror("SIOCGIFINDEX"); // 获取索引失败
        return 1;
    }

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

    // 构造以太网头
    struct ether_header *eh = (struct ether_header *)buffer;
    memcpy(eh->ether_shost, if_mac.ifr_hwaddr.sa_data, ETH_ALEN); // 设置源MAC地址
    memcpy(eh->ether_dhost, DEST_MAC, ETH_ALEN);                  // 目标MAC地址
    eh->ether_type = htons(ETHER_TYPE);                           // 设置以太网类型字段

    // 构造 IP 头
    struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header));
    iph->ihl = 5;
                                // IP头长度
    iph->version = 4;
                                // IPv4
    iph->tos = 0;
                                // 服务类型
    iph->tot_len = htons(sizeof(struct iphdr) + sizeof(struct udphdr) +
strlen("Hello, this is a test message.")); // 总长度
    iph->id = htonl(54321);
                                // 标识符
    iph->frag_off = 0;
                                 // 不分片
    iph->ttl = 255;
                                // 生存时间
    iph->protocol = IPPROTO_UDP;
                                 // 协议类型为UDP
    iph->check = 0;
                                // 校验和开始为0
    iph->saddr = inet_addr(S_ADDR);
                                // 源IP地址
    iph->daddr = inet_addr(D_ADDR);
                                // 目的IP地址
    iph->check = checksum((unsigned short *)iph, sizeof(struct iphdr));
                                // 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>
```

```c
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <netinet/ether.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <linux/if_packet.h>
#include <time.h>

#define BUFFER_SIZE 1518  // 以太网帧最大长度
#define ETHER_TYPE 0x0800 // 以太网类型字段（0x0800表示IP协议）

// 全局变量
int UDP_SRC_PORT;
int UDP_DST_PORT;
char *S_ADDR;
char *D_ADDR;
unsigned char DEST_MAC[6];

void load_config(const char *filename) {
    FILE *file = fopen(filename, "r");
    if (!file) {
        perror("Could not open config file");
        exit(EXIT_FAILURE);
    }

    char line[256];
    while (fgets(line, sizeof(line), file)) {
        // 去掉换行符
        line[strcspn(line, "\n")] = 0;

        // 根据等号分割键值对
        char key[32];
        char value[256]; // 扩大值的大小以支持更长的字符串
        if (sscanf(line, "%[^=]=%s", key, value) == 2) {
            if (strcmp(key, "UDP_SRC_PORT") == 0) {
                UDP_SRC_PORT = atoi(value);
            } else if (strcmp(key, "UDP_DST_PORT") == 0) {
                UDP_DST_PORT = atoi(value);
            } else if (strcmp(key, "DEST_MAC") == 0) {
                sscanf(value, "%hhx:%hhx:%hhx:%hhx:%hhx:%hhx",
                       &DEST_MAC[0], &DEST_MAC[1], &DEST_MAC[2],
                       &DEST_MAC[3], &DEST_MAC[4], &DEST_MAC[5]);
            } else if (strcmp(key, "S_ADDR") == 0) {
                S_ADDR = strdup(value); // 分配内存并复制源地址
            } else if (strcmp(key, "D_ADDR") == 0) {
                D_ADDR = strdup(value); // 分配内存并复制目的地址
            }
        }
    }

    fclose(file);
}

void print_packet_info(const unsigned char *buffer) {
```

```
    struct ether_header *eh = (struct ether_header *)buffer;                    // 以太网头
    struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header)); // IP头
    struct udphdr *udph = (struct udphdr *)(buffer + sizeof(struct ether_header) + sizeof(struct iphdr)); // UDP头
    char *data = (char *)(buffer + sizeof(struct ether_header) + sizeof(struct iphdr) + sizeof(struct udphdr)); // 数据部分

    // 获取当前时间
    time_t now;
    time(&now);
    char *time_str = ctime(&now);          // 获取当前时间
    time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符

    // 打印信息
    printf("Sended Packet:\n");
    printf("Time: %s\n", time_str);
    printf("Source MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh->ether_shost));
    printf("Destination MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh->ether_dhost));
    printf("Source IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->saddr));
    printf("Destination IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->daddr));
    printf("TTL: %d\n", iph->ttl);
    printf("Source Port: %d\n", ntohs(udph->source)); // 源端口
    printf("Destination Port: %d\n", ntohs(udph->dest)); // 目的端口
    printf("Data: %s\n", data); // 打印数据
    printf("--------------------------------\n");
}


// 计算校验和的函数
unsigned short checksum(void *b, int len) {
    unsigned short *buf = b;
    unsigned int sum = 0;
    unsigned short result;

    // 对每两个字节进行求和
    for (sum = 0; len > 1; len -= 2)
        sum += *buf++;
    // 如果有剩下的字节，加上它
    if (len == 1)
        sum += *(unsigned char *)buf;
    // 处理溢出
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    result = ~sum; // 取反作为校验和
    return result;
}

int main() {
    // 先加载配置文件
    load_config("config.txt");

```

```
    // 创建原始套接字，允许捕获所有以太网数据
    int sockfd;
    struct ifreq if_idx, if_mac;       // 用于获取接口索引和MAC地址
    struct sockaddr_ll socket_address; // 套接字地址结构
    char buffer[BUFFER_SIZE];          // 数据缓冲区
    char data[256];                    // 用于存储用户输入的数据

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

    // 获取接口索引
    memset(&if_idx, 0, sizeof(struct ifreq));
    strncpy(if_idx.ifr_name, "ens33", IFNAMSIZ - 1); // eth0 为目标网络接口名称
    if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0) {
        perror("SIOCGIFINDEX"); // 获取索引失败
        return 1;
    }

    // 获取接口 MAC 地址
    memset(&if_mac, 0, sizeof(struct ifreq));
    strncpy(if_mac.ifr_name, "ens33", IFNAMSIZ - 1);
    if (ioctl(sockfd, SIOCGIFHWADDR, &if_mac) < 0) {
        perror("SIOCGIFHWADDR"); // 获取MAC地址失败
        return 1;
    }

    while (1) {
        printf("Enter message to send (type 'exit' to quit): ");
        fgets(data, sizeof(data), stdin);
        data[strcspn(data, "\n")] = 0; // 去掉换行符

        if (strcmp(data, "exit") == 0) {
            break; // 输入"exit"，退出循环
        }

        // 构造以太网头
        struct ether_header *eh = (struct ether_header *)buffer;
        memcpy(eh->ether_shost, if_mac.ifr_hwaddr.sa_data, ETH_ALEN); // 设置源MAC地址
        memcpy(eh->ether_dhost, DEST_MAC, ETH_ALEN);                  // 目标MAC地址
        eh->ether_type = htons(ETHER_TYPE);                           // 设置以太网类
型字段

        // 构造 IP 头
        struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header));
        iph->ihl = 5;
                                    // IP头长度
        iph->version = 4;
                                    // IPv4
        iph->tos = 0;
                                    // 服务类型
        iph->tot_len = htons(sizeof(struct iphdr) + sizeof(struct udphdr) +
strlen(data)); // 总长度
        iph->id = htonl(54321);
                                    // 标识符
```

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

4 路由转发程序

- **代码**

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <netinet/ip.h>
#include <netinet/if_ether.h>
#include <netinet/ether.h>
#include <sys/socket.h>
#include <unistd.h>
#include <linux/if_packet.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <time.h>

#define BUFFER_SIZE 65536 // 定义缓冲区大小

// 计算校验和函数
unsigned short checksum(void *b, int len) {
    unsigned short *buf = b;
    unsigned int sum = 0;
    unsigned short result;

    // 计算校验和
    for (sum = 0; len > 1; len -= 2)
        sum += *buf++;

    // 如果有剩下的字节，加上它
    if (len == 1)
        sum += *(unsigned char *)buf;

    // 处理溢出
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    result = ~sum; // 取反作为校验和
    return result;
}

int main() {
    int sockfd;
    struct sockaddr saddr;
    unsigned char *buffer = (unsigned char *)malloc(BUFFER_SIZE);

    // 创建原始套接字
    sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP));
    if (sockfd < 0) {
        perror("Socket creation failed");
        return 1;
    }

    while (1) {
        int saddr_len = sizeof(saddr);
        // 接收数据包
```

```
        int data_size = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, &saddr,
(socklen_t*)&saddr_len);
        if (data_size < 0) {
            perror("Recvfrom error");
            return 1;
        }

        // 解析以太网头和 IP 头
        struct ethhdr *eth_header = (struct ethhdr *)buffer;
        struct iphdr *ip_header = (struct iphdr *)(buffer + sizeof(struct ethhdr));
        char src_ip[INET_ADDRSTRLEN];
        char dest_ip[INET_ADDRSTRLEN];

        // 转换 IP 地址格式
        inet_ntop(AF_INET, &(ip_header->saddr), src_ip, INET_ADDRSTRLEN);
        inet_ntop(AF_INET, &(ip_header->daddr), dest_ip, INET_ADDRSTRLEN);

        // 判断源和目的 IP 是否符合要求
        if (strcmp(src_ip, "192.168.1.1") == 0 && strcmp(dest_ip, "192.168.1.3") ==
0) {
            // 获取当前系统时间
            time_t rawtime;
            struct tm *timeinfo;
            char time_str[100];

            time(&rawtime);
            timeinfo = localtime(&rawtime);

            // 格式化时间字符串
            strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S", timeinfo);

            // 打印信息
            printf("[%s] Captured packet from %s to %s\n", time_str, src_ip,
dest_ip);

            // 修改 TTL
            ip_header->ttl -= 1;
            ip_header->check = 0; // 重置校验和
            ip_header->check = checksum((unsigned short *)ip_header, ip_header->ihl *
4); // 计算新校验和

            // 发送数据包到目的主机
            struct ifreq ifr, ifr_mac;
            struct sockaddr_ll dest;

            // 获取网卡接口索引
            memset(&ifr, 0, sizeof(ifr));
            snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), "eth0");
            if (ioctl(sockfd, SIOCGIFINDEX, &ifr) < 0) {
                perror("ioctl");
                return 1;
            }

            // 获取网卡接口 MAC 地址
            memset(&ifr_mac, 0, sizeof(ifr_mac));
            snprintf(ifr_mac.ifr_name, sizeof(ifr_mac.ifr_name), "eth0");
```

```
            if (ioctl(sockfd, SIOCGIFHWADDR, &ifr_mac) < 0) {
                perror("ioctl");
                return 1;
            }

            // 设置目标 MAC 地址（假设目标地址已知）
            unsigned char target_mac[ETH_ALEN] = {0x00, 0x0c, 0x29, 0x48, 0xd3,
0xf7}; // 替换为实际的目标 MAC 地址
            memset(&dest, 0, sizeof(dest));
            dest.sll_ifindex = ifr.ifr_ifindex;
            dest.sll_halen = ETH_ALEN;
            memcpy(dest.sll_addr, target_mac, ETH_ALEN); // 复制目标 MAC 地址

            // 构造新的以太网帧头
            memcpy(eth_header->h_dest, target_mac, ETH_ALEN); // 目标 MAC 地址
            memcpy(eth_header->h_source, ifr_mac.ifr_hwaddr.sa_data, ETH_ALEN); // 源
MAC 地址
            eth_header->h_proto = htons(ETH_P_IP); // 以太网类型为 IP

            printf("Interface name: %s, index: %d\n", ifr.ifr_name, ifr.ifr_ifindex);

            // 发送数据包
            if (sendto(sockfd, buffer, data_size, 0, (struct sockaddr *)&dest,
sizeof(dest)) < 0) {
                perror("Sendto error");
                return 1;
            }
            printf("Datagram forwarded.\n");
        } else {
            printf("Ignored packet from %s to %s\n", src_ip, dest_ip);
        }
    }

    close(sockfd);
    free(buffer);
    return 0;
}

```

- 接收路由表的版本

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <netinet/ip.h>
#include <netinet/if_ether.h>
#include <netinet/ether.h>
#include <sys/socket.h>
#include <unistd.h>
#include <linux/if_packet.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <time.h>
#include <netinet/udp.h>

```

```
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
```

```
            strcpy(routing_table[route_count].src_ip, src_ip);
            strcpy(routing_table[route_count].dest_ip, dest_ip);

            // 解析 MAC 地址
            if (sscanf(mac_str, "%hhx:%hhx:%hhx:%hhx:%hhx:%hhx",
                       &routing_table[route_count].target_mac[0],
                       &routing_table[route_count].target_mac[1],
                       &routing_table[route_count].target_mac[2],
                       &routing_table[route_count].target_mac[3],
                       &routing_table[route_count].target_mac[4],
                       &routing_table[route_count].target_mac[5]) == 6)
            {
                printf("Successfully added route: %s -> %s with MAC: %02x:%02x:%02x:%02x:%02x:%02x\n",
                       routing_table[route_count].src_ip,
                       routing_table[route_count].dest_ip,
                       routing_table[route_count].target_mac[0],
                       routing_table[route_count].target_mac[1],
                       routing_table[route_count].target_mac[2],
                       routing_table[route_count].target_mac[3],
                       routing_table[route_count].target_mac[4],
                       routing_table[route_count].target_mac[5]);
                route_count++;
            }
            else
            {
                fprintf(stderr, "Invalid MAC address format: %s\n", mac_str);
            }
        }
        else
        {
            fprintf(stderr, "Invalid routing table entry: %s\n", line);
        }
    }
    fclose(file);
}

// 打印路由表
void print_routing_table()
{
    printf("Routing Table:\n");
    printf("SRC IP\t\tDEST IP\t\tTARGET MAC\n");
    for (int i = 0; i < route_count; i++)
    {
        printf("%s\t%s\t%02x:%02x:%02x:%02x:%02x:%02x\n",
               routing_table[i].src_ip,
               routing_table[i].dest_ip,
               routing_table[i].target_mac[0], routing_table[i].target_mac[1],
               routing_table[i].target_mac[2], routing_table[i].target_mac[3],
               routing_table[i].target_mac[4], routing_table[i].target_mac[5]);
    }
}

void print_packet_info(const unsigned char *buffer) {
    struct ether_header *eh = (struct ether_header *)buffer;                    // 以太网头
```

```
    struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header)); // IP头
    struct udphdr *udph = (struct udphdr *)(buffer + sizeof(struct ether_header) + sizeof(struct iphdr)); // UDP头
    char *data = (char *)(buffer + sizeof(struct ether_header) + sizeof(struct iphdr) + sizeof(struct udphdr)); // 数据部分

    // 获取当前时间
    time_t now;
    time(&now);
    char *time_str = ctime(&now);          // 获取当前时间
    time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符

    // 打印信息
    printf("Sended Packet:\n");
    printf("Time: %s\n", time_str);
    printf("Source MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh->ether_shost));
    printf("Destination MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh->ether_dhost));
    printf("Source IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->saddr));
    printf("Destination IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->daddr));
    printf("TTL: %d\n", iph->ttl);
    printf("Source Port: %d\n", ntohs(udph->source)); // 源端口
    printf("Destination Port: %d\n", ntohs(udph->dest)); // 目的端口
    printf("Data: %s\n", data); // 打印数据
    printf("--------------------------------\n");
}

int main() {
    int sockfd;
    struct sockaddr saddr;
    unsigned char *buffer = (unsigned char *)malloc(BUFFER_SIZE);

    // Load routing table from configuration file
    load_routing_table("config.txt");

    // Print routing table
    print_routing_table();

    sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP));
    if (sockfd < 0) {
        perror("Socket creation failed");
        return 1;
    }

    while (1) {
        int saddr_len = sizeof(saddr);
        int data_size = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, &saddr, (socklen_t*)&saddr_len);
        if (data_size < 0) {
            perror("Recvfrom error");
            return 1;
        }

```

```
        struct ethhdr *eth_header = (struct ethhdr *)buffer;
        struct iphdr *ip_header = (struct iphdr *)(buffer + sizeof(struct ethhdr));
        char src_ip[INET_ADDRSTRLEN];
        char dest_ip[INET_ADDRSTRLEN];

        inet_ntop(AF_INET, &(ip_header->saddr), src_ip, INET_ADDRSTRLEN);
        inet_ntop(AF_INET, &(ip_header->daddr), dest_ip, INET_ADDRSTRLEN);

        for (int i = 0; i < route_count; i++) {
            if (strcmp(src_ip, routing_table[i].src_ip) == 0 && strcmp(dest_ip, routing_table[i].dest_ip) == 0) {
                // 获取当前系统时间
                time_t rawtime;
                struct tm *timeinfo;
                char time_str[100];

                time(&rawtime);
                timeinfo = localtime(&rawtime);
                strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S", timeinfo);

                // 打印信息
                printf("[%s] Captured packet from %s to %s\n", time_str, src_ip, dest_ip);
                print_packet_info(buffer);

                // 修改 TTL
                ip_header->ttl -= 1;
                ip_header->check = 0;
                ip_header->check = checksum((unsigned short *)ip_header, ip_header->ihl * 4);

                // 发送数据包到目的主机
                struct ifreq ifr;
                struct sockaddr_ll dest;

                // 获取网卡接口索引
                memset(&ifr, 0, sizeof(ifr));
                snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), "ens33");
                if (ioctl(sockfd, SIOCGIFINDEX, &ifr) < 0) {
                    perror("ioctl");
                    return 1;
                }

                memset(&dest, 0, sizeof(dest));
                dest.sll_ifindex = ifr.ifr_ifindex;
                dest.sll_halen = ETH_ALEN;
                memcpy(dest.sll_addr, routing_table[i].target_mac, ETH_ALEN); // 使用路由表中的目标MAC地址

                // 构造新的以太网帧头
                memcpy(eth_header->h_dest, routing_table[i].target_mac, ETH_ALEN);
                memcpy(eth_header->h_source, ifr.ifr_hwaddr.sa_data, ETH_ALEN);
                eth_header->h_proto = htons(ETH_P_IP);

                if (sendto(sockfd, buffer, data_size, 0, (struct sockaddr *)&dest, sizeof(dest)) < 0) {
```

```
                    perror("Sendto error");
                    return 1;
                }
                printf("Datagram forwarded to %s\n", routing_table[i].dest_ip);
                break;
            } else {
                // printf("Ignored packet from %s to %s\n", src_ip, dest_ip);
            }
        }
    }

    close(sockfd);
    free(buffer);
    return 0;
}
```

5 接受主机程序

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <netinet/ip.h>
#include <netinet/if_ether.h>
#include <netinet/ether.h>
#include <sys/socket.h>
#include <unistd.h>
#include <linux/if_packet.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <time.h>
#include <netinet/udp.h>

#define PORT 54321

void print_packet_info(const unsigned char *buffer) {
    struct ether_header *eh = (struct ether_header *)buffer;                    // 以太网头
    struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header)); // IP头
    struct udphdr *udph = (struct udphdr *)(buffer + sizeof(struct ether_header) + sizeof(struct iphdr)); // UDP头
    char *data = (char *)(buffer + sizeof(struct ether_header) + sizeof(struct iphdr) + sizeof(struct udphdr)); // 数据部分

    // 获取当前时间
    time_t now;
    time(&now);
    char *time_str = ctime(&now);          // 获取当前时间
    time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符

    // 打印信息
    printf("Sended Packet:\n");
    printf("Time: %s\n", time_str);
```

```
    printf("Source MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh->ether_shost));
    printf("Destination MAC Address: %s\n", ether_ntoa((struct ether_addr *)eh->ether_dhost));
    printf("Source IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->saddr));
    printf("Destination IP Address: %s\n", inet_ntoa(*(struct in_addr *)&iph->daddr));
    printf("TTL: %d\n", iph->ttl);
    printf("Source Port: %d\n", ntohs(udph->source)); // 源端口
    printf("Destination Port: %d\n", ntohs(udph->dest)); // 目的端口
    printf("Data: %s\n", data); // 打印数据
    printf("--------------------------------\n");
}

int main() {
    int sockfd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_len = sizeof(client_addr);
    char buffer[1024];

    // 创建 UDP 套接字
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // 绑定套接字到端口
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Bind failed");
        return 1;
    }

    // 接收数据包
    int recv_len = recvfrom(sockfd, buffer, sizeof(buffer) - 1, 0, (struct sockaddr *)&client_addr, &addr_len);
    if (recv_len < 0) {
        perror("Recvfrom failed");
        return 1;
    }

    buffer[recv_len] = '\0';
    printf("Received message: %s\n", buffer);

    close(sockfd);
    return 0;
}
```

- 可循环接收版本

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

```c
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <netinet/udp.h>
#include <sys/socket.h>
#include <unistd.h>
#include <time.h>

#define PORT 54321
#define BUFFER_SIZE 1024

int main() {
    int sockfd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_len = sizeof(client_addr);
    char buffer[BUFFER_SIZE];

    // 创建 UDP 套接字
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        return EXIT_FAILURE;
    }

    // 绑定套接字到端口
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Bind failed");
        close(sockfd);
        return EXIT_FAILURE;
    }

    // 循环接收数据包
    printf("Server is listening on port %d...\n", PORT);
    while (1) {
        // 清空接收缓冲区
        memset(buffer, 0, sizeof(buffer));

        // 接收数据包
        int recv_len = recvfrom(sockfd, buffer, sizeof(buffer) - 1, 0, (struct sockaddr *)&client_addr, &addr_len);
        if (recv_len < 0) {
            perror("Recvfrom failed");
            break; // 发生错误时退出循环
        }

        // 获取当前时间
        time_t now = time(NULL);
        char *time_str = ctime(&now); // 获取当前时间
        if (time_str) {
            time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符
        }
```

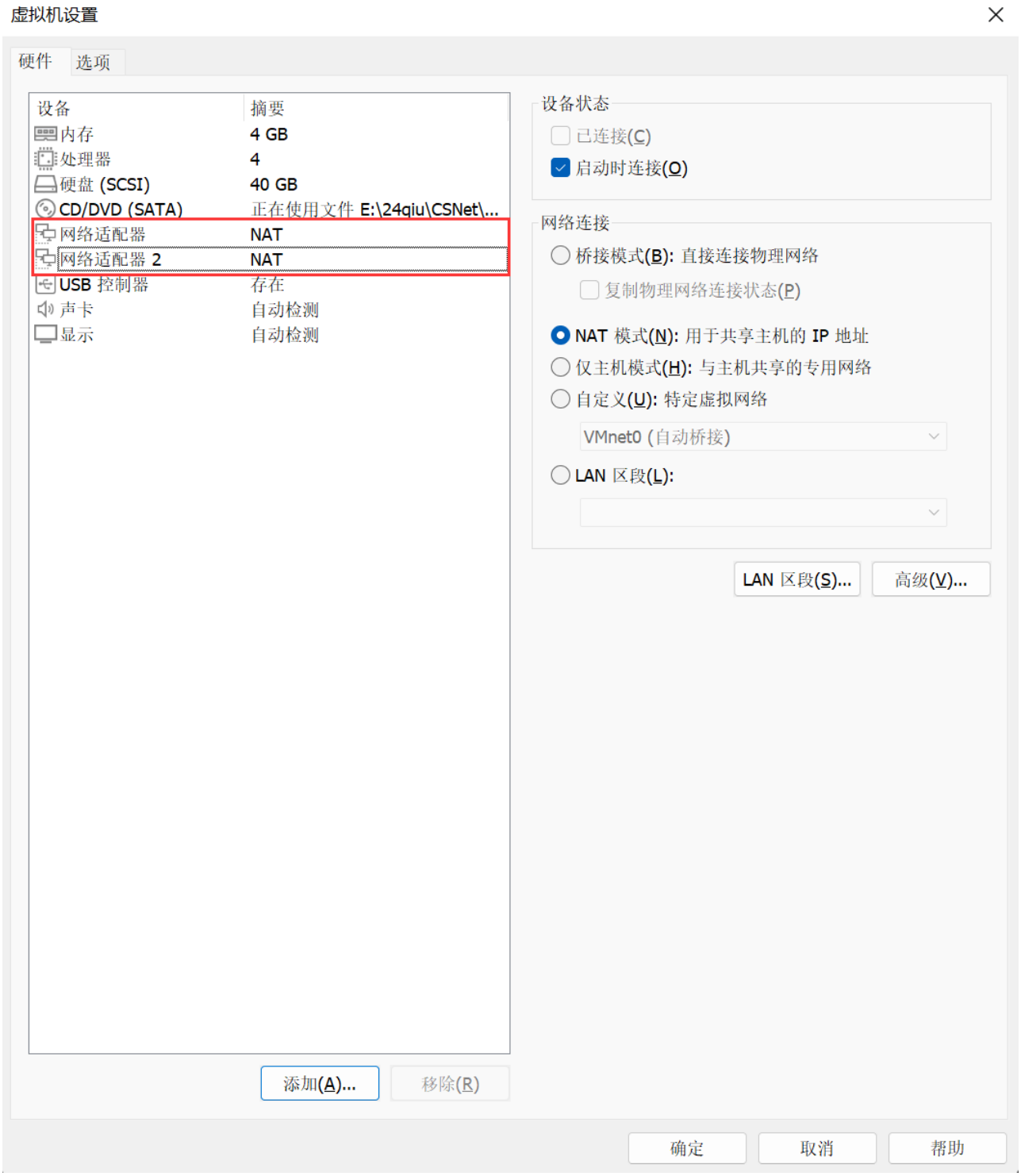
```
57
58          buffer[recv_len] = '\0'; // 确保字符串结束
59
60          // 打印信息
61          printf("Received Packet:\n");
62          printf("Time: %s\n", time_str);
63          printf("Sender: %s:%d\n", inet_ntoa(client_addr.sin_addr),
    ntohs(client_addr.sin_port));
64          printf("Message: %s\n", 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。


虚拟机设置
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
添加(A)...
移除(R)
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)...
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#
```

```
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

```

```
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

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

- 主机1

```
auto ens33
iface ens33 inet static
    address 192.168.222.20
    netmask 255.255.255.0
    gateway 192.168.222.10
```

- 主机2

```
auto ens33
iface ens33 inet static
    address 192.168.223.20
    netmask 255.255.255.0
    gateway 192.168.223.10

auto ens37
iface ens37 inet static
    address 192.168.222.30
    netmask 255.255.255.0
    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）：

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

2. **添加到 `192.168.223.0/24` 网络的路由**：

```
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**：

```
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

```
sudo ip route add 192.168.222.0/24 via 192.168.223.10
```

2.3 添加默认网关

```
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：

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>
#include <time.h>

#define DEST_IP "192.168.223.20"
#define DEST_PORT 12345
#define BUFFER_SIZE 1024

int main() {
    int sockfd;
    struct sockaddr_in dest_addr, local_addr;
    char message[BUFFER_SIZE];
    socklen_t addr_len = sizeof(local_addr);

    // 创建 UDP 套接字
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // 设置目的地址
    memset(&dest_addr, 0, sizeof(dest_addr));
    dest_addr.sin_family = AF_INET;
    dest_addr.sin_port = htons(DEST_PORT);
    inet_pton(AF_INET, DEST_IP, &dest_addr.sin_addr);

    printf("Enter messages to send (type 'exit' to quit):\n");

    while (1) {
        printf("Message: ");
        fgets(message, BUFFER_SIZE, stdin);

        // 去掉换行符
        message[strcspn(message, "\n")] = 0;

        // 检查是否输入 "exit" 来退出
        if (strcmp(message, "exit") == 0) {
            printf("Exiting...\n");
            break;
        }

        // 发送数据包
        if (sendto(sockfd, message, strlen(message), 0, (struct sockaddr *)&dest_addr,
sizeof(dest_addr)) < 0) {
            perror("Sendto failed");
            return 1;
        }
```

```
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
```

```c
struct route_entry route_table[1]; // 只声明一个空的数组项
int route_table_size = 1; // 路由表大小

unsigned short checksum(unsigned short *buf, int nwords) {
    unsigned long sum;
    for (sum = 0; nwords > 0; nwords--)
        sum += *buf++;
    sum = (sum >> 16) + (sum & 0xffff);
    sum += (sum >> 16);
    return (unsigned short)(~sum);
}

struct route_entry *lookup_route(uint32_t dest_ip) {
    for (int i = 0; i < route_table_size; i++) {
        if ((dest_ip & route_table[i].netmask) == (route_table[i].dest & route_table[i].netmask)) {
            return &route_table[i];
        }
    }
    return NULL;
}


void print_packet_info(const unsigned char *buffer, int data_size) {
    struct ether_header *eth_header = (struct ether_header *)buffer; // 以太网头
    struct iphdr *ip_header = (struct iphdr *)(buffer + sizeof(struct ether_header)); // IP头
    char *data = (char *)(buffer + sizeof(struct ether_header) + ip_header->ihl * 4 + sizeof(struct udphdr)); // 数据部分

    // 获取当前时间
    time_t now;
    time(&now);
    char *time_str = ctime(&now);
    time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符

    // 计算数据部分的大小
    int data_length = data_size - (data - (char *)buffer); // 数据部分的长度

    // 创建一个临时数组以存储数据
    char temp[data_length + 1];  // +1 用于字符串终止符

    // 将数据复制到临时数组
    memcpy(temp, data, data_length);
    temp[data_length] = '\0'; // 添加字符串结束符

    // 打印信息
    printf("-------- Received Packet ------------\n");
    printf("Time: %s\n", time_str);
    printf("Source MAC Address: %s\n", ether_ntoa((struct ether_addr *)eth_header->ether_shost));
    printf("Destination MAC Address: %s\n", ether_ntoa((struct ether_addr *)eth_header->ether_dhost));
    printf("Source IP Address: %s\n", inet_ntoa(*(struct in_addr *)&ip_header->saddr));
```

```
    printf("Destination IP Address: %s\n", inet_ntoa(*(struct in_addr *)&ip_header->daddr));
    printf("TTL: %d\n", ip_header->ttl);
    printf("Data: %s\n", temp); // 打印数据内容

    // 打印数据的十六进制表示
    printf("Hex Data: ");
    for (int i = 0; i < data_length; i++) {
        printf("%02x ", (unsigned char)temp[i]);
    }
    printf("\n");
    printf("-------------------------------------\n");
}

void print_send_packet_info(const unsigned char *buffer, int data_size) {
    struct ether_header *eth_header = (struct ether_header *)buffer; // 以太网头
    struct iphdr *ip_header = (struct iphdr *)(buffer + sizeof(struct ether_header)); // IP头
    char *data = (char *)(buffer + sizeof(struct ether_header) + ip_header->ihl * 4 + sizeof(struct udphdr)); // 数据部分

    // 获取当前时间
    time_t now;
    time(&now);
    char *time_str = ctime(&now);
    time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符

    // 计算数据部分的大小
    int data_length = data_size - (data - (char *)buffer); // 数据部分的长度

    // 创建一个临时数组以存储数据
    char temp[data_length + 1];  // +1 用于字符串终止符

    // 将数据复制到临时数组
    memcpy(temp, data, data_length);
    temp[data_length] = '\0'; // 添加字符串结束符

    // 打印信息
    printf("-------- Send Packet ------------\n");
    printf("Time: %s\n", time_str);
    printf("Source MAC Address: %s\n", ether_ntoa((struct ether_addr *)eth_header->ether_shost));
    printf("Destination MAC Address: %s\n", ether_ntoa((struct ether_addr *)eth_header->ether_dhost));
    printf("Source IP Address: %s\n", inet_ntoa(*(struct in_addr *)&ip_header->saddr));
    printf("Destination IP Address: %s\n", inet_ntoa(*(struct in_addr *)&ip_header->daddr));
    printf("TTL: %d\n", ip_header->ttl);
}

int main() {
    int sockfd;
    struct sockaddr saddr;
    unsigned char *buffer = (unsigned char *)malloc(BUFFER_SIZE);
    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));
```

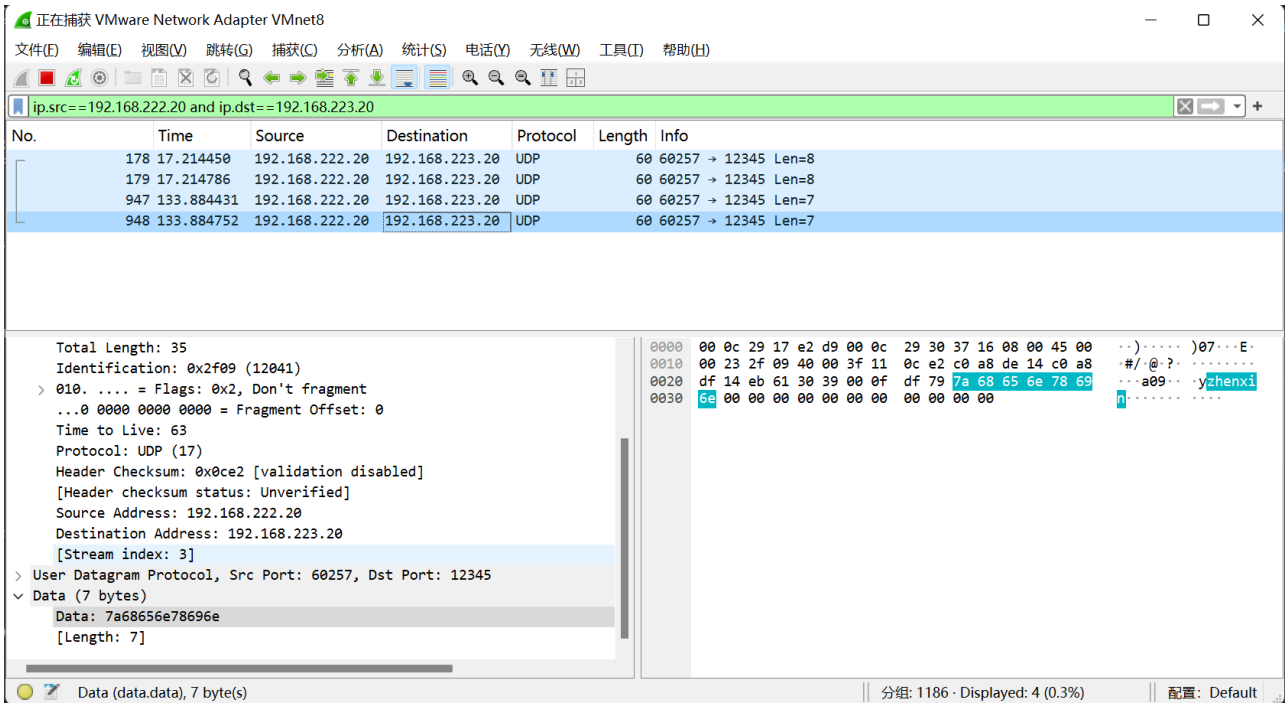
```
        dest.sll_ifindex = ifr.ifr_ifindex;
        dest.sll_halen = ETH_ALEN;
        memcpy(dest.sll_addr, target_mac, ETH_ALEN);
        // 打印目标 MAC 地址
        printf("Send to MAC: %02x:%02x:%02x:%02x:%02x:%02x\n",
               target_mac[0], target_mac[1], target_mac[2],
               target_mac[3], target_mac[4], target_mac[5]);

        // 设置以太网帧头
        struct ether_header *eth_header = (struct ether_header *)buffer;
        memcpy(eth_header->ether_dhost, target_mac, ETH_ALEN);
        memcpy(eth_header->ether_shost, ifr_mac.ifr_hwaddr.sa_data, ETH_ALEN);
        eth_header->ether_type = htons(ETH_P_IP);

        print_send_packet_info(buffer,data_size);
        printf("Forwarding packet on interface: %s (index: %d)\n", ifr.ifr_name,
ifr.ifr_ifindex);

        // 发送数据包
        if (sendto(sockfd, buffer, data_size, 0, (struct sockaddr *)&dest,
sizeof(dest)) < 0) {
            perror("Sendto error");
            return 1;
        }

        printf("Packet forwarded to %s\n", inet_ntoa(*(struct in_addr *)&ip_header-
>daddr));
        printf("------------------------------------\n");
    }

    close(sockfd);
    free(buffer);
    return 0;
}
```

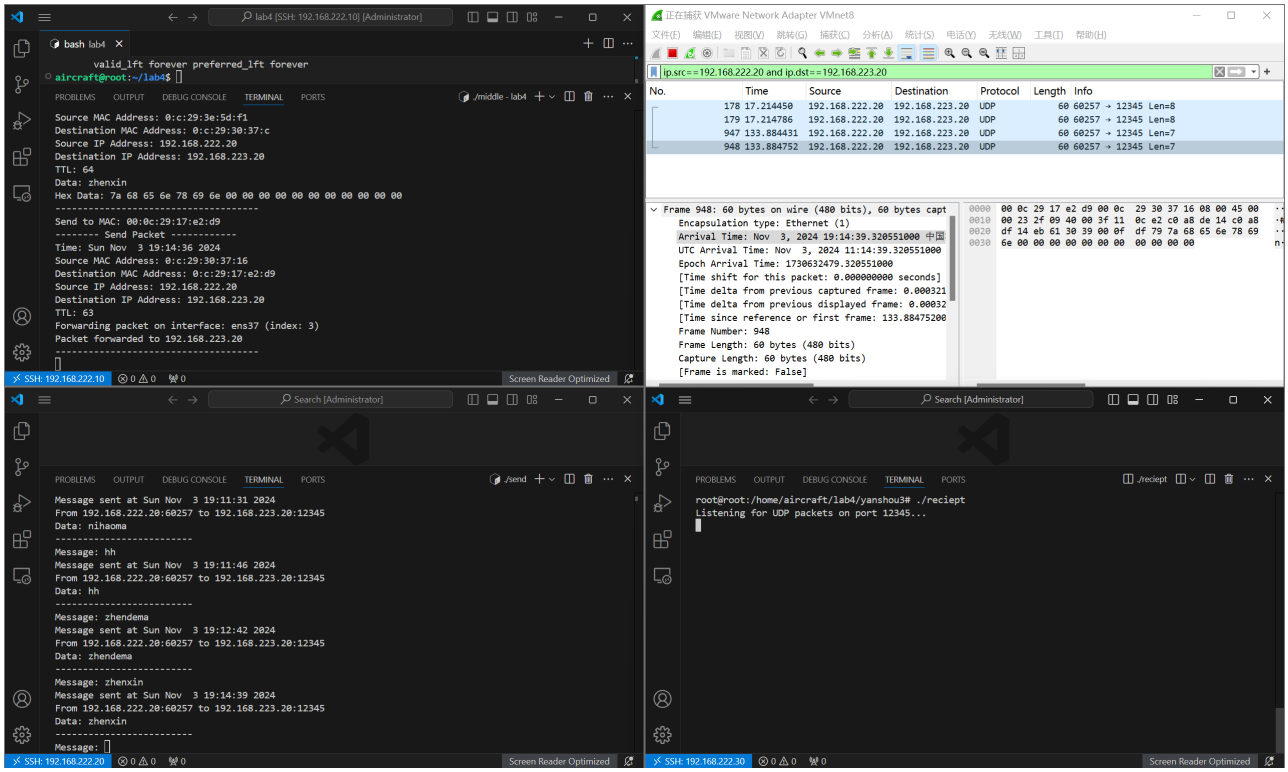
5 接收主机程序（192.168.223.20）

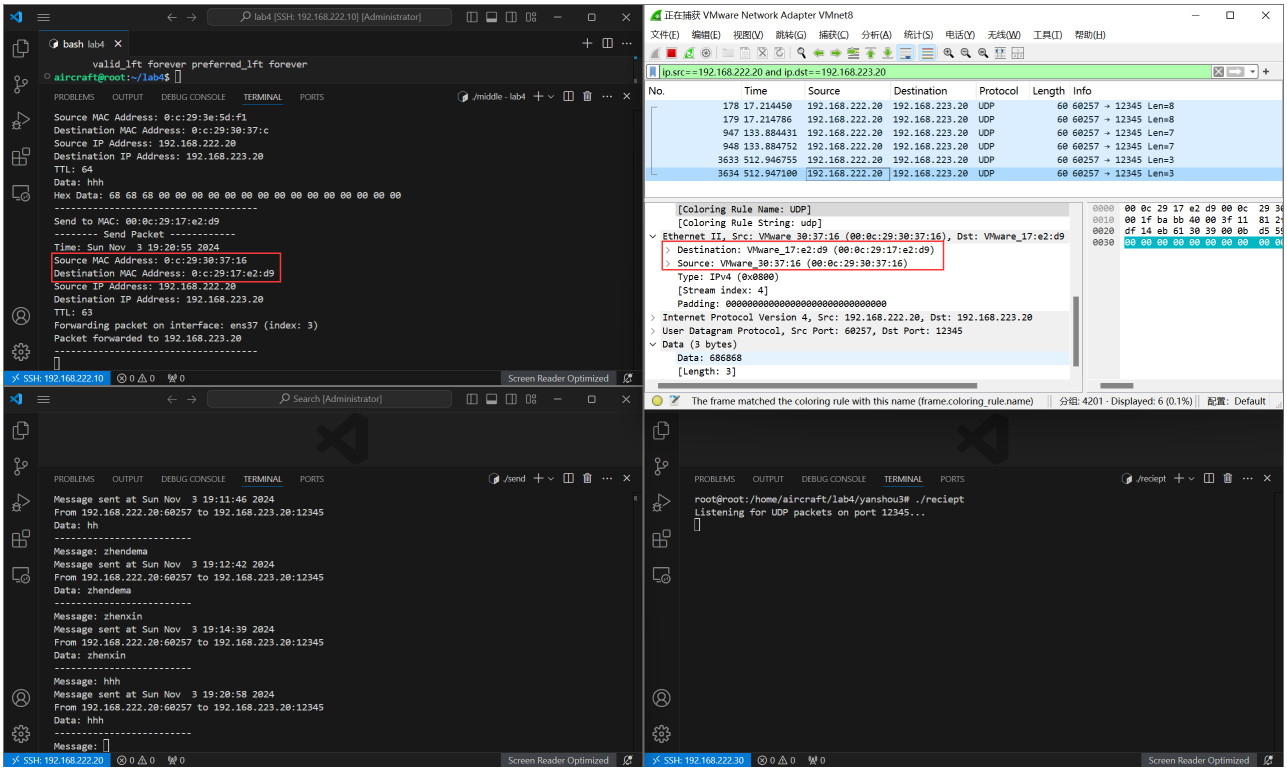
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>
#include <time.h>

#define PORT 12345

void print_packet_info(const struct sockaddr_in *client_addr, const struct sockaddr_in
*server_addr, const char *buffer, int recv_len) {
    // 获取当前时间
    time_t now;
    time(&now);
    char *time_str = ctime(&now);
    time_str[strlen(time_str) - 1] = '\0'; // 去掉换行符

```

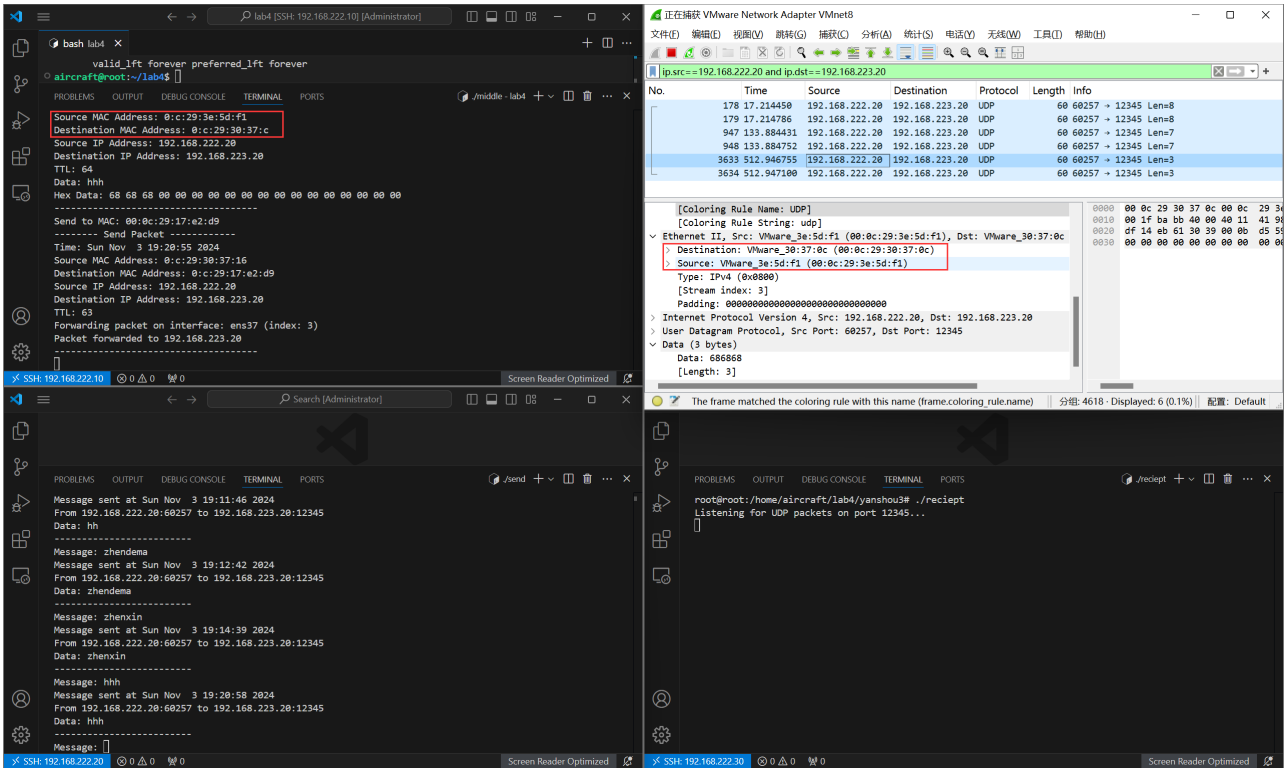
```c
    // 打印接收的包信息
    printf("-------- Received Packet ------------\n");
    printf("Time: %s\n", time_str);
    printf("Source IP Address: %s\n", inet_ntoa(client_addr->sin_addr));
    printf("Source Port: %d\n", ntohs(client_addr->sin_port));
    printf("Destination IP Address: %s\n", inet_ntoa(server_addr->sin_addr));
    printf("Destination Port: %d\n", ntohs(server_addr->sin_port));

    printf("Data (hex): ");
    for (int i = 0; i < recv_len; i++) {
        printf("%02x ", (unsigned char)buffer[i]);
    }
    printf("\n---------------------------------------\n");
}

int main() {
    int sockfd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_len = sizeof(client_addr);
    char buffer[1024];

    // 创建 UDP 套接字
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // 绑定套接字到端口
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Bind failed");
        close(sockfd);
        return 1;
    }

    printf("Listening for UDP packets on port %d...\n", PORT);

    while (1) {
        // 接收数据包
        int recv_len = recvfrom(sockfd, buffer, sizeof(buffer) - 1, 0, (struct
sockaddr *)&client_addr, &addr_len);
        if (recv_len < 0) {
            perror("Recvfrom failed");
            close(sockfd);
            return 1;
        }

        // 处理数据包信息
        buffer[recv_len] = '\0'; // 确保数据末尾为 '\0'（仅适用于文本数据）
        print_packet_info(&client_addr, &server_addr, buffer, recv_len);
    }
```

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






valid_lft forever preferred_lft forever
aircraft@root:~/lab4$
Source MAC Address: 0:c:29:3e:5d: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: hhh
Hex Data: 68 68 68 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Send to MAC: 00:0c:29:17:e2:d9
-------- Send Packet -----------
Time: Sun Nov 3 19:20:55 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
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 (00:0c:29:30:37:16), Dst: VMware_17:e2:d9
Destination: VMware_17:e2:d9 (00:0c:29:17:e2:d9)
Source: VMware_30:37:16 (00:0c: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]
The frame matched the coloring rule with this name (frame.coloring_rule.name)
分组: 4201 · Displayed: 6 (0.1%)
配置: Default
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# ./reciept
Listening for UDP packets on port 12345...


valid_lft forever preferred_lft forever
aircraft@root:~/lab4$
Source MAC Address: 0:c:29:3e:5d: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: hhh
Hex Data: 68 68 68 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Send to MAC: 00:0c:29:17:e2:d9
-------- Send Packet -----------
Time: Sun Nov 3 19:20:55 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
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 (00:0c:29:3e:5d:f1), Dst: VMware_30:37:0c
Destination: VMware_30:37:0c (00:0c:29:30:37:0c)
Source: VMware_3e:5d:f1 (00:0c: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]
The frame matched the coloring rule with this name (frame.coloring_rule.name)
分组: 4618 · Displayed: 6 (0.1%)
配置: Default
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# ./reciept
Listening for UDP packets on port 12345...