



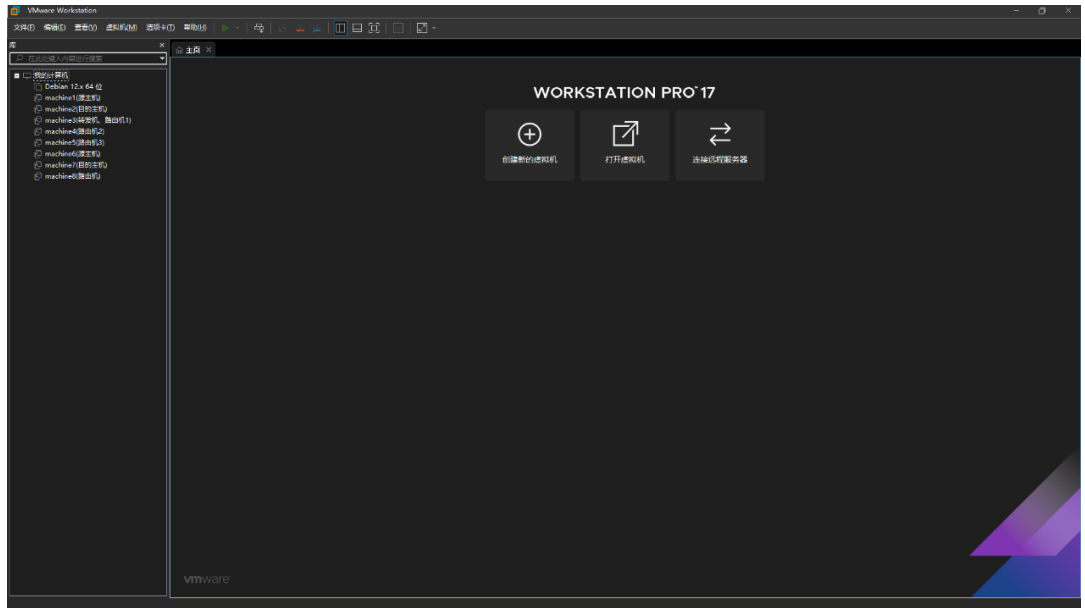
哈尔滨工业大学  
Harbin Institute of Technology

# 计算机网络 课程实验报告

实验名称	IP 数据报的收发与转发					
姓名	杨明达		院系	未来技术学院		
班级	22R0311		学号	2022110829		
任课教师	李全龙		指导教师	李全龙、郭勇		
实验地点	G001		实验时间	周六 5、6 节		
实验课表现	出勤、表现得分(10)		实验报告 得分(40)		实验总分	
	操作结果得分(50)					
教师评语						



计算机科学与技术学院 SINCE 1956...  
School of Computer Science and Technology

实验目的：
了解原始套接字的基本概念和使用方法；掌握路由器进行 IP 数据报转发的基本原理；实现于原始套接字的 IP 数据报的发送和接收；实现基于原始套接字的 IP 数据报转发，包括 AF_INET 和 AF_PACKET 原始套接字的应用。
实验内容：
<p>(1) 使用虚拟机实现多主机间的 UDP 数据报收发及转发</p> <p>利用虚拟机搭建实验环境，掌握 Linux 下的 Socket 网络编程。</p> <p><b>选做 1：</b>改进程序，示例程序只实现了一个数据包（携带 1 条消息）的发、转、收过程，要求实现每条消息由<b>控制台输入</b>，并且<b>不限制</b>发送消息的数目。</p> <p>(2) 基于单网口主机的 IP 数据转发及收发</p> <p>在局域网中，<b>模拟</b> IP 数据报的路由转发过程。通过原始套接字实现了完整的数据封装过程，实现了 UDP 头部、IP 头部、MAC 帧头部的构造。</p> <p><b>选做 2：</b>扩展实验的网络规模，由原始方案中 3 台主机增加到<b>不少于 5 台</b>主机，共同完成 IP 数据报转发及收发过程，要求采用<b>转发表</b>改进示例程序，增加程序通用性。</p> <p>(3) 基于双网口主机的路由转发</p> <p>构造了静态路由表，并实现了不同子网间的 IP 数据报查表转发过程。</p> <p><b>选做 3：</b>通过<b>完善路由表</b>，改进示例程序实现<b>双向传输</b>。</p>
实验过程：
<p>一、实验环境配置</p> <p>使用虚拟机软件 VMware 在 Windows 笔记本电脑上创建 Linux 虚拟机，从而模拟出你所需的网络环境。以下是实现的具体步骤：</p> <p>(1) 安装虚拟机软件：下载并安装 VMware Workstation Player。</p> 
<p>图 1-1 VMware Workstation Player 界面</p> <p>(2) 下载 Linux 发行版：选择一个适合的 Linux 发行版 Debian，并下载 ISO 文件。</p>



#### (4) 配置虚拟机

##### 1) 开启虚拟机，选择 Start installer 选项

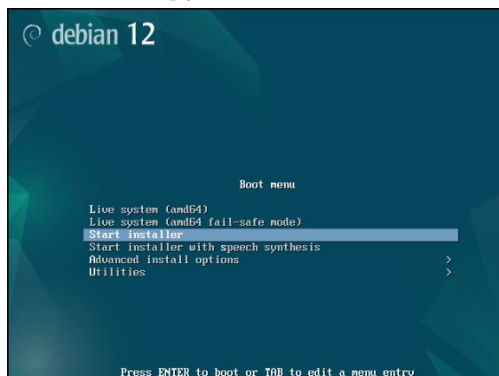


图 1-5 Boot menu 界面

2) 在后续配置中，依次选择 English->United States->American English->debian->domain name 及相关密码->Eastern->Guided-use entire disk->SCSI33->All files in one partition->Finish partitioning and write changes to disk->Yes->Yes->China->mirrors.tuna.tsinghua.edu.cn->Yes->/dev/sda。创建成功后输入相关 login 信息及密码，即可进入命令行界面。

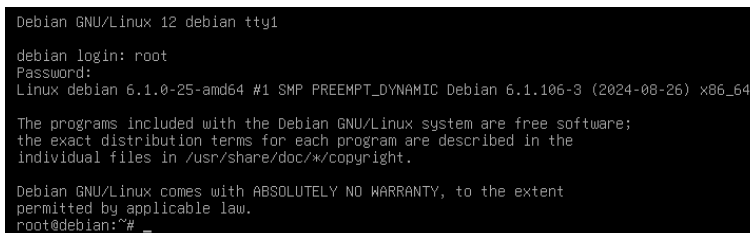


图 1-6 创建成功界面

#### (5) 部署实验环境：在 Linux 虚拟机中安装所需的网络编程实验软件和工具。

##### 1) 更新包列表

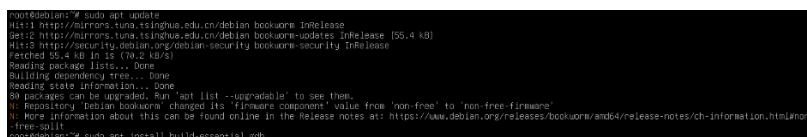


图 1-7 更新包列表

##### 2) 安装GCC、GDB和其他开发工具

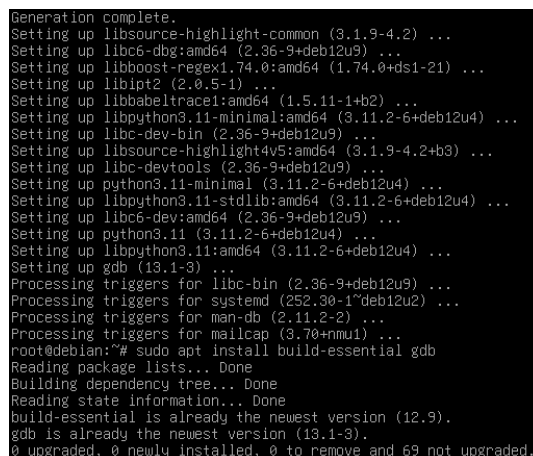


图 1-8 安装 GCC、GDB

图 1-13 检查 SSH 服务状态

5) 配置防火墙

```
root@debian:~# sudo ufw allow ssh
Rules updated
Rules updated (v6)
root@debian:~# sudo ufw status
Status: inactive
```

图 1-14 配置防火墙

6) 连接到SSH服务器

显示所有命令，选择Remote-SSH: Connect to Host



图 1-15 SSH 连接（1）

添加新的SSH Host

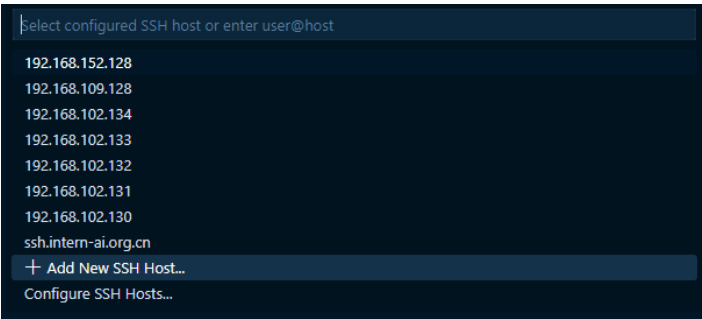


图 1-16 SSH 连接（2）

进行SSH连接

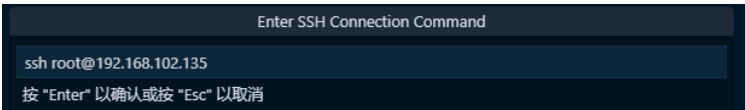


图 1-17 SSH 连接（3）

加SSH信息加入config文件中

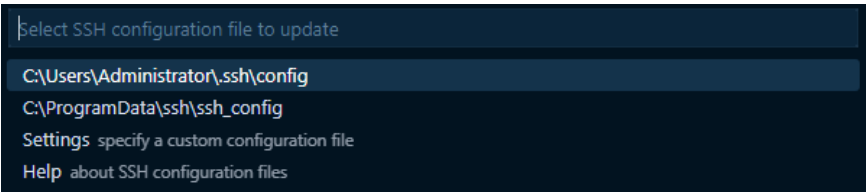


图 1-18 SSH 连接（4）

进行连接。



图 1-19 SSH 连接（5）

## 二、使用虚拟机实现多主机间的UDP数据报收发及转发

### 2.1 环境配置

采用 UDP 原始套接字，将 UDP 数据报由 A 主机发送给 B 主机，再由 B 主机转发给 C 主机，完成了一条消息的传送。注意，此实现方案，UDP 数据报的传输过程中，目的IP 地址发生了变化，因此属于代理转发的场景。

创建一台Linux虚拟机之后，通过虚拟机克隆得到三台Linux虚拟机，构成如下实验环境。其中源主机的IP地址是192.168.102.130，MAC地址是00:0C:29:E2:D5:BE；目的主机的IP地址是192.168.102.131，MAC地址是00:0C:29:CF:92:CA；转发机的IP地址是192.168.102.132，MAC地址是00:0C:29:9B:08:90。

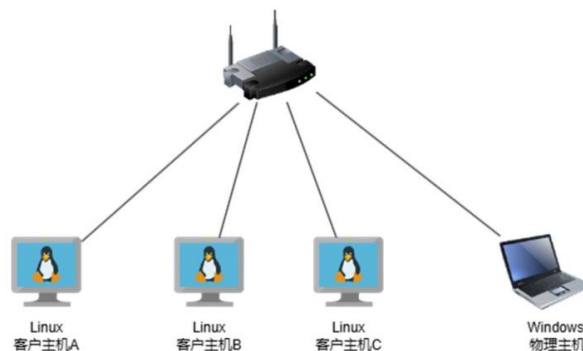


图 2-1 网络拓扑图

### 2.2 必做内容

采用UDP原始套接字，将一条消息由A主机发送给B主机，再由B主机转发给C主机。

#### (1) 发送IP数据报

- 1) 创建一个文件send\_ip.c
- 2) 编写以下代码实现IP数据报的发送功能

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <arpa/inet.h>
5. #include <sys/socket.h>
6. int main()
7. {
8.     int sockfd;
9.     struct sockaddr_in dest_addr;
10.    char *message = "Hello, this is a UDP datagram!";
11.    int port = 12345; // 目标端口号
12.    // 创建 UDP 套接字
13.    if ((sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
```

```
14. {
15.     perror("socket");
16.     return 1;
17. }
18. // 目标地址
19. dest_addr.sin_family = AF_INET;
20. dest_addr.sin_port = htons(port);
21. dest_addr.sin_addr.s_addr = inet_addr("192.168.102.132");//目标 IP 地址
22. // 发送数据报
23. if (sendto(sockfd, message, strlen(message), 0, (struct sockaddr *)&dest
    _addr,
24.         sizeof(dest_addr)) < 0)
25. {
26.     perror("sendto");
27.     return 1;
28. }
29. printf("Datagram sent.\n");
30. return 0;
31. }
```

3) 使用GCC编译并运行程序: gcc -o send\_ip send\_ip.c

运行: ./send\_ip

(2) 转发IP数据报

1) 创建一个文件forward\_ip.c

2) 编写以下代码以实现IP数据报的转发功能

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <arpa/inet.h>
5. #include <sys/socket.h>
6. int main()
7. {
8.     int sockfd;
9.     struct sockaddr_in src_addr, dest_addr, my_addr;
10.    char buffer[1024];
11.    socklen_t addr_len;
12.    int src_port = 12345; // 原始端口号
13.    int dest_port = 54321; // 目标端口号 (接收程序的端口号)
14.    // 创建 UDP 套接字
15.    if ((sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
16.    {
17.        perror("socket");
```



```

18.     return 1;
19. }
20. // 本地地址
21. my_addr.sin_family = AF_INET;
22. my_addr.sin_port = htons(src_port);
23. my_addr.sin_addr.s_addr = INADDR_ANY;
24. // 绑定套接字到本地地址
25. if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr)) < 0)
26. {
27.     perror("bind");
28.     return 1;
29. }
30. // 接收数据报
31. addr_len = sizeof(src_addr);
32. if (recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *)&src_
    addr,
33.         &addr_len) < 0)
34. {
35.     perror("recvfrom");
36.     return 1;
37. }
38. printf("Datagram received: %s\n", buffer);
39. // 修改目标地址为接收程序主机的 IP 地址
40. dest_addr.sin_family = AF_INET;
41. dest_addr.sin_port = htons(dest_port);
42. dest_addr.sin_addr.s_addr = inet_addr("192.168.102.131"); // 替换为接收程
    序主机的实际 IP 地址
43. // 发送数据报
44. if (sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr *)&dest_a
    ddr,
45.         sizeof(dest_addr)) < 0)
46. {
47.     perror("sendto");
48.     return 1;
49. }
50. printf("Datagram forwarded.\n");
51. return 0;
52. }

```

3) 使用GCC编译并运行程序: gcc -o forward\_ip forward\_ip.c

运行: ./forward\_ip

(3) 接收IP数据报

1) 创建一个文件recv\_ip.c

2) 编写一下代码以实现IP数据报的接收功能

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <arpa/inet.h>
4. #include <sys/socket.h>
5. int main()
6. {
7.     int sockfd;
8.     struct sockaddr_in src_addr, my_addr;
9.     char buffer[1024];
10.    socklen_t addr_len;
11.    int port = 54321; // 修改后的接收端口号
12.    // 创建 UDP 套接字
13.    if ((sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
14.    {
15.        perror("socket");
16.        return 1;
17.    }
18.    // 本地地址
19.    my_addr.sin_family = AF_INET;
20.    my_addr.sin_port = htons(port);
21.    my_addr.sin_addr.s_addr = INADDR_ANY;
22.    // 绑定套接字到本地地址
23.    if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr)) < 0)
24.    {
25.        perror("bind");
26.        return 1;
27.    }
28.    // 接收数据报
29.    addr_len = sizeof(src_addr);
30.    if (recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *)&src_
        addr,
31.                &addr_len) < 0)
32.    {
33.        perror("recvfrom");
34.        return 1;
35.    }
36.    printf("Datagram received: %s\n", buffer);
37.    return 0;
38. }
```

3) 使用GCC编译并运行程序: gcc -o recv\_ip recv\_ip.c

运行: ./recv\_ip

## 2.3 选做内容

示例程序只实现了一个数据包（携带1条消息）的发、转、收过程，要求实现每条消息由控制台输入，并且不限制发送消息的数目。

(1) 发送IP数据报

- 1) 创建一个文件send\_ip\_advanced.c
- 2) 编写一下代码以实现IP数据报的发送功能

```
1. #include <stdio.h>
2. #include <time.h>
3. #include <stdlib.h>
4. #include <string.h>
5. #include <arpa/inet.h>
6. #include <sys/socket.h>
7.
8. #define UDP_SRC_PORT 12345
9. #define UDP_FWD_PORT 12345
10. #define UDP_DST_PORT 12345
11. #define UDP_SRC_IP "192.168.102.130"
12. #define UDP_FWD_IP "192.168.102.132"
13. #define UDP_DST_IP "192.168.102.131"
14.
15. int main()
16. {
17.     // 创建 UDP 套接字
18.     int sockfd;
19.     if ((sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0){
20.         perror("Socket");
21.         return 1;
22.     } else {
23.         printf("[SENDER:%10u]: Create Socket successfully.\n", time(0));
24.     }
25.
26.     // 目标地址
27.     struct sockaddr_in fwd_addr;
28.     fwd_addr.sin_family = AF_INET;
29.     fwd_addr.sin_port = htons(UDP_FWD_PORT);
30.     fwd_addr.sin_addr.s_addr = inet_addr(UDP_FWD_IP);
31.
32.     char message[256];
33.     memset(message, 0, sizeof(message));
34.     while(1){
35.         printf("[SENDER:%10u] Please input message: ", time(0));
36.         fgets(message, sizeof(message), stdin);
37.
38.         message[strlen(message) - 1] = 0;
```

```
39.  
40.    // 发送数据报  
41.    socklen_t addr_len = sizeof(fwd_addr);  
42.  
43.    if (sendto(sockfd, message, strlen(message), 0,  
44.              (struct sockaddr *)&fwd_addr, addr_len) < 0) {  
45.        perror("Sendto");  
46.        return 1;  
47.    }  
48.  
49.    printf("[SENDER:%10u] Datagram sent: [%s](%d).\n",  
50.          time(0), message, strlen(message));  
51. }  
52. return 0;  
53. }
```

3) 使用GCC编译并运行程序: gcc -o send\_ip\_advanced send\_ip\_advanced.c

运行: ./send\_ip\_advanced

(2) 转发IP数据报

1) 创建一个文件forward\_ip\_advanced.c

2) 编写一下代码以实现IP数据报的转发功能

```
1. #include <stdio.h>  
2. #include <time.h>  
3. #include <stdlib.h>  
4. #include <string.h>  
5. #include <arpa/inet.h>  
6. #include <sys/socket.h>  
7.  
8. #define UDP_SRC_PORT 12345  
9. #define UDP_FWD_PORT 12345  
10. #define UDP_DST_PORT 12345  
11. #define UDP_SRC_IP "192.168.102.130"  
12. #define UDP_FWD_IP "192.168.102.132"  
13. #define UDP_DST_IP "192.168.102.131"  
14.  
15. int main()  
16. {  
17.    // 创建 UDP 套接字  
18.    int sockfd;  
19.    if ((sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0) {  
20.        perror("Socket");  
21.        return 1;  
22.    }
```

```
23.
24. // 转发地址
25. struct sockaddr_in fwd_addr;
26. fwd_addr.sin_family = AF_INET;
27. fwd_addr.sin_port = htons(UDP_FWD_PORT);
28. fwd_addr.sin_addr.s_addr = inet_addr(UDP_FWD_IP);
29.
30. // 目标地址
31. struct sockaddr_in dst_addr;
32. dst_addr.sin_family = AF_INET;
33. dst_addr.sin_port = htons(UDP_DST_PORT);
34. dst_addr.sin_addr.s_addr = inet_addr(UDP_DST_IP);
35.
36. // 绑定套接字到本地地址
37. if (bind(sockfd, (struct sockaddr *)&fwd_addr, sizeof(fwd_addr)) < 0) {
38.     perror("Bind");
39.     return 1;
40. } else {
41.     printf("[FORWARD:%10u] Bind socket successfully.\n", time(0));
42. }
43.
44. char message[1024];
45. memset(message, 0, sizeof(message));
46. while(1){
47.     // 接收数据报
48.     struct sockaddr_in src_addr;
49.     socklen_t addr_len1 = sizeof(src_addr);
50.
51.     int rcv_len = recvfrom(sockfd, message, sizeof(message),
52.                             0, (struct sockaddr *)&src_addr, &addr_len1);
53.     if (rcv_len < 0) {
54.         perror("Recvfrom");
55.         return 1;
56.     }
57.     message[rcv_len] = '\0';
58.     printf("[FORWARD:%10u] Datagram received: [%s](%d).\n",
59.            time(0), message, strlen(message));
60.
61.     // 发送数据报
62.     socklen_t addr_len2 = sizeof(dst_addr);
63.     if (sendto(sockfd, message, strlen(message), 0,
64.                (struct sockaddr *)&dst_addr, addr_len2) < 0) {
65.         perror("Sendto");
```

```
66.     return 1;
67. }
68.     printf("[FORWARD:%10u] Datagram forwarded: [%s](%d).\n",
69.             time(0), message, strlen(message));
70. }
71.     return 0;
72. }
```

3) 使用GCC编译并运行程序: `gcc -o forward_ip_advanced forward_ip_advanced.c`

运行: `./forward_ip_advanced`

(3) 接收IP数据报

1) 创建一个文件`recv_ip_advanced.c`

2) 编写一下代码以实现IP数据报的接收功能

```
1. #include <stdio.h>
2. #include <time.h>
3. #include <stdlib.h>
4. #include <string.h>
5. #include <arpa/inet.h>
6. #include <sys/socket.h>
7.
8. #define UDP_SRC_PORT 12345
9. #define UDP_FWD_PORT 12345
10. #define UDP_DST_PORT 12345
11. #define UDP_SRC_IP "192.168.102.130"
12. #define UDP_FWD_IP "192.168.102.132"
13. #define UDP_DST_IP "192.168.102.131"
14.
15. int main()
16. {
17.     // 创建 UDP 套接字
18.     int sockfd;
19.     if ((sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0) {
20.         perror("Socket");
21.         return 1;
22.     }
23.
24.     // 目标地址
25.     struct sockaddr_in dst_addr;
26.     dst_addr.sin_family = AF_INET;
27.     dst_addr.sin_port = htons(UDP_DST_PORT);
28.     dst_addr.sin_addr.s_addr = inet_addr(UDP_DST_IP);
29.
30.     // 绑定套接字到本地地址
```

```
31. if (bind(sockfd, (struct sockaddr *)&dst_addr, sizeof(dst_addr)) < 0) {
32.     perror("Bind");
33.     return 1;
34. } else {
35.     printf("[RECVER:%10u] Bind socket successfully.\n", time(0));
36. }
37.
38. char message[1024];
39. memset(message, 0, sizeof(message));
40. while(1){
41.     // 接收数据报
42.     struct sockaddr_in src_addr;
43.     socklen_t addr_len = sizeof(src_addr);
44.
45.     int recv_len = recvfrom(sockfd, message, sizeof(message), 0,
46.                             (struct sockaddr *)&src_addr, &addr_len);
47.     if (recv_len < 0) {
48.         perror("Recvfrom");
49.         return 1;
50.     }
51.     message[recv_len] = '\0';
52.     printf("[RECVER:%10u] Datagram received: [%s](%d).\n",
53.           time(0), message, strlen(message));
54. }
55. return 0;
56. }
```

- 3) 使用GCC编译并运行程序: gcc -o recv\_ip\_advanced recv\_ip\_advanced.c  
运行: ./recv\_ip\_advanced

### 三、基于单网口主机的IP数据报转发及收发

#### 3.1 环境配置

网络拓扑图如下图所示。

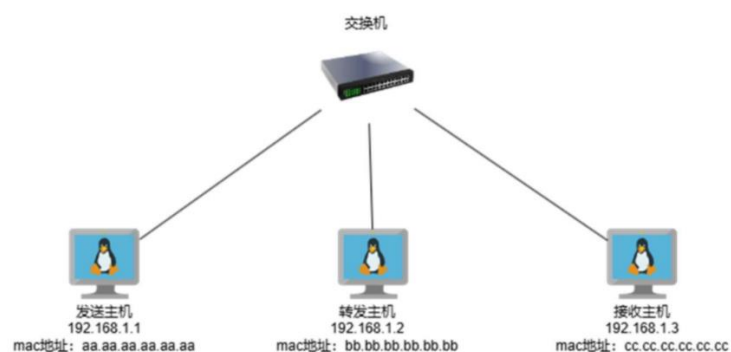


图 3-1 网络拓扑图

数据包的处理流程为：

(1) 源主机发送数据包

- 1) 源主机发送的数据包通过交换机到达路由器。
- 2) 数据包的 IP 头部源地址为 192.168.1.2，目的地址为 192.168.1.3。
- 3) 数据包在封装到下层的以太网帧的时候，进行了特殊处理，即构造的以太网帧的目的 MAC 地址为路由主机 192.168.1.2 的 MAC 地址 bb.bb.bb.bb.bb.bb，以使数据报被交给路由主机，进而转发到目的主机。

(2) 路由器处理数据包

- 1) 路由器接收到数据包后，解析 IP 头部信息。
- 2) 修改 TTL（生存时间）字段，并重新计算校验和。
- 3) 根据路由表(示例程序未实现路由表)决定将数据包转发到目的主机。

(3) 目的主机接收数据包

- 1) 目的主机接收到数据包并处理。

创建一台Linux虚拟机之后，通过虚拟机克隆得到三台Linux虚拟机，构成如下实验环境。其中源主机的IP地址是192.168.102.130，MAC地址是00:0C:29:E2:D5:BE；目的主机的IP地址是192.168.102.131，MAC地址是00:0C:29:CF:92:CA；路由机1的IP地址是192.168.102.132，MAC地址是00:0C:29:9B:08:90。

### 3.2 必做内容

在局域网中，模拟 IP 数据报的路由转发过程。通过原始套接字实现了完整的数据封装过程，实现了 UDP 头部、IP 头部、MAC 帧头部的构造。

(1) 发送主机程序

- 1) 创建一个文件mysend.c
- 2) 编写以下代码实现发送功能

```
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. #define DEST_MAC0 0x00
14. #define DEST_MAC1 0x0c
15. #define DEST_MAC2 0x29
16. #define DEST_MAC3 0x9b
17. #define DEST_MAC4 0x08
18. #define DEST_MAC5 0x90
19. #define ETHER_TYPE 0x0800
20. #define BUFFER_SIZE 1518
```



```
21. #define UDP_SRC_PORT 12345
22. #define UDP_DST_PORT 12345
23. unsigned short checksum(void *b, int len)
24. {
25.
26.     unsigned short *buf = b;
27.     unsigned int sum = 0;
28.     unsigned short result;
29.     for (sum = 0; len > 1; len -= 2)
30.         sum += *buf++;
31.     if (len == 1)
32.         sum += *(unsigned char *)buf;
33.     sum = (sum >> 16) + (sum & 0xFFFF);
34.     sum += (sum >> 16);
35.     result = ~sum;
36.     return result;
37. }
38. int main()
39. {
40.     int sockfd;
41.     struct ifreq if_idx, if_mac;
42.     struct sockaddr_ll socket_address;
43.     char buffer[BUFFER_SIZE];
44.     //char msg[] = "Hello, this is a test message.hitymd";
45.     char msg[] = "Hello, this is a test message.111111111";
46.     // 创建原始套接字
47.     if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) == -1)
48.     {
49.         perror("socket");
50.         return 1;
51.     }
52.     // 获取接口索引
53.     memset(&if_idx, 0, sizeof(struct ifreq));
54.     strncpy(if_idx.ifr_name, "ens33", IFNAMSIZ - 1);
55.     if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0)
56.     {
57.         perror("SIOCGIFINDEX");
58.         return 1;
59.     }
60.     // 获取接口 MAC 地址
61.     memset(&if_mac, 0, sizeof(struct ifreq));
62.     strncpy(if_mac.ifr_name, "ens33", IFNAMSIZ - 1);
63.     if (ioctl(sockfd, SIOCGIFHWADDR, &if_mac) < 0)
64.     {
```

```
65.     perror("SIOCGIFHWADDR");
66.     return 1;
67. }
68. // 构造以太网头
69. struct ether_header *eh = (struct ether_header *)buffer;
70. eh->ether_shost[0] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[0];
71. eh->ether_shost[1] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[1];
72. eh->ether_shost[2] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[2];
73. eh->ether_shost[3] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[3];
74. eh->ether_shost[4] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[4];
75. eh->ether_shost[5] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[5];
76.
77. for (int i = 0; i < 6; i++)
78. {
79.     printf("%02x ", eh->ether_shost[i]);
80. }
81. printf("\n");
82.
83. eh->ether_dhost[0] = DEST_MAC0;
84. eh->ether_dhost[1] = DEST_MAC1;
85. eh->ether_dhost[2] = DEST_MAC2;
86. eh->ether_dhost[3] = DEST_MAC3;
87. eh->ether_dhost[4] = DEST_MAC4;
88. eh->ether_dhost[5] = DEST_MAC5;
89. eh->ether_type = htons(ETHER_TYPE);
90. // 构造 IP 头
91. struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header));
92. iph->ihl = 5;
93. iph->version = 4;
94. iph->tos = 0;
95. iph->tot_len = htons(sizeof(struct iphdr) + sizeof(struct udphdr) + strlen(msg));
96. iph->id = htonl(54321);
97. iph->frag_off = 0;
98. iph->ttl = 255;
99. iph->protocol = IPPROTO_UDP;
100. iph->check = 0;
101. iph->saddr = inet_addr("192.168.102.130");//源主机
102. iph->daddr = inet_addr("192.168.102.131");//目的主机
103. iph->check = checksum((unsigned short *)iph, sizeof(struct iphdr));
104. // 构造 UDP 头
105. struct udphdr *udph = (struct udphdr *)(buffer + sizeof(struct ether_header))
```

```
106.             + sizeof(struct iphdr));
107.   udph->source = htons(UDP_SRC_PORT);
108.   udph->dest = htons(UDP_DST_PORT);
109.   udph->len = htons(sizeof(struct udphdr) + strlen(msg));
110.   udph->check = 0; // UDP 校验和可选
111.   // 填充数据
112.   char *data = (char *)(buffer + sizeof(struct ether_header)
113.   + sizeof(struct iphdr) + sizeof(struct udphdr));
114.   strcpy(data, msg);
115.   // 设置 socket 地址结构
116.   socket_address.sll_ifindex = if_idx.ifr_ifindex;
117.   socket_address.sll_halen = ETH_ALEN;
118.   socket_address.sll_addr[0] = DEST_MAC0;
119.   socket_address.sll_addr[1] = DEST_MAC1;
120.   socket_address.sll_addr[2] = DEST_MAC2;
121.   socket_address.sll_addr[3] = DEST_MAC3;
122.   socket_address.sll_addr[4] = DEST_MAC4;
123.   socket_address.sll_addr[5] = DEST_MAC5;
124.   // 发送数据包
125.   int len = sizeof(struct ether_header) + sizeof(struct iphdr)
126.             + sizeof(struct udphdr) + strlen(msg);
127.   printf("len=%d\n", len);
128.
129.   if (sendto(sockfd, buffer, len, 0, (struct sockaddr *)&socket_address
130.             , sizeof(struct sockaddr_ll)) < 0)
131.   {
132.     perror("sendto");
133.     return 1;
134.   }
135.   close(sockfd);
136.   return 0;
137. }
```

3) 使用GCC编译并运行程序: gcc -o mysend mysend.c

运行: ./mysend

(2) 路由转发程序

1) 创建一个文件myroute.c

2) 编写以下代码实现转发功能

```
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. #define BUFFER_SIZE 65536
15. unsigned short checksum(void *b, int len)
16. {
17.     unsigned short *buf = b;
18.     unsigned int sum = 0;
19.     unsigned short result;
20.     for (sum = 0; len > 1; len -= 2)
21.         sum += *buf++;
22.     if (len == 1)
23.         sum += *(unsigned char *)buf;
24.     sum = (sum >> 16) + (sum & 0xFFFF);
25.     sum += (sum >> 16);
26.     result = ~sum;
27.     return result;
28. }
29. int main()
30. {
31.     int sockfd;
32.     struct sockaddr saddr;
33.     unsigned char *buffer = (unsigned char *)malloc(BUFFER_SIZE);
34.     sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP));
35.     if (sockfd < 0)
36.     {
37.         perror("Socket creation failed");
38.         return 1;
39.     }
40.     while (1)
41.     {
42.         int saddr_len = sizeof(saddr);
43.         int data_size = recvfrom(sockfd, buffer, BUFFER_SIZE,
44.                                 0, &saddr, (socklen_t *)&saddr_len);
45.         if (data_size < 0)
46.         {
47.             perror("Recvfrom error");
48.             return 1;
```

```
49.     }
50.     struct ethhdr *eth_header = (struct ethhdr *)buffer;
51.     struct iphdr *ip_header = (struct iphdr *)(buffer + sizeof(struct ethh
dr));
52.     char src_ip[INET_ADDRSTRLEN];
53.     char dest_ip[INET_ADDRSTRLEN];
54.     inet_ntop(AF_INET, &(ip_header->saddr), src_ip, INET_ADDRSTRLEN);
55.     inet_ntop(AF_INET, &(ip_header->daddr), dest_ip, INET_ADDRSTRLEN);
56.     if (strcmp(src_ip, "192.168.102.130") == 0 && strcmp(dest_ip, "192.168
.102.131") == 0)
57.     {
58.         // 获取当前系统时间
59.         time_t rawtime;
60.         struct tm *timeinfo;
61.         char time_str[100];
62.         time(&rawtime);
63.         timeinfo = localtime(&rawtime);
64.         // 格式化时间字符串
65.         strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S", timeinfo);
66.
67.         // 打印信息
68.         printf("[%s] Captured packet from %s to %s\n", time_str, src_ip, des
t_ip);
69.         // 修改 TTL
70.         ip_header->ttl -= 1;
71.         ip_header->check = 0;
72.         //ip_header->tot_len = htons(20 + 8 + 30); // 总长度=IP 首部长+IP 数
据长度
73.
74.         ip_header->check = checksum((unsigned short *)ip_header, ip_head
er->ihl * 4);
75.         // 发送数据包到目的主机
76.         struct ifreq ifr, ifr_mac;
77.         struct sockaddr_ll dest;
78.         // 获取网卡接口索引
79.         memset(&ifr, 0, sizeof(ifr));
80.         snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), "ens33");
81.         if (ioctl(sockfd, SIOCGIFINDEX, &ifr) < 0)
82.         {
83.             perror("ioctl");
84.             return 1;
85.         }
86.         // 获取网卡接口 MAC 地址
```

```
87.     memset(&ifr_mac, 0, sizeof(ifr_mac));
88.     snprintf(ifr_mac.ifr_name, sizeof(ifr_mac.ifr_name), "ens33");
89.     if (ioctl(sockfd, SIOCGIFHWADDR, &ifr_mac) < 0)
90.     {
91.         perror("ioctl");
92.         return 1;
93.     }
94.
95.     // 设置目标 MAC 地址 (假设目标地址已知)
96.     unsigned char target_mac[ETH_ALEN] = {0x00, 0x0c, 0x29, 0xcf, 0x92,
        0xca}; // 替换为实际的目标 MAC 地址
97.     memset(&dest, 0, sizeof(dest));
98.     dest.sll_ifindex = ifr.ifr_ifindex;
99.     dest.sll_halen = ETH_ALEN;
100.    memcpy(dest.sll_addr, target_mac, ETH_ALEN);
101.    // 构造新的以太网帧头
102.    memcpy(eth_header->h_dest, target_mac, ETH_ALEN);
    // 目标 MAC 地址
103.    memcpy(eth_header->h_source, ifr_mac.ifr_hwaddr.sa_data, ETH_ALEN)
    ; // 源 MAC 地址
104.    eth_header->h_proto = htons(ETH_P_IP); // 以
    以太网类型为 IP
105.    printf("Interface name: %s, index: %d\n", ifr.ifr_name, ifr.ifr_if
        index);
106.    if (sendto(sockfd, buffer, data_size, 0, (struct sockaddr *)&dest,
        sizeof(dest)) < 0)
107.    {
108.        perror("Sendto error");
109.        return 1;
110.    }
111.    printf("Datagram forwarded.\n");
112.    }
113.    else
114.    {
115.        // printf("Ignored packet from %s to %s\n", src_ip, dest_ip);
116.    }
117.    }
118.    }
119.    close(sockfd);
120.    free(buffer);
121.    return 0;
122. }
```

3) 使用GCC编译并运行程序: gcc -o myroute myroute.c

运行：./myroute

### (3) 接收主机程序

1) 创建一个文件myrec.c

2) 编写以下代码实现接收功能

```
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. #define PORT 12345
8. int main()
9. {
10.     int sockfd;
11.     struct sockaddr_in server_addr, client_addr;
12.     socklen_t addr_len = sizeof(client_addr);
13.     char buffer[1024];
14.     // 创建 UDP 套接字
15.     sockfd = socket(AF_INET, SOCK_DGRAM, 0);
16.     if (sockfd < 0)
17.     {
18.         perror("Socket creation failed");
19.         return 1;
20.     }
21.     // 绑定套接字到端口
22.     memset(&server_addr, 0, sizeof(server_addr));
23.     server_addr.sin_family = AF_INET;
24.     server_addr.sin_addr.s_addr = INADDR_ANY;
25.     server_addr.sin_port = htons(PORT);
26.     if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) <
27.         0)
28.     {
29.         perror("Bind failed");
30.         return 1;
31.     }
32.     // 接收数据包
33.     int recv_len = recvfrom(sockfd, buffer, sizeof(buffer) - 1,
34.                             0, (struct sockaddr *)&client_addr, &addr_len);
35.     if (recv_len < 0)
36.     {
37.         perror("Recvfrom failed");
38.         return 1;
39.     }
```

```

38. }
39. buffer[recv_len] = '\0';
40. printf("Received message: %s\n", buffer);
41. close(sockfd);
42. return 0;
43. }

```

3) 使用GCC编译并运行程序: `gcc -o myrec myrec.c`

运行: `./myrec`

### 3.3 选做内容

扩展实验的网络规模, 由原始方案中 3 台主机增加到不少于 5 台主机, 共同完成 IP 数据报转发及收发过程, 要求采用转发表改进示例程序, 增加程序通用性。



图 2-3 网络拓扑图

创建一台Linux虚拟机之后, 通过虚拟机克隆得到五台Linux虚拟机, 构成如下实验环境。其中源主机的IP地址是192.168.102.130, MAC地址是00:0C:29:E2:D5:BE; 目的主机的IP地址是192.168.102.131, MAC地址是00:0C:29:CF:92:CA; 路由机1的IP地址是192.168.102.132, MAC地址是00:0C:29:9B:08:90; 路由机2的IP地址是192.168.102.133, MAC地址是00:0C:29:A8:B7:60; 路由机3的IP地址是192.168.102.134, MAC地址是00:0C:29:9C:A4:1C。

#### (1) 发送主机程序

1) 创建一个文件`mysend_advanced.c`

2) 编写以下代码实现发送功能

```

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

```



```
16. #define DEST_MAC0    0x00
17. #define DEST_MAC1    0x0C
18. #define DEST_MAC2    0x29
19. #define DEST_MAC3    0x9B
20. #define DEST_MAC4    0x08
21. #define DEST_MAC5    0x90
22.
23. #define ETHER_TYPE    0x0800
24. #define BUFFER_SIZE   1518
25.
26. #define UDP_SRC_PORT   12345
27. #define UDP_FWD_PORT   12345
28. #define UDP_DST_PORT   12345
29. #define UDP_SRC_IP     "192.168.102.130"
30. #define UDP_FWD_IP     "192.168.102.132"
31. #define UDP_DST_IP     "192.168.102.131"
32.
33. unsigned short checksum(void *b, int len){
34.     unsigned short *buf = b;
35.     unsigned int sum = 0;
36.     unsigned short result;
37.     for (sum = 0; len > 1; len -= 2){
38.         sum += *buf++;
39.     }
40.     if (len == 1)
41.         sum += *(unsigned char *)buf;
42.     sum = (sum >> 16) + (sum & 0xFFFF);
43.     sum += (sum >> 16);
44.     result = ~sum;
45.     return result;
46. }
47.
48. /*
49. 以太网帧：以太网头 + IP 头 + UDP 头 + 载荷（信息）。
50. 构建 socket 地址告诉操作系统，此次 Socket 通信将会通过 ens33 网口发送往指定 MAC 地址，
51. 传输的层次为数据链路层使用以太网协议。
52. */
53.
54. int main(){
55.     char buffer[BUFFER_SIZE];
56.
57.     // 创建原始套接字
58.     int sockfd;
```

```
59.  if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) == -1) {
60.      perror("Socket");
61.      return 1;
62.  }
63.
64.  // 获取接口索引
65.  struct ifreq if_idx;
66.  memset(&if_idx, 0, sizeof(struct ifreq));
67.  strncpy(if_idx.ifr_name, "ens33", IFNAMSIZ - 1);
68.  if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0)
69.  {
70.      perror("SIOCGIFINDEX");
71.      return 1;
72.  }
73.  // 获取接口 MAC 地址
74.  struct ifreq if_mac;
75.  memset(&if_mac, 0, sizeof(struct ifreq));
76.  strncpy(if_mac.ifr_name, "ens33", IFNAMSIZ - 1);
77.  if (ioctl(sockfd, SIOCGIFHWADDR, &if_mac) < 0)
78.  {
79.      perror("SIOCGIFHWADDR");
80.      return 1;
81.  }
82.
83.  char message[256];
84.  memset(message, 0, sizeof(message));
85.  while(1){
86.      printf("[SENDER:%10u] Please input message: ", time(0));
87.      fgets(message, sizeof(message), stdin);
88.
89.      message[strlen(message) - 1] = 0;
90.
91.      // 构造以太网头
92.      struct ether_header *eh = (struct ether_header *)buffer;
93.      eh->ether_shost[0] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[0];
94.      eh->ether_shost[1] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[1];
95.      eh->ether_shost[2] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[2];
96.      eh->ether_shost[3] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[3];
97.      eh->ether_shost[4] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[4];
98.      eh->ether_shost[5] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[5];
99.
100.      printf("[SENDER:%10u] SRC MAC Address = ", time(0));
101.      for (int i = 0; i < 6; i++){
102.          printf("%02X%c", eh->ether_shost[i], ":\n"[i + 1 == 6]);
```

```
103.     }
104.
105.     eh->ether_dhost[0] = DEST_MAC0;
106.     eh->ether_dhost[1] = DEST_MAC1;
107.     eh->ether_dhost[2] = DEST_MAC2;
108.     eh->ether_dhost[3] = DEST_MAC3;
109.     eh->ether_dhost[4] = DEST_MAC4;
110.     eh->ether_dhost[5] = DEST_MAC5;
111.     eh->ether_type = htons(ETHER_TYPE);
112.
113.     printf("[SENDER:%10u] DST MAC Address = ", time(0));
114.     for (int i = 0; i < 6; i++){
115.         printf("%02X%c", eh->ether_dhost[i], ":\n"[i + 1 == 6]);
116.     }
117.
118.     // 构造 IP 头
119.     struct iphdr *iph = (struct iphdr *)(buffer + sizeof(struct ether_header));
120.     iph->ihl = 5;
121.     iph->version = 4;
122.     iph->tos = 0;
123.     iph->tot_len = htons(sizeof(struct iphdr) + sizeof(struct udphdr) +
        strlen(message));
124.     iph->id = htonl(UDP_DST_PORT);
125.     iph->frag_off = 0;
126.     iph->ttl = 255;
127.     iph->protocol = IPPROTO_UDP;
128.     iph->check = 0;
129.     iph->saddr = inet_addr(UDP_SRC_IP);
130.     iph->daddr = inet_addr(UDP_DST_IP);
131.     iph->check = checksum((unsigned short *)iph, sizeof(struct iphdr));
132.
133.     printf("[SENDER:%10u] SRC IP Address = %s\n", time(0), UDP_SRC_IP);
134.     printf("[SENDER:%10u] DST IP Address = %s\n", time(0), UDP_DST_IP);
135.     printf("[SENDER:%10u] TTL = %d\n", time(0), iph->ttl);
136.
137.     // 构造 UDP 头
138.     struct udphdr *udph = (struct udphdr *)(buffer + sizeof(struct ether_header)
        + sizeof(struct iphdr));
139.     udph->source = htons(UDP_SRC_PORT);
```

```
141.     udph->dest = htons(UDP_DST_PORT);
142.     udph->len = htons(sizeof(struct udphdr) + strlen(message));
143.     udph->check = 0;
144.
145.     // 填充数据
146.     char *data = (char *)(buffer + sizeof(struct ether_header)
147.
148.         + sizeof(struct iphdr) + sizeof(struct udphdr));
149.     strcpy(data, message);
150.
151.     // 设置 socket 地址结构
152.     struct sockaddr_ll socket_address;
153.     socket_address.sll_ifindex = if_idx.ifr_ifindex;
154.     socket_address.sll_halen = ETH_ALEN;
155.     socket_address.sll_addr[0] = DEST_MAC0;
156.     socket_address.sll_addr[1] = DEST_MAC1;
157.     socket_address.sll_addr[2] = DEST_MAC2;
158.     socket_address.sll_addr[3] = DEST_MAC3;
159.     socket_address.sll_addr[4] = DEST_MAC4;
160.     socket_address.sll_addr[5] = DEST_MAC5;
161.
162.     // 发送数据包
163.     int len = sizeof(struct ether_header) + sizeof(struct iphdr)
164.         + sizeof(struct udphdr) + strlen(message);
165.     printf("[SENDER:%10u] Sending message (load = [%s](%d)), total length = %dByte\n",
166.
167.         time(0), message, strlen(message), len);
168.
169.     if (sendto(sockfd, buffer, len, 0, (struct sockaddr *)&socket_address,
170.
171.         sizeof(struct sockaddr_ll)) < 0){
172.         perror("Sendto");
173.         return 1;
174.     }
175. }
```

3) 使用GCC编译并运行程序: gcc -o mysend\_advanced mysend\_advanced.c

运行: ./mysend\_advanced

(2) 路由转发1程序

1) 创建一个文件myroute\_advanced.c

2) 编写以下代码实现转发功能

```
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
16.
17. struct Hop {
18.     const char *src; // 原始地址
19.     const char *dst; // 目的地址
20.     const char *hop; // 下一跳地址
21.
22.     // 下一跳 MAC 地址
23.     unsigned char hop_mac0;
24.     unsigned char hop_mac1;
25.     unsigned char hop_mac2;
26.     unsigned char hop_mac3;
27.     unsigned char hop_mac4;
28.     unsigned char hop_mac5;
29. }table[256];
30.
31. unsigned short checksum(void *b, int len)
32. {
33.     unsigned short *buf = b;
34.     unsigned int sum = 0;
35.     unsigned short result;
36.     for (sum = 0; len > 1; len -= 2)
37.         sum += *buf++;
38.     if (len == 1)
39.         sum += *(unsigned char *)buf;
40.     sum = (sum >> 16) + (sum & 0xFFFF);
41.     sum += (sum >> 16);
42.     result = ~sum;
43.     return result;
```



```
83.  {
84.    perror("Socket");
85.    return 1;
86.  }
87.  while (1) {
88.    int saddr_len = sizeof(saddr);
89.    int data_size = recvfrom(sockfd, buffer, BUFFER_SIZE,
90.                             0, &saddr, (socklen_t *)&saddr_len);
91.    if (data_size < 0) {
92.      perror("Recvfrom");
93.      return 1;
94.    }
95.
96.    struct ethhdr *eth_header = (struct ethhdr *)buffer;
97.    struct iphdr *ip_header = (struct iphdr *)(buffer + sizeof(struct ethh
dr));
98.    char src_ip[INET_ADDRSTRLEN];
99.    char dst_ip[INET_ADDRSTRLEN];
100.    inet_ntop(AF_INET, &(ip_header->saddr), src_ip, INET_ADDRSTRLEN);
101.    inet_ntop(AF_INET, &(ip_header->daddr), dst_ip, INET_ADDRSTRLEN);
102.
103.    int p = 0;
104.    for(int i = 1; i <= table_size; ++i){
105.      int ok1 = !strcmp(table[i].src, src_ip);
106.      int ok2 = !strcmp(table[i].dst, dst_ip);
107.
108.      if(ok1 && ok2){
109.        p = i;
110.        break;
111.      }
112.    }
113.
114.    if(p == 0){ // 未在路由表里找到信息，忽略
115.      // printf("[ROUTER:%10u] Ignored packet from %s to %s\n", time(0),
src_ip, dst_ip);
116.      continue;
117.    }
118.
119.    // 获取当前系统时间
120.    time_t rawtime;
121.    struct tm *timeinfo;
122.    char time_str[100];
123.    time(&rawtime);
124.    timeinfo = localtime(&rawtime);
```

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

127.
128. // 打印信息
129. printf("[ROUTER:%10u] At %s captured packet from %s to %s\n",
130.         time(0), time_str, src_ip, dst_ip);
131. printf("[ROUTER:%10u] SRC_IP = %15s\n", time(0), src_ip);
132. printf("[ROUTER:%10u] DST_IP = %15s\n", time(0), dst_ip);
133. printf("[ROUTER:%10u] SRC_MAC = %02X:%02X:%02X:%02X:%02X:%02X\n",
134.         time(0),
135.         eth_header->h_source[0],
136.         eth_header->h_source[1],
137.         eth_header->h_source[2],
138.         eth_header->h_source[3],
139.         eth_header->h_source[4],
140.         eth_header->h_source[5]
141. );
142. printf("[ROUTER:%10u] DST_MAC = %02X:%02X:%02X:%02X:%02X:%02X\n",
143.         time(0),
144.         eth_header->h_dest[0],
145.         eth_header->h_dest[1],
146.         eth_header->h_dest[2],
147.         eth_header->h_dest[3],
148.         eth_header->h_dest[4],
149.         eth_header->h_dest[5]
150. );
151. printf("[ROUTER:%10u] TTL = %d\n", time(0), ip_header->ttl);
152.
153. // 修改 TTL
154. ip_header->ttl -= 1;
155. ip_header->check = 0;
156. //ip_header->tot_len = htons(20 + 8 + 30); // 总长度=IP 首部长度+IP 数
    据长度
157.
158. ip_header->check = checksum((unsigned short *)ip_header, ip_head
    er->ihl * 4);
159.
160. // 发送数据包到目的主机
161. struct ifreq ifr, ifr_mac;
162. struct sockaddr_ll hop;
163.
164. // 获取网卡接口索引
165. memset(&ifr, 0, sizeof(ifr));
```



```
166.     snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), "ens33");
167.     if (ioctl(sockfd, SIOCGIFINDEX, &ifr) < 0)
168.     {
169.         perror("Ioctl");
170.         return 1;
171.     }
172.
173.     // 获取网卡接口 MAC 地址
174.     memset(&ifr_mac, 0, sizeof(ifr_mac));
175.     snprintf(ifr_mac.ifr_name, sizeof(ifr_mac.ifr_name), "ens33");
176.     if (ioctl(sockfd, SIOCGIFHWADDR, &ifr_mac) < 0)
177.     {
178.         perror("Ioctl");
179.         return 1;
180.     }
181.
182.     // 设置目标 MAC 地址
183.     unsigned char target_mac[ETH_ALEN] = {
184.         table[p].hop_mac0, table[p].hop_mac1, table[p].hop_mac2,
185.         table[p].hop_mac3, table[p].hop_mac4, table[p].hop_mac5
186.     };
187.
188.     memset(&hop, 0, sizeof(hop));
189.     hop.sll_ifindex = ifr.ifr_ifindex;
190.     hop.sll_halen = ETH_ALEN;
191.     memcpy(hop.sll_addr, target_mac, ETH_ALEN);
192.     // 构造新的以太网帧头
193.     memcpy(eth_header->h_dest, target_mac, ETH_ALEN);
194.     // 目标 MAC 地址
195.     memcpy(eth_header->h_source, ifr_mac.ifr_hwaddr.sa_data, ETH_ALEN);
196.     // 源 MAC 地址
197.     eth_header->h_proto = htons(ETH_P_IP);
198.     // 以太网类型为 IP
199.
200.     printf("[ROUTER:%10u] Interface name: %s, index: %d\n",
201.           time(0), ifr.ifr_name, ifr.ifr_ifindex);
202.     if (sendto(sockfd, buffer, data_size, 0, (struct sockaddr *)&hop, si
203.           zeof(hop)) < 0)
204.     {
205.         perror("Sendto");
206.         return 1;
207.     }
208.     printf("[ROUTER:%10u] Datagram forwarded.\n", time(0));
209. }
```

```
206. close(sockfd);
207. return 0;
208. }
```

3) 使用GCC编译并运行程序: `gcc -o myroute_advanced myroute_advanced.c`

运行: `./myroute_advanced`

(3) 路由转发2程序

1) 创建一个文件myroute\_advanced.c

2) 编写以下代码实现转发功能

```
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
16.
17. struct Hop {
18.     const char *src; // 原始地址
19.     const char *dst; // 目的地址
20.     const char *hop; // 下一跳地址
21.
22.     // 下一跳 MAC 地址
23.     unsigned char hop_mac0;
24.     unsigned char hop_mac1;
25.     unsigned char hop_mac2;
26.     unsigned char hop_mac3;
27.     unsigned char hop_mac4;
28.     unsigned char hop_mac5;
29. }table[256];
30.
31. unsigned short checksum(void *b, int len)
32. {
33.     unsigned short *buf = b;
34.     unsigned int sum = 0;
```

```
35. unsigned short result;
36. for (sum = 0; len > 1; len -= 2)
37.     sum += *buf++;
38. if (len == 1)
39.     sum += *(unsigned char *)buf;
40. sum = (sum >> 16) + (sum & 0xFFFF);
41. sum += (sum >> 16);
42. result = ~sum;
43. return result;
44. }
45.
46. int main()
47. {
48.     // 初始化路由表
49.
50.     int table_size = 1;
51.
52.     table[1].src = "192.168.102.130"; // 从 1 号主机（发送方）传来
53.     table[1].dst = "192.168.102.131"; // 到 2 号主机（接收方）处去
54.     table[1].hop = "192.168.102.134"; // 下一跳应该发送去的 IP 地址
55.
56.     // 下一跳应该发送去的 MAC 地址
57.     table[1].hop_mac0 = 0x00;
58.     table[1].hop_mac1 = 0x0C;
59.     table[1].hop_mac2 = 0x29;
60.     table[1].hop_mac3 = 0x9C;
61.     table[1].hop_mac4 = 0xA4;
62.     table[1].hop_mac5 = 0x1C;
63.
64.     printf("=====\n");
65.     printf("|      src_ip      |      dst_ip      |      hop_ip      | h\n");
66.     printf("| op_MAC      |\n");
67.     printf("+-----+-----+-----+-----+\n");
68.     for(int i = 1; i <= table_size; ++ i){
69.         printf("| %15s | %15s | %15s | %02X:%02X:%02X:%02X:%02X:%02X |\n",
70.             table[i].src, table[i].dst, table[i].hop,
71.             table[i].hop_mac0, table[i].hop_mac1, table[i].hop_mac2,
72.             table[i].hop_mac3, table[i].hop_mac4, table[i].hop_mac5
73.         );
74.     }
```

```
74.  printf("=====  
=====\\n");  
75.  
76.  struct sockaddr saddr;  
77.  char buffer[BUFFER_SIZE];  
78.  
79.  // 创建 UDP 套接字  
80.  int sockfd;  
81.  sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP));  
82.  if (sockfd < 0)  
83.  {  
84.      perror("Socket");  
85.      return 1;  
86.  }  
87.  while (1) {  
88.      int saddr_len = sizeof(saddr);  
89.      int data_size = recvfrom(sockfd, buffer, BUFFER_SIZE,  
90.                              0, &saddr, (socklen_t *)&saddr_len);  
91.      if (data_size < 0) {  
92.          perror("Recvfrom");  
93.          return 1;  
94.      }  
95.  
96.      struct ethhdr *eth_header = (struct ethhdr *)buffer;  
97.      struct iphdr *ip_header = (struct iphdr *)(buffer + sizeof(struct ethh  
dr));  
98.      char src_ip[INET_ADDRSTRLEN];  
99.      char dst_ip[INET_ADDRSTRLEN];  
100.      inet_ntop(AF_INET, &(ip_header->saddr), src_ip, INET_ADDRSTRLEN);  
101.      inet_ntop(AF_INET, &(ip_header->daddr), dst_ip, INET_ADDRSTRLEN);  
102.  
103.      int p = 0;  
104.      for(int i = 1; i <= table_size; ++ i){  
105.          int ok1 = !strcmp(table[i].src, src_ip);  
106.          int ok2 = !strcmp(table[i].dst, dst_ip);  
107.  
108.          if(ok1 && ok2){  
109.              p = i;  
110.              break;  
111.          }  
112.      }  
113.  
114.      if(p == 0){ // 未在路由表里找到信息, 忽略
```

```
115.     // printf("[ROUTER:%10u] Ignored packet from %s to %s\n", time(0),
    src_ip, dst_ip);
116.     continue;
117. }
118.
119. // 获取当前系统时间
120. time_t rawtime;
121. struct tm *timeinfo;
122. char time_str[100];
123. time(&rawtime);
124. timeinfo = localtime(&rawtime);
125. // 格式化时间字符串
126. strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S", timeinfo);
127.
128. // 打印信息
129. printf("[ROUTER:%10u] At %s captured packet from %s to %s\n",
130.         time(0), time_str, src_ip, dst_ip);
131. printf("[ROUTER:%10u] SRC_IP = %15s\n", time(0), src_ip);
132. printf("[ROUTER:%10u] DST_IP = %15s\n", time(0), dst_ip);
133. printf("[ROUTER:%10u] SRC_MAC = %02X:%02X:%02X:%02X:%02X:%02X\n",
134.         time(0),
135.         eth_header->h_source[0],
136.         eth_header->h_source[1],
137.         eth_header->h_source[2],
138.         eth_header->h_source[3],
139.         eth_header->h_source[4],
140.         eth_header->h_source[5]
141. );
142. printf("[ROUTER:%10u] DST_MAC = %02X:%02X:%02X:%02X:%02X:%02X\n",
143.         time(0),
144.         eth_header->h_dest[0],
145.         eth_header->h_dest[1],
146.         eth_header->h_dest[2],
147.         eth_header->h_dest[3],
148.         eth_header->h_dest[4],
149.         eth_header->h_dest[5]
150. );
151. printf("[ROUTER:%10u] TTL = %d\n", time(0), ip_header->ttl);
152.
153. // 修改 TTL
154. ip_header->ttl -= 1;
155. ip_header->check = 0;
```

```
156.    //ip_header->tot_len = htons(20 + 8 + 30); // 总长度=IP 首部长度+IP 数
      据长度
157.
158.    ip_header->check = checksum((unsigned short *)ip_header, ip_head
      er->ihl * 4);
159.
160.    // 发送数据包到目的主机
161.    struct ifreq ifr, ifr_mac;
162.    struct sockaddr_ll hop;
163.
164.    // 获取网卡接口索引
165.    memset(&ifr, 0, sizeof(ifr));
166.    snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), "ens33");
167.    if (ioctl(sockfd, SIOCGIFINDEX, &ifr) < 0)
168.    {
169.        perror("Ioctl");
170.        return 1;
171.    }
172.
173.    // 获取网卡接口 MAC 地址
174.    memset(&ifr_mac, 0, sizeof(ifr_mac));
175.    snprintf(ifr_mac.ifr_name, sizeof(ifr_mac.ifr_name), "ens33");
176.    if (ioctl(sockfd, SIOCGIFHWADDR, &ifr_mac) < 0)
177.    {
178.        perror("Ioctl");
179.        return 1;
180.    }
181.
182.    // 设置目标 MAC 地址
183.    unsigned char target_mac[ETH_ALEN] = {
184.        table[p].hop_mac0, table[p].hop_mac1, table[p].hop_mac2,
185.        table[p].hop_mac3, table[p].hop_mac4, table[p].hop_mac5
186.    };
187.
188.    memset(&hop, 0, sizeof(hop));
189.    hop.sll_ifindex = ifr.ifr_ifindex;
190.    hop.sll_halen = ETH_ALEN;
191.    memcpy(hop.sll_addr, target_mac, ETH_ALEN);
192.    // 构造新的以太网帧头
193.    memcpy(eth_header->h_dest, target_mac, ETH_ALEN);
      // 目标 MAC 地址
194.    memcpy(eth_header->h_source, ifr_mac.ifr_hwaddr.sa_data, ETH_ALEN);
      // 源 MAC 地址
```

```
195.     eth_header->h_proto = htons(ETH_P_IP);
        // 以太网类型为 IP
196.
197.     printf("[ROUTER:%10u] Interface name: %s, index: %d\n",
198.            time(0), ifr.ifr_name, ifr.ifr_ifindex);
199.     if (sendto(sockfd, buffer, data_size, 0, (struct sockaddr *)&hop, sizeof(hop)) < 0)
200.     {
201.         perror("Sendto");
202.         return 1;
203.     }
204.     printf("[ROUTER:%10u] Datagram forwarded.\n", time(0));
205. }
206. close(sockfd);
207. return 0;
208. }
```

3) 使用GCC编译并运行程序: gcc -o myroute\_advanced myroute\_advanced.c

运行: ./myroute\_advanced

(4) 路由转发3程序

1) 创建一个文件myroute\_advanced.c

2) 编写以下代码实现转发功能

```
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
16.
17. struct Hop {
18.     const char *src; // 原始地址
19.     const char *dst; // 目的地址
20.     const char *hop; // 下一跳地址
21. }
```

```
22. // 下一跳 MAC 地址
23. unsigned char hop_mac0;
24. unsigned char hop_mac1;
25. unsigned char hop_mac2;
26. unsigned char hop_mac3;
27. unsigned char hop_mac4;
28. unsigned char hop_mac5;
29. }table[256];
30.
31. unsigned short checksum(void *b, int len)
32. {
33.     unsigned short *buf = b;
34.     unsigned int sum = 0;
35.     unsigned short result;
36.     for (sum = 0; len > 1; len -= 2)
37.         sum += *buf++;
38.     if (len == 1)
39.         sum += *(unsigned char *)buf;
40.     sum = (sum >> 16) + (sum & 0xFFFF);
41.     sum += (sum >> 16);
42.     result = ~sum;
43.     return result;
44. }
45.
46. int main()
47. {
48.     // 初始化路由表
49.
50.     int table_size = 1;
51.
52.     table[1].src = "192.168.102.130"; // 从 1 号主机（发送方）传来
53.     table[1].dst = "192.168.102.131"; // 到 2 号主机（接收方）处去
54.     table[1].hop = "192.168.102.131"; // 下一跳应该发送去的 IP 地址
55.
56.     // 下一跳应该发送去的 MAC 地址
57.     table[1].hop_mac0 = 0x00;
58.     table[1].hop_mac1 = 0x0C;
59.     table[1].hop_mac2 = 0x29;
60.     table[1].hop_mac3 = 0xCF;
61.     table[1].hop_mac4 = 0x92;
62.     table[1].hop_mac5 = 0xCA;
63.
64.     printf("=====\n");
```





```
104.     for(int i = 1; i <= table_size; ++ i){
105.         int ok1 = !strcmp(table[i].src, src_ip);
106.         int ok2 = !strcmp(table[i].dst, dst_ip);
107.
108.         if(ok1 && ok2){
109.             p = i;
110.             break;
111.         }
112.     }
113.
114.     if(p == 0){        // 未在路由表里找到信息，忽略
115.         // printf("[ROUTER:%10u] Ignored packet from %s to %s\n", time(0),
116.             src_ip, dst_ip);
117.         continue;
118.     }
119.     // 获取当前系统时间
120.     time_t rawtime;
121.     struct tm *timeinfo;
122.     char time_str[100];
123.     time(&rawtime);
124.     timeinfo = localtime(&rawtime);
125.     // 格式化时间字符串
126.     strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S", timeinfo);
127.
128.     // 打印信息
129.     printf("[ROUTER:%10u] At %s captured packet from %s to %s\n",
130.         time(0), time_str, src_ip, dst_ip);
131.     printf("[ROUTER:%10u] SRC_IP = %15s\n", time(0), src_ip);
132.     printf("[ROUTER:%10u] DST_IP = %15s\n", time(0), dst_ip);
133.     printf("[ROUTER:%10u] SRC_MAC = %02X:%02X:%02X:%02X:%02X:%02X\n",
134.         time(0),
135.         eth_header->h_source[0],
136.         eth_header->h_source[1],
137.         eth_header->h_source[2],
138.         eth_header->h_source[3],
139.         eth_header->h_source[4],
140.         eth_header->h_source[5]
141.     );
142.     printf("[ROUTER:%10u] DST_MAC = %02X:%02X:%02X:%02X:%02X:%02X\n",
143.         time(0),
144.         eth_header->h_dest[0],
145.         eth_header->h_dest[1],
```

```
146.     eth_header->h_dest[2],
147.     eth_header->h_dest[3],
148.     eth_header->h_dest[4],
149.     eth_header->h_dest[5]
150. );
151. printf("[ROUTER:%10u] TTL = %d\n", time(0), ip_header->ttl);
152.
153. // 修改 TTL
154. ip_header->ttl -= 1;
155. ip_header->check = 0;
156. //ip_header->tot_len = htons(20 + 8 + 30); // 总长度=IP 首部长度+IP 数
    据长度
157.
158.     ip_header->check = checksum((unsigned short *)ip_header, ip_head
    er->ihl * 4);
159.
160. // 发送数据包到目的主机
161. struct ifreq ifr, ifr_mac;
162. struct sockaddr_ll hop;
163.
164. // 获取网卡接口索引
165. memset(&ifr, 0, sizeof(ifr));
166. snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), "ens33");
167. if (ioctl(sockfd, SIOCGIFINDEX, &ifr) < 0)
168. {
169.     perror("Ioctl");
170.     return 1;
171. }
172.
173. // 获取网卡接口 MAC 地址
174. memset(&ifr_mac, 0, sizeof(ifr_mac));
175. snprintf(ifr_mac.ifr_name, sizeof(ifr_mac.ifr_name), "ens33");
176. if (ioctl(sockfd, SIOCGIFHWADDR, &ifr_mac) < 0)
177. {
178.     perror("Ioctl");
179.     return 1;
180. }
181.
182. // 设置目标 MAC 地址
183. unsigned char target_mac[ETH_ALEN] = {
184.     table[p].hop_mac0, table[p].hop_mac1, table[p].hop_mac2,
185.     table[p].hop_mac3, table[p].hop_mac4, table[p].hop_mac5
186. };
187.
```

```
188.     memset(&hop, 0, sizeof(hop));
189.     hop.sll_ifindex = ifr.ifr_ifindex;
190.     hop.sll_halen = ETH_ALEN;
191.     memcpy(hop.sll_addr, target_mac, ETH_ALEN);
192.     // 构造新的以太网帧头
193.     memcpy(eth_header->h_dest, target_mac, ETH_ALEN);
        // 目标 MAC 地址
194.     memcpy(eth_header->h_source, ifr_mac.ifr_hwaddr.sa_data, ETH_ALEN);
        // 源 MAC 地址
195.     eth_header->h_proto = htons(ETH_P_IP);
        // 以太网类型为 IP
196.
197.     printf("[ROUTER:%10u] Interface name: %s, index: %d\n",
198.            time(0), ifr.ifr_name, ifr.ifr_ifindex);
199.     if (sendto(sockfd, buffer, data_size, 0, (struct sockaddr *)&hop, si
        zeof(hop)) < 0)
200.     {
201.         perror("Sendto");
202.         return 1;
203.     }
204.     printf("[ROUTER:%10u] Datagram forwarded.\n", time(0));
205. }
206. close(sockfd);
207. return 0;
208. }
```

3) 使用GCC编译并运行程序: gcc -o myroute\_advanced myroute\_advanced.c

运行: ./myroute\_advanced

(5) 接收主机程序

1) 创建一个文件mysend\_advanced.c

2) 编写以下代码实现接收功能

```
1. #include <stdio.h>
2. #include <time.h>
3. #include <stdlib.h>
4. #include <string.h>
5. #include <arpa/inet.h>
6. #include <sys/socket.h>
7.
8. #define UDP_SRC_PORT 12345
9. #define UDP_FWD_PORT 12345
10. #define UDP_DST_PORT 12345
11. #define UDP_SRC_IP "192.168.102.130"
12. #define UDP_FWD_IP "192.168.102.132"
```

```
13. #define UDP_DST_IP    "192.168.102.131"
14.
15. int main()
16. {
17.     // 创建 UDP 套接字
18.     int sockfd;
19.     if ((sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0) {
20.         perror("Socket");
21.         return 1;
22.     }
23.
24.     // 目标地址
25.     struct sockaddr_in dst_addr;
26.     dst_addr.sin_family = AF_INET;
27.     dst_addr.sin_port = htons(UDP_DST_PORT);
28.     dst_addr.sin_addr.s_addr = inet_addr(UDP_DST_IP);
29.
30.     // 绑定套接字到本地地址
31.     if (bind(sockfd, (struct sockaddr *)&dst_addr, sizeof(dst_addr)) < 0) {
32.         perror("Bind");
33.         return 1;
34.     } else {
35.         printf("[RECVER:%10u] Bind socket successfully.\n", time(0));
36.     }
37.
38.     char message[1024];
39.     while(1){
40.         // 接收数据报
41.         struct sockaddr_in src_addr;
42.         socklen_t addr_len = sizeof(src_addr);
43.
44.         int recv_len = recvfrom(sockfd, message, sizeof(message),
45.                                0, (struct sockaddr *)&src_addr, &addr_len);
46.         if (recv_len < 0) {
47.             perror("Recvfrom");
48.             return 1;
49.         }
50.         message[recv_len] = '\0';
51.         printf("[RECVER:%10u] Datagram received: [%s](%d).\n",
52.               time(0), message, strlen(message));
53.     }
54.     return 0;
```

55. }

3) 使用GCC编译并运行程序: gcc -o mysend\_advanced mysend\_advanced.c  
运行: ./mysend\_advanced

四、基于双网口主机的路由转发

4.1 环境配置

网络拓扑图如下图所示。



图 4-1 网络拓扑图

(1) 静态IP设置

通过vim /etc/network/interfaces添加配置将虚拟机设置静态IP及双网口。

创建一台Linux虚拟机之后，通过虚拟机克隆得到三台Linux虚拟机，构成如下实验环境。其中源主机的IP地址是192.168.109.128，MAC地址是00:0C:29:A3:44:AE；目的主机的IP地址是192.168.152.128，MAC地址是00:0C:29:8E:33:8E；

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug ens33
# iface ens33 inet dhcp
iface ens33 inet static
address 192.168.109.128/24
```

图 4-2 源主机静态 IP 设置

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug ens33
# iface ens33 inet dhcp
iface ens33 inet static
address 192.168.152.128/24
```

图 4-2 目的主机静态 IP 设置

路由机有两个网络适配器，ens33处IP地址是192.168.109.2，MAC地址是00:0C:29:08:3C:53；ens37处IP地址是192.168.152.2，MAC地址是00:0C:29:08:3C:5D。



图 4-3 双网口设置

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug ens33
# iface ens33 inet dhcp
iface ens33 inet static
address 192.168.109.2/24

auto ens37
iface ens37 inet static
address 192.168.152.2/24
```

图 4-4 路由主机静态 IP 设置

## (2) 设置网关

- 1) 在源主机中设置 `ip route add default via 192.168.109.2`
- 2) 在目的主机中设置 `ip route add default via 192.168.152.2`

## 4.2 必做内容

构造了静态路由表，并实现了不同子网间的 IP 数据报查表转发过程。

### (1) 发送主机程序

- 1) 创建一个文件 `send.c`
- 2) 编写以下代码实现发送功能

```
1. #include <arpa/inet.h>
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <string.h>
5. #include <sys/socket.h>
6. #include <unistd.h>
7. #define DEST_IP "192.168.152.128"
8. #define DEST_PORT 12345
9. #define MESSAGE "Hello, this is a test message."
10. int main() {
11.     int sockfd;
12.     struct sockaddr_in dest_addr;
13.     // 创建 UDP 套接字
```

```
14. sockfd = socket(AF_INET, SOCK_DGRAM, 0);
15. if (sockfd < 0) {
16.     perror("Socket creation failed");
17.     return 1;
18. }
19. // 设置目的地址
20. memset(&dest_addr, 0, sizeof(dest_addr));
21. dest_addr.sin_family = AF_INET;
22. dest_addr.sin_port = htons(DEST_PORT);
23. inet_pton(AF_INET, DEST_IP, &dest_addr.sin_addr);
24. // 发送数据包
25. if (sendto(sockfd, MESSAGE, strlen(MESSAGE), 0,
26.             (struct sockaddr *)&dest_addr, sizeof(dest_addr)) < 0) {
27.     perror("Sendto failed");
28.     return 1;
29. }
30. printf("Message sent to %s:%d\n", DEST_IP, DEST_PORT);
31. close(sockfd);
32. return 0;
33. }
```

3) 使用GCC编译并运行程序: gcc -o send send.c

运行: ./send

(2) 路由主机程序

1) 创建一个文件route.c

2) 编写以下代码实现路由功能

```
1. #include <arpa/inet.h>
2. #include <linux/if_packet.h>
3. #include <net/if.h>
4. #include <netinet/if_ether.h>
5. #include <netinet/ip.h>
6. #include <stdio.h>
7. #include <stdlib.h>
8. #include <string.h>
9. #include <sys/ioctl.h>
10. #include <sys/socket.h>
11. #include <time.h>
12. #include <unistd.h>
13.
14. #define BUFFER_SIZE 65536
15. struct route_entry {
16.     uint32_t dest;
17.     uint32_t gateway;
```



```
18.     uint32_t netmask;
19.     char interface[IFNAMSIZ];
20. };
21. struct route_entry route_table[1];
22.
23. int route_table_size = sizeof(route_table) / sizeof(route_table[0]);
24.
25. void convert_to_ip_string(uint32_t ip_addr, char *ip_str) {
26.     struct in_addr addr;
27.     addr.s_addr =
28.         ip_addr; // htonl(ip_addr); // 转换为网络字节
    序 inet_ntop(AF_INET,
29.                 // &addr, ip_str, INET_ADDRSTRLEN);
30.     inet_ntop(AF_INET, &addr, ip_str, INET_ADDRSTRLEN);
31. }
32.
33. unsigned short checksum(void *b, int len) {
34.     unsigned short *buf = b;
35.     unsigned int sum = 0;
36.     unsigned short result;
37.     for (sum = 0; len > 1; len -= 2) sum += *buf++;
38.     if (len == 1) sum += *(unsigned char *)buf;
39.     sum = (sum >> 16) + (sum & 0xFFFF);
40.     sum += (sum >> 16);
41.     result = ~sum;
42.     return result;
43. }
44.
45. struct route_entry *lookup_route(uint32_t dest_ip) {
46.     char ip_str[32];
47.
48.     for (int i = 0; i < route_table_size; i++) {
49.         // convert_to_ip_string(dest_ip, ip_str);
50.         // printf("IP Address: %s\n", ip_str);
51.
52.         // convert_to_ip_string(route_table[i].dest, ip_str);
53.         // printf("IP Address: %s\n", ip_str);
54.
55.         if ((dest_ip & route_table[i].netmask) ==
56.             (route_table[i].dest & route_table[i].netmask)) {
57.             convert_to_ip_string(dest_ip, ip_str);
58.             printf("-----IP Address: %s\n", ip_str);
59.
60.             convert_to_ip_string(route_table[i].dest, ip_str);
```

```
61.         printf("-----IP Address: %s\n", ip_str);
62.         return &route_table[i];
63.     }
64. }
65. return NULL;
66. }
67.
68. void initialize_route_table() {
69.     route_table[0].dest = inet_addr("192.168.152.128");
70.     route_table[0].gateway = inet_addr("192.168.109.2");
71.     route_table[0].netmask = inet_addr("255.255.255.0");
72.     strcpy(route_table[0].interface, "ens37");
73. }
74.
75. int main() {
76.     int sockfd;
77.     struct sockaddr saddr;
78.     unsigned char *buffer = (unsigned char *)malloc(BUFFER_SIZE);
79.
80.     initialize_route_table();
81.
82.     sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP));
83.     if (sockfd < 0) {
84.         perror("Socket creation failed");
85.         return 1;
86.     }
87.     while (1) {
88.         int saddr_len = sizeof(saddr);
89.         int data_size = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, &saddr,
90.                                 (socklen_t *)&saddr_len);
91.         if (data_size < 0) {
92.             perror("Recvfrom error");
93.             return 1;
94.         }
95.         if (data_size == 0) continue;
96.         struct ethhdr *eth_header = (struct ethhdr *)buffer;
97.
98.         struct iphdr *ip_header =
99.             (struct iphdr *)(buffer + sizeof(struct ethhdr));
100.        struct route_entry *route = lookup_route(ip_header->daddr);
101.
102.        if (route == NULL) {
103.            // fprintf(stderr, "No route to host\n");
104.            continue;
```

```
105.     }
106.     char ip_s[32], ip_d[32];
107.     convert_to_ip_string(ip_header->saddr, ip_s);
108.     convert_to_ip_string(ip_header->daddr, ip_d);
109.
110.     printf("Captured packet from %s to %s\n", ip_s, ip_d);
111.
112.     // 修改 TTL
113.     ip_header->ttl -= 1;
114.     ip_header->check = 0;
115.     ip_header->check =
116.         checksum((unsigned short *)ip_header, ip_header->ihl * 4);
117.     // 发送数据包到目的主机
118.     struct ifreq ifr, ifr_mac;
119.     struct sockaddr_ll dest;
120.     // 获取网卡接口索引
121.     memset(&ifr, 0, sizeof(ifr));
122.     snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), route->interface);
123.
124.     if (ioctl(sockfd, SIOCGIFINDEX, &ifr) < 0) {
125.         perror("ioctl");
126.         return 1;
127.     }
128.     // 获取网卡接口 MAC 地址
129.     memset(&ifr_mac, 0, sizeof(ifr_mac));
130.     snprintf(ifr_mac.ifr_name, sizeof(ifr_mac.ifr_name), route->inte
131.         rface);
132.     if (ioctl(sockfd, SIOCGIFHWADDR, &ifr_mac) < 0) {
133.         perror("ioctl");
134.         return 1;
135.     }
136.     // 设置目标 MAC
137.     // 地址（假设目标地址已知,此处做了简化处理，实际上，如果查找路由表后，
138.     // 存在“下
139.     // 一跳”，应该利用 ARP 协议获得 route->gateway 的 MAC
140.     // 地址，如果是“直接交付”的话，也应使用 ARP 协议获得 目的主机的 MAC
141.     // 地址。）
142.     unsigned char target_mac[ETH_ALEN] = {0x00, 0x0c, 0x29,
143.         0x8E, 0x33, 0x8E}; //
144.     // 替换为实际的目标 MAC 地址
145.     memset(&dest, 0, sizeof(dest));
146.     dest.sll_ifindex = ifr.ifr_ifindex;
147.     dest.sll_halen = ETH_ALEN;
148.     memcpy(dest.sll_addr, target_mac, ETH_ALEN);
```

```
146.         // 构造新的以太网帧头
147.         memcpy(eth_header->h_dest, target_mac, ETH_ALEN); // 目标 MAC 地
    址
148.         memcpy(eth_header->h_source, ifr_mac.ifr_hwaddr.sa_data,
149.                 ETH_ALEN); // 源 MAC 地址
150.         eth_header->h_proto = htons(ETH_P_IP); // 以太网类型为 IP
151.         printf("Interface name: %s, index: %d\n", ifr.ifr_name,
152.                 ifr.ifr_ifindex);
153.
154.         if (sendto(sockfd, buffer, data_size, 0, (struct sockaddr *)&des
    t,
155.                 sizeof(dest)) < 0) {
156.             perror("Sendto error");
157.             return 1;
158.         }
159.     }
160.     close(sockfd);
161.     free(buffer);
162.     return 0;
163. }
```

3) 使用GCC编译并运行程序: gcc -o route route.c

运行: ./route

(3) 接收主机程序

1) 创建一个文件recv.c

2) 编写以下代码实现发送功能

```
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. #define PORT 12345
8. int main()
9. {
10.     int sockfd;
11.     struct sockaddr_in server_addr, client_addr;
12.     socklen_t addr_len = sizeof(client_addr);
13.     char buffer[1024];
14.     // 创建 UDP 套接字
15.     sockfd = socket(AF_INET, SOCK_DGRAM, 0);
16.     if (sockfd < 0)
17.     {
```

```
18.     perror("Socket creation failed");
19.     return 1;
20. }
21. // 绑定套接字到端口
22. memset(&server_addr, 0, sizeof(server_addr));
23. server_addr.sin_family = AF_INET;
24. server_addr.sin_addr.s_addr = INADDR_ANY;
25. server_addr.sin_port = htons(PORT);
26. if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) <
    0)
27. {
28.     perror("Bind failed");
29.     return 1;
30. }
31. // 接收数据包
32. int recv_len = recvfrom(sockfd, buffer, sizeof(buffer) - 1, 0, (struct s
    ockaddr *)&client_addr, &addr_len);
33. if (recv_len < 0)
34. {
35.     perror("Recvfrom failed");
36.     return 1;
37. }
38. buffer[recv_len] = '\0';
39. printf("Received message: %s\n", buffer);
40. close(sockfd);
41. return 0;
42. }
```

3) 使用GCC编译并运行程序: gcc -o recv recv.c  
运行: ./recv

#### 4.3 选做内容

通过完善路由表, 改进示例程序实现双向传输。

(1) 发送主机程序

1) 创建一个文件send\_advanced.c

2) 编写以下代码实现发送功能

```
1. #include <arpa/inet.h>
2. #include <linux/if_packet.h>
3. #include <net/if.h>
4. #include <netinet/ether.h>
5. #include <netinet/ip.h>
6. #include <netinet/udp.h>
7. #include <stdio.h>
8. #include <stdlib.h>
```

```
9. #include <string.h>
10. #include <sys/ioctl.h>
11. #include <sys/socket.h>
12. #include <time.h>
13. #include <unistd.h>
14.
15. // 路由器 1 网口 1 的 MAC 地址
16. #define DEST_MAC0 0x00
17. #define DEST_MAC1 0x0C
18. #define DEST_MAC2 0x29
19. #define DEST_MAC3 0x08
20. #define DEST_MAC4 0x3C
21. #define DEST_MAC5 0x53
22.
23. #define ETHER_TYPE 0x0800
24. #define BUFFER_SIZE 1518
25.
26. #define UDP_SRC_PORT1 12345 // 发送用的端口
27. #define UDP_SRC_PORT2 54321 // 接收用的端口
28. #define UDP_DST_PORT1 12345 // 发送用的端口
29. #define UDP_DST_PORT2 54321 // 接收用的端口
30. #define UDP_SRC_IP "192.168.109.128"
31. #define UDP_FWD_IP1 "192.168.109.2"
32. #define UDP_FWD_IP2 "192.168.152.2"
33. #define UDP_DST_IP "192.168.152.128"
34.
35. unsigned short checksum(void *b, int len) {
36.     unsigned short *buf = b;
37.     unsigned int sum = 0;
38.     unsigned short result;
39.     for (sum = 0; len > 1; len -= 2) {
40.         sum += *buf++;
41.     }
42.     if (len == 1) sum += *(unsigned char *)buf;
43.     sum = (sum >> 16) + (sum & 0xFFFF);
44.     sum += (sum >> 16);
45.     result = ~sum;
46.     return result;
47. }
48.
49. /*
50. 以太网帧：以太网头 + IP 头 + UDP 头 + 载荷（信息）。
51. 构建 socket 地址告诉操作系统，此次 Socket 通信将会通过 ens33 网口发送往指定
52. MAC 地址，传输的层次为数据链路层使用以太网协议。
```

```
53.
54. vmhgfs-fuse .host:/ /mnt/hgfs; cd "/mnt/hgfs/Machine 1"
55. gcc t2-route.c; ./a.out
56. */
57.
58. int main() {
59.     char buffer[BUFFER_SIZE];
60.
61.     // 创建原始套接字
62.     int sockfd;
63.     if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) == -
    1) {
64.         perror("Socket");
65.         return 1;
66.     }
67.
68.     // 获取接口索引
69.     struct ifreq if_idx;
70.     memset(&if_idx, 0, sizeof(struct ifreq));
71.     strncpy(if_idx.ifr_name, "ens33", IFNAMSIZ - 1);
72.     if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0) {
73.         perror("SIOCGIFINDEX");
74.         return 1;
75.     }
76.     // 获取接口 MAC 地址
77.     struct ifreq if_mac;
78.     memset(&if_mac, 0, sizeof(struct ifreq));
79.     strncpy(if_mac.ifr_name, "ens33", IFNAMSIZ - 1);
80.     if (ioctl(sockfd, SIOCGIFHWADDR, &if_mac) < 0) {
81.         perror("SIOCGIFHWADDR");
82.         return 1;
83.     }
84.
85.     // 创建原始套接字
86.     int sockfd_recv;
87.     if ((sockfd_recv = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0) {
88.         perror("Socket");
89.         return 1;
90.     }
91.
92.     // 目标地址
93.     struct sockaddr_in recv_addr;
94.     recv_addr.sin_family = AF_INET;
95.     recv_addr.sin_port = htons(UDP_SRC_PORT2);
```

```
96.     recv_addr.sin_addr.s_addr = inet_addr(UDP_SRC_IP);
97.
98.     // 绑定套接字到本地地址
99.     if (bind(sockfd_recv, (struct sockaddr *)&recv_addr, sizeof(recv_addr))
100.    ) <
101.         0) {
102.             perror("Bind");
103.             return 1;
104.         } else {
105.             printf("[RECVER:%10u] Bind socket successfully.\n", time(0));
106.         }
107.     char message[256];
108.     memset(message, 0, sizeof(message));
109.     while (1) {
110.         printf("[SENDER:%10u] Please input message: ", time(0));
111.         fgets(message, sizeof(message), stdin);
112.
113.         message[strlen(message) - 1] = 0;
114.
115.         // 构造以太网头
116.         struct ether_header *eh = (struct ether_header *)buffer;
117.         eh->ether_shost[0] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[0];
118.         eh->ether_shost[1] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[1];
119.         eh->ether_shost[2] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[2];
120.         eh->ether_shost[3] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[3];
121.         eh->ether_shost[4] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[4];
122.         eh->ether_shost[5] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[5];
123.
124.         printf("[SENDER:%10u] SRC MAC Address = ", time(0));
125.         for (int i = 0; i < 6; i++) {
126.             printf("%02X%c", eh->ether_shost[i], ":\n"[i + 1 == 6]);
127.         }
128.
129.         eh->ether_dhost[0] = DEST_MAC0;
130.         eh->ether_dhost[1] = DEST_MAC1;
131.         eh->ether_dhost[2] = DEST_MAC2;
132.         eh->ether_dhost[3] = DEST_MAC3;
133.         eh->ether_dhost[4] = DEST_MAC4;
134.         eh->ether_dhost[5] = DEST_MAC5;
135.         eh->ether_type = htons(ETHER_TYPE);
136.
137.         printf("[SENDER:%10u] DST MAC Address = ", time(0));
138.         for (int i = 0; i < 6; i++) {
```



```
139.         printf("%02X%c", eh->ether_dhost[i], ":\n"[i + 1 == 6]);
140.     }
141.
142.     // 构造 IP 头
143.     struct iphdr *iph =
144.         (struct iphdr *)(buffer + sizeof(struct ether_header));
145.     iph->ihl = 5;
146.     iph->version = 4;
147.     iph->tos = 0;
148.     iph->tot_len = htons(sizeof(struct iphdr) + sizeof(struct udphdr
149.         ) +
150.             strlen(message));
151.     iph->id = htonl(UDP_DST_PORT2);
152.     iph->frag_off = 0;
153.     iph->ttl = 255;
154.     iph->protocol = IPPROTO_UDP;
155.     iph->check = 0;
156.     iph->saddr = inet_addr(UDP_SRC_IP);
157.     iph->daddr = inet_addr(UDP_DST_IP);
158.     iph->check = checksum((unsigned short *)iph, sizeof(struct iphdr
159.         ));
160.     printf("[SENDER:%10u] SRC IP Address = %s\n", time(0), UDP_SRC_I
161.         P);
162.     printf("[SENDER:%10u] DST IP Address = %s\n", time(0), UDP_DST_I
163.         P);
164.     printf("[SENDER:%10u] TTL = %d\n", time(0), iph->ttl);
165.
166.     // 构造 UDP 头
167.     struct udphdr *udph =
168.         (struct udphdr *)(buffer + sizeof(struct ether_header) +
169.             sizeof(struct iphdr));
170.     udph->source = htons(UDP_SRC_PORT1);
171.     udph->dest = htons(UDP_DST_PORT2);
172.     udph->len = htons(sizeof(struct udphdr) + strlen(message));
173.     udph->check = 0;
174.
175.     // 填充数据
176.     char *data = (char *)(buffer + sizeof(struct ether_header) +
177.         sizeof(struct iphdr) + sizeof(struct udphd
178.             r));
179.     strcpy(data, message);
180.
181.     // 设置 socket 地址结
```

```
178.     struct sockaddr_ll socket_address;
179.     socket_address.sll_ifindex = if_idx.ifr_ifindex;
180.     socket_address.sll_halen = ETH_ALEN;
181.     socket_address.sll_addr[0] = DEST_MAC0;
182.     socket_address.sll_addr[1] = DEST_MAC1;
183.     socket_address.sll_addr[2] = DEST_MAC2;
184.     socket_address.sll_addr[3] = DEST_MAC3;
185.     socket_address.sll_addr[4] = DEST_MAC4;
186.     socket_address.sll_addr[5] = DEST_MAC5;
187.
188.     // 发送数据包
189.     int len = sizeof(struct ether_header) + sizeof(struct iphdr) +
190.               sizeof(struct udphdr) + strlen(message);
191.     printf(
192.         "[SENDER:%10u] Sending message (load = [%s](%d)), total leng
193.         th = "
194.         "%dByte\n",
195.         time(0), message, strlen(message), len);
196.     if (sendto(sockfd, buffer, len, 0, (struct sockaddr *)&socket_ad
197.         dress,
198.             sizeof(struct sockaddr_ll)) < 0) {
199.         perror("Sendto");
200.         return 1;
201.     }
202.     // 接收数据报
203.     printf("[RECVER:%10u] Try to receive message.\n", time(0));
204.     // 接收数据报
205.     struct sockaddr_in send_addr;
206.     socklen_t addr_len = sizeof(send_addr);
207.
208.     int rcv_len = recvfrom(sockfd_rcv, message, sizeof(message), 0
209.         ,
210.         (struct sockaddr *)&send_addr, &addr_len
211.         );
212.     if (rcv_len < 0) {
213.         perror("Recvfrom");
214.         return 1;
215.     }
216.     message[rcv_len] = '\0';
217.     printf("[RECVER:%10u] Datagram received: [%s](%d).\n", time(0),
218.         message,
219.         strlen(message));
```

```
217.     }
218.     close(sockfd);
219.     close(sockfd_recv);
220.     return 0;
221. }
```

3) 使用GCC编译并运行程序: `gcc -o send_advanced send_advanced.c`

运行: `./send_advanced`

(2) 路由主机程序

1) 创建一个文件`route_advanced.c`

2) 编写以下代码实现路由功能

```
1. #include <arpa/inet.h>
2. #include <linux/if_packet.h>
3. #include <net/if.h>
4. #include <netinet/ether.h>
5. #include <netinet/if_ether.h>
6. #include <netinet/ip.h>
7. #include <netinet/udp.h>
8. #include <stdio.h>
9. #include <stdlib.h>
10. #include <string.h>
11. #include <sys/ioctl.h>
12. #include <sys/socket.h>
13. #include <time.h>
14. #include <unistd.h>
15.
16. #define BUFFER_SIZE 65536
17.
18. struct Hop {
19.     const char *addr; // 子网地址
20.     const char *mask; // 子网掩码
21.
22.     int ifr; // 网口出口
23. } table_hop[256];
24.
25. struct Mac {
26.     const char *addr; // IP 地址
27.
28.     // 对应的 MAC 地址
29.     unsigned char hop_mac0;
30.     unsigned char hop_mac1;
31.     unsigned char hop_mac2;
32.     unsigned char hop_mac3;
```

```
33.     unsigned char hop_mac4;
34.     unsigned char hop_mac5;
35. } table_mac[256];
36.
37. const char IFRS[][10] = { // 不同 IFC ID 对应的名称
38.     "", "ens33", "ens37"};
39.
40. unsigned short checksum(void *b, int len) {
41.     unsigned short *buf = b;
42.     unsigned int sum = 0;
43.     unsigned short result;
44.     for (sum = 0; len > 1; len -= 2) sum += *buf++;
45.     if (len == 1) sum += *(unsigned char *)buf;
46.     sum = (sum >> 16) + (sum & 0xFFFF);
47.     sum += (sum >> 16);
48.     result = ~sum;
49.     return result;
50. }
51.
52. unsigned int parse_ip(const char *s) {
53.     unsigned int ip[4] = {0, 0, 0, 0};
54.     int c = 0, w = 0;
55.     for (int p = 0; s[p]; ++p) {
56.         if (s[p] == '.') {
57.             ip[c++] = w, w = 0;
58.         } else {
59.             w = w * 10 + s[p] - '0';
60.         }
61.     }
62.     return ((ip[0] * 256 + ip[1]) * 256 + ip[2]) * 256 + ip[3];
63. }
64.
65. int main() {
66.     // 初始化地址表
67.     int table_mac_size = 2;
68.     table_mac[1].addr = "192.168.109.128";
69.     table_mac[1].hop_mac0 = 0x00;
70.     table_mac[1].hop_mac1 = 0x0C;
71.     table_mac[1].hop_mac2 = 0x29;
72.     table_mac[1].hop_mac3 = 0xA3;
73.     table_mac[1].hop_mac4 = 0x44;
74.     table_mac[1].hop_mac5 = 0xAE;
75.
76.     table_mac[2].addr = "192.168.152.128";
```

```
77.     table_mac[2].hop_mac0 = 0x00;
78.     table_mac[2].hop_mac1 = 0x0C;
79.     table_mac[2].hop_mac2 = 0x29;
80.     table_mac[2].hop_mac3 = 0x8E;
81.     table_mac[2].hop_mac4 = 0x33;
82.     table_mac[2].hop_mac5 = 0x8E;
83.
84.     printf("MAC Table:\n");
85.     printf("=====\n");
86.     printf("|      addr      |      mac      |\n");
87.     printf("+-----+-----+\n");
88.     // printf("+ 192.168.1.1 | aa:aa:aa:aa:aa:aa |\n");
89.     for (int i = 1; i <= table_mac_size; ++i) {
90.         printf("| %15s | %02X:%02X:%02X:%02X:%02X:%02X |\n", table_mac[i]
            .addr,
91.             table_mac[i].hop_mac0, table_mac[i].hop_mac1,
92.             table_mac[i].hop_mac2, table_mac[i].hop_mac3,
93.             table_mac[i].hop_mac4, table_mac[i].hop_mac5);
94.     }
95.     printf("=====\n");
96.
97.     // 初始化路由表
98.     int table_hop_size = 2;
99.
100.    table_hop[1].addr = "192.168.109.0";
101.    table_hop[1].mask = "255.255.255.0";
102.    table_hop[1].ifr = 1;
103.
104.    table_hop[2].addr = "192.168.152.0";
105.    table_hop[2].mask = "255.255.255.0";
106.    table_hop[2].ifr = 2;
107.
108.    printf("Route Table:\n");
109.    printf("=====\n");
110.    printf("|      addr      |      mask      | ifr |\n");
111.    printf("+-----+-----+-----+\n");
112.    for (int i = 1; i <= table_hop_size; ++i) {
113.        // printf("|      src_ip      |      dst_ip      | ifr |\n");
114.        printf("| %15s | %15s | %3d |\n", table_hop[i].addr,
            table_hop[i].mask, table_hop[i].ifr);
115.    }
116.
117.    printf("=====\n");
118.
119.    struct sockaddr saddr;
```

```
120.     char buffer[BUFFER_SIZE];
121.
122.     // 创建 UDP 套接字
123.     int sockfd;
124.     sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP));
125.     if (sockfd < 0) {
126.         perror("Socket");
127.         return 1;
128.     }
129.     while (1) {
130.         int saddr_len = sizeof(saddr);
131.         int data_size = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, &saddr,
132.                                   (socklen_t *)&saddr_len);
133.         if (data_size < 0) {
134.             perror("Recvfrom");
135.             return 1;
136.         }
137.
138.         struct ethhdr *eth_header = (struct ethhdr *)buffer;
139.         struct iphdr *ip_header =
140.             (struct iphdr *)(buffer + sizeof(struct ethhdr));
141.         struct udphdr *udph = (struct udphdr *)(buffer + sizeof(struct e
142.             thhdr) +
143.                                   sizeof(struct iphdr));
144.         char src_ip[INET_ADDRSTRLEN];
145.         char dst_ip[INET_ADDRSTRLEN];
146.         inet_ntop(AF_INET, &(ip_header->saddr), src_ip, INET_ADDRSTRLEN)
147.             ;
148.         inet_ntop(AF_INET, &(ip_header->daddr), dst_ip, INET_ADDRSTRLEN)
149.             ;
150.
151.         unsigned src_ip_val = parse_ip(src_ip);
152.         unsigned dst_ip_val = parse_ip(dst_ip);
153.
154.         int p = 0;
155.         for (int i = 1; i <= table_hop_size; ++i) {
156.             unsigned addr = parse_ip(table_hop[i].addr);
157.             unsigned mask = parse_ip(table_hop[i].mask);
158.
159.             if ((dst_ip_val & mask) == addr) {
160.                 p = table_hop[i].ifr;
161.                 break;
162.             }
163.         }
164.     }
```

```
160.     }
161.
162.     if (p == 0) { // 未在路由表里找到信息, 忽略
163.         // printf("[ROUTER:%10u] Ignored packet from %s to %s\n", ti
me(0),
164.             // src_ip, dst_ip);
165.         continue;
166.     }
167.
168.     int q = 0;
169.     for (int i = 1; i <= table_mac_size; ++i) {
170.         unsigned addr = parse_ip(table_mac[i].addr);
171.
172.         if ((!strcmp(dst_ip, table_mac[i].addr))) {
173.             q = i;
174.             break;
175.         }
176.     }
177.
178.     if (p == 0 || q == 0) { // 未在路由表里找到信息, 忽略
179.         // printf("[ROUTER:%10u] Ignored packet from %s to %s\n", ti
me(0),
180.             // src_ip, dst_ip);
181.         continue;
182.     }
183.
184.     // 获取当前系统时间
185.     time_t rawtime;
186.     struct tm *timeinfo;
187.     char time_str[100];
188.     time(&rawtime);
189.     timeinfo = localtime(&rawtime);
190.     // 格式化时间字符串
191.     strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S", timein
fo);
192.
193.     // 打印信息
194.     printf("[ROUTER:%10u] At %s captured packet from %s to %s\n", ti
me(0),
195.         time_str, src_ip, dst_ip);
196.     printf("[ROUTER:%10u] SRC_IP = %15s\n", time(0), src_ip);
197.     printf("[ROUTER:%10u] DST_IP = %15s\n", time(0), dst_ip);
198.     printf("[ROUTER:%10u] SRC_MAC = %02X:%02X:%02X:%02X:%02X:%02X\n"
,
,
```

```
199.         time(0), eth_header->h_source[0], eth_header->h_source[1]
200.         ,
201.         eth_header->h_source[2], eth_header->h_source[3],
202.         eth_header->h_source[4], eth_header->h_source[5]);
203.     printf("[ROUTER:%10u] DST_MAC = %02X:%02X:%02X:%02X:%02X:%02X\n"
204.         ,
205.         time(0), eth_header->h_dest[0], eth_header->h_dest[1],
206.         eth_header->h_dest[2], eth_header->h_dest[3],
207.         eth_header->h_dest[4], eth_header->h_dest[5]);
208.     // printf("[ROUTER:%10u] SRC Port = %d\n", time(0), udph->len);
209.
210.     // printf("[ROUTER:%10u] SRC Port = %d\n", time(0), udph->source
211.     );
212.     // printf("[ROUTER:%10u] DST Port = %d\n", time(0), udph->dest);
213.
214.     printf("[ROUTER:%10u] TTL = %d\n", time(0), ip_header->ttl);
215.
216.     // 修改 TTL
217.     ip_header->ttl -= 1;
218.     ip_header->check = 0;
219.     // ip_header->tot_len = htons(20 + 8 + 30); //
220.     // 总长度=IP 首部长+IP 数据长度
221.
222.     ip_header->check =
223.         checksum((unsigned short *)ip_header, ip_header->ihl * 4);
224.
225.     // 发送数据包到目的主机
226.     struct ifreq ifr, ifr_mac;
227.     struct sockaddr_ll hop;
228.
229.     // 获取网卡接口索引
230.     memset(&ifr, 0, sizeof(ifr));
231.     snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), IFRS[p]);
232.     if (ioctl(sockfd, SIOCGIFINDEX, &ifr) < 0) {
233.         perror("Ioctl");
234.         return 1;
235.     }
236.
237.     // 获取网卡接口 MAC 地址
238.     memset(&ifr_mac, 0, sizeof(ifr_mac));
239.     snprintf(ifr_mac.ifr_name, sizeof(ifr_mac.ifr_name), IFRS[p]);
240.     if (ioctl(sockfd, SIOCGIFHWADDR, &ifr_mac) < 0) {
241.         perror("Ioctl");
242.         return 1;
243.     }
244. }
```



```

238.
239.     // 设置目标 MAC 地址
240.     unsigned char target_mac[ETH_ALEN] = {
241.         table_mac[q].hop_mac0, table_mac[q].hop_mac1,
242.         table_mac[q].hop_mac2, table_mac[q].hop_mac3,
243.         table_mac[q].hop_mac4, table_mac[q].hop_mac5};
244.
245.     memset(&hop, 0, sizeof(hop));
246.     hop.sll_ifindex = ifr.ifr_ifindex;
247.     hop.sll_halen = ETH_ALEN;
248.     memcpy(hop.sll_addr, target_mac, ETH_ALEN);
249.     // 构造新的以太网帧头
250.     memcpy(eth_header->h_dest, target_mac, ETH_ALEN); // 目标 MAC 地
    址
251.     memcpy(eth_header->h_source, ifr_mac.ifr_hwaddr.sa_data,
252.         ETH_ALEN); // 源 MAC 地址
253.     eth_header->h_proto = htons(ETH_P_IP); // 以太网类型为 IP
254.
255.     printf("[ROUTER:%10u] Interface name: %s, index: %d\n", time(0),
256.         ifr.ifr_name, ifr.ifr_ifindex);
257.     if (sendto(sockfd, buffer, data_size, 0, (struct sockaddr *)&hop
258.         ,
259.         sizeof(hop)) < 0) {
260.         perror("Sendto");
261.         return 1;
262.     }
263.     printf("[ROUTER:%10u] Datagram forwarded.\n", time(0));
264.     close(sockfd);
265.     return 0;
266. }

```

3) 使用GCC编译并运行程序: gcc -o route\_advanced route\_advanced.c

运行: ./route\_advanced

(3) 接收主机程序

1) 创建一个文件recv\_advanced.c

2) 编写以下代码实现发送功能

```

1. #include <arpa/inet.h>
2. #include <linux/if_packet.h>
3. #include <net/if.h>
4. #include <netinet/ether.h>
5. #include <netinet/ip.h>

```

```
6. #include <netinet/udp.h>
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10. #include <sys/ioctl.h>
11. #include <sys/socket.h>
12. #include <time.h>
13. #include <unistd.h>
14.
15. // 路由器 1 网口 2 的 MAC 地址
16. #define DEST_MAC0 0x00
17. #define DEST_MAC1 0x0C
18. #define DEST_MAC2 0x29
19. #define DEST_MAC3 0x08
20. #define DEST_MAC4 0x3C
21. #define DEST_MAC5 0x5D
22.
23. #define ETHER_TYPE 0x0800
24. #define BUFFER_SIZE 1518
25.
26. #define UDP_SRC_PORT1 12345 // 发送用的端口
27. #define UDP_SRC_PORT2 54321 // 接收用的端口
28. #define UDP_DST_PORT1 12345 // 发送用的端口
29. #define UDP_DST_PORT2 54321 // 接收用的端口
30. #define UDP_SRC_IP "192.168.109.128"
31. #define UDP_FWD_IP1 "192.168.109.2"
32. #define UDP_FWD_IP2 "192.168.152.2"
33. #define UDP_DST_IP "192.168.152.128"
34.
35. unsigned short checksum(void *b, int len) {
36.     unsigned short *buf = b;
37.     unsigned int sum = 0;
38.     unsigned short result;
39.     for (sum = 0; len > 1; len -= 2) {
40.         sum += *buf++;
41.     }
42.     if (len == 1) sum += *(unsigned char *)buf;
43.     sum = (sum >> 16) + (sum & 0xFFFF);
44.     sum += (sum >> 16);
45.     result = ~sum;
46.     return result;
47. }
48.
49. /*
```

```
50. 以太网帧：以太网头 + IP 头 + UDP 头 + 载荷（信息）。
51. 构建 socket 地址告诉操作系统，此次 Socket 通信将会通过 ens33 网口发送往指定
52. MAC 地址，传输的层次为数据链路层使用以太网协议。
53. */
54.
55. int main() {
56.     char buffer[BUFFER_SIZE];
57.
58.     // 创建原始套接字
59.     int sockfd;
60.     if ((sockfd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL))) == -
    1) {
61.         perror("Socket");
62.         return 1;
63.     }
64.
65.     // 获取接口索引
66.     struct ifreq if_idx;
67.     memset(&if_idx, 0, sizeof(struct ifreq));
68.     strncpy(if_idx.ifr_name, "ens33", IFNAMSIZ - 1);
69.     if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0) {
70.         perror("SIOCGIFINDEX");
71.         return 1;
72.     }
73.     // 获取接口 MAC 地址
74.     struct ifreq if_mac;
75.     memset(&if_mac, 0, sizeof(struct ifreq));
76.     strncpy(if_mac.ifr_name, "ens33", IFNAMSIZ - 1);
77.     if (ioctl(sockfd, SIOCGIFHWADDR, &if_mac) < 0) {
78.         perror("SIOCGIFHWADDR");
79.         return 1;
80.     }
81.     // 创建原始套接字
82.     int sockfd_recv;
83.     if ((sockfd_recv = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0) {
84.         perror("Socket");
85.         return 1;
86.     }
87.     // 目标地址
88.     struct sockaddr_in recv_addr;
89.     recv_addr.sin_family = AF_INET;
90.     recv_addr.sin_port = htons(UDP_DST_PORT2);
91.     recv_addr.sin_addr.s_addr = inet_addr(UDP_DST_IP);
92.
```

```
93.    // 绑定套接字到本地地址
94.    if (bind(sockfd_recv, (struct sockaddr *)&recv_addr, sizeof(recv_addr)
    ) <
95.        0) {
96.        perror("Bind");
97.        return 1;
98.    } else {
99.        printf("[RECVER:%10u] Bind socket successfully.\n", time(0));
100.    }
101.
102.    char message[256];
103.    memset(message, 0, sizeof(message));
104.    while (1) {
105.        printf("[RECVER:%10u] Try to receive message.\n", time(0));
106.        // 接收数据报
107.        struct sockaddr_in send_addr;
108.        socklen_t addr_len = sizeof(send_addr);
109.
110.        int rcv_len = recvfrom(sockfd_recv, message, sizeof(message), 0
        ,
111.                                (struct sockaddr *)&send_addr, &addr_len
        );
112.        if (rcv_len < 0) {
113.            perror("Recvfrom");
114.            return 1;
115.        }
116.        message[rcv_len] = '\0';
117.        printf("[RECVER:%10u] Datagram received: [%s](%d).\n", time(0),
        message,
118.                strlen(message));
119.
120.        printf("[SENDER:%10u] Please input message: ", time(0));
121.        fgets(message, sizeof(message), stdin);
122.
123.        message[strlen(message) - 1] = 0;
124.
125.        // 构造以太网头
126.        struct ether_header *eh = (struct ether_header *)buffer;
127.        eh->ether_shost[0] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[0];
128.        eh->ether_shost[1] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[1];
129.        eh->ether_shost[2] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[2];
130.        eh->ether_shost[3] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[3];
131.        eh->ether_shost[4] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[4];
132.        eh->ether_shost[5] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[5];
```

```
133.
134.     printf("[SENDER:%10u] SRC MAC Address = ", time(0));
135.     for (int i = 0; i < 6; i++) {
136.         printf("%02X%c", eh->ether_shost[i], ":\n"[i + 1 == 6]);
137.     }
138.
139.     eh->ether_dhost[0] = DEST_MAC0;
140.     eh->ether_dhost[1] = DEST_MAC1;
141.     eh->ether_dhost[2] = DEST_MAC2;
142.     eh->ether_dhost[3] = DEST_MAC3;
143.     eh->ether_dhost[4] = DEST_MAC4;
144.     eh->ether_dhost[5] = DEST_MAC5;
145.     eh->ether_type = htons(ETHER_TYPE);
146.
147.     printf("[SENDER:%10u] DST MAC Address = ", time(0));
148.     for (int i = 0; i < 6; i++) {
149.         printf("%02X%c", eh->ether_dhost[i], ":\n"[i + 1 == 6]);
150.     }
151.     // 构造 IP 头
152.     struct iphdr *iph =
153.         (struct iphdr *) (buffer + sizeof(struct ether_header));
154.     iph->ihl = 5;
155.     iph->version = 4;
156.     iph->tos = 0;
157.     iph->tot_len = htons(sizeof(struct iphdr) + sizeof(struct udphdr
158.         ) + strlen(message));
159.     iph->id = htonl(UDP_SRC_PORT2);
160.     iph->frag_off = 0;
161.     iph->ttl = 255;
162.     iph->protocol = IPPROTO_UDP;
163.     iph->check = 0;
164.     iph->saddr = inet_addr(UDP_DST_IP);
165.     iph->daddr = inet_addr(UDP_SRC_IP);
166.     iph->check = checksum((unsigned short *)iph, sizeof(struct iphdr
167.         ));
168.     printf("[SENDER:%10u] SRC IP Address = %s\n", time(0), UDP_DST_I
169.         P);
170.     printf("[SENDER:%10u] DST IP Address = %s\n", time(0), UDP_SRC_I
171.         P);
172.     printf("[SENDER:%10u] TTL = %d\n", time(0), iph->ttl);
173.
174.     // 构造 UDP 头
175.     struct udphdr *udph =
```

```
173.         (struct udphdr *) (buffer + sizeof(struct ether_header) +
174.                               sizeof(struct iphdr));
175.         udph->source = htons(UDP_DST_PORT1);
176.         udph->dest = htons(UDP_SRC_PORT2);
177.         udph->len = htons(sizeof(struct udphdr) + strlen(message));
178.         udph->check = 0;
179.         // 填充数据
180.         char *data = (char *) (buffer + sizeof(struct ether_header) +
181.                                sizeof(struct iphdr) + sizeof(struct udphdr));
182.         strcpy(data, message);
183.         // 设置 socket 地址结构
184.         struct sockaddr_ll socket_address;
185.         socket_address.sll_ifindex = if_idx.ifr_ifindex;
186.         socket_address.sll_halen = ETH_ALEN;
187.         socket_address.sll_addr[0] = DEST_MAC0;
188.         socket_address.sll_addr[1] = DEST_MAC1;
189.         socket_address.sll_addr[2] = DEST_MAC2;
190.         socket_address.sll_addr[3] = DEST_MAC3;
191.         socket_address.sll_addr[4] = DEST_MAC4;
192.         socket_address.sll_addr[5] = DEST_MAC5;
193.         // 发送数据包
194.         int len = sizeof(struct ether_header) + sizeof(struct iphdr) +
195.                   sizeof(struct udphdr) + strlen(message);
196.         printf(
197.             "[SENDER:%10u] Sending message (load = [%s](%d)), total length = "
198.             "%dByte\n",
199.             time(0), message, strlen(message), len);
200.         if (sendto(sockfd, buffer, len, 0, (struct sockaddr *)&socket_address,
201.                    sizeof(struct sockaddr_ll)) < 0) {
202.             perror("Sendto");
203.             return 1;
204.         }
205.     }
206.     close(sockfd);
207.     close(sockfd_recv);
208.     return 0;
209. }
```

- 3) 使用GCC编译并运行程序: gcc -o recv\_advanced recv\_advanced.c  
运行: ./recv\_advanced

## 实验结果:

## 一、使用虚拟机实现多主机间的 UDP 数据报收发及转发

## (1) 必做内容

利用虚拟机搭建实验环境，掌握 Linux 下的 Socket 网络编程。

图 1-1 多主机单数据报收发与转发

同时打开源主机、转发机、目的主机，依次运行 ./recv\_ip、./forward\_ip、./send\_ip 通过观察三个主机的结果，发现源主机打印 Datagram sent，即发送成功；转发机打印 Datagram received 和 Datagram forward，即转发成功；目的主机打印 Data received，即接收成功。

## (2) 选做内容

改进程序，示例程序只实现了一个数据包（携带 1 条消息）的发、转、收过程，要求实现每条消息由控制台输入，并且不限制发送消息的数目。

图 1-2 多主机多数据报收发与转发

依次运行 ./recv\_ip\_advanced ./forward\_ip\_advanced、./send\_ip\_advanced，同时打开源主机、转发机、目的主机通过观察三个主机的结果，发现三个主机每条消息前面都有 [SENDER] [FORWARD] [RECVER] 的头信息，这里面携带每条消息的时间戳。

每次输入待发送数据后，发现源主机打印 Datagram sent，即发送成功；转发机打印 Datagram received 和 Datagram forward，即转发成功；目的主机打印 Data received，即接收成功。

## 二、基于单网口主机的 IP 数据转发及收发

## (1) 必做内容

在局域网中，模拟 IP 数据报的路由转发过程。通过原始套接字实现了完整的数据封装过程，实现了 UDP 头部、IP 头部、MAC 帧头部的构造。

图 2-1 基于单网口主机的单 IP 数据报转发与收发

同时打开源主机、路由机、目的主机，依次运行 ./myrecv、./myroute、./mysend 通过观察三个主机的结果，发现源主机打印自己的 MAC 地址和数据的长度，即发送成功；路由机打印 Captured packet from 192.168.102.130 to 192.168.102.131 和 Datagram forwarded，即捕获和转发成功；目的主机打印 Received message，即接收成功。

## (2) 选做内容

扩展实验的网络规模，由原始方案中 3 台主机增加到不少于 5 台主机，共同完成 IP 数据报转发及收发过程，要求采用转发表改进示例程序，增加程序通用性。



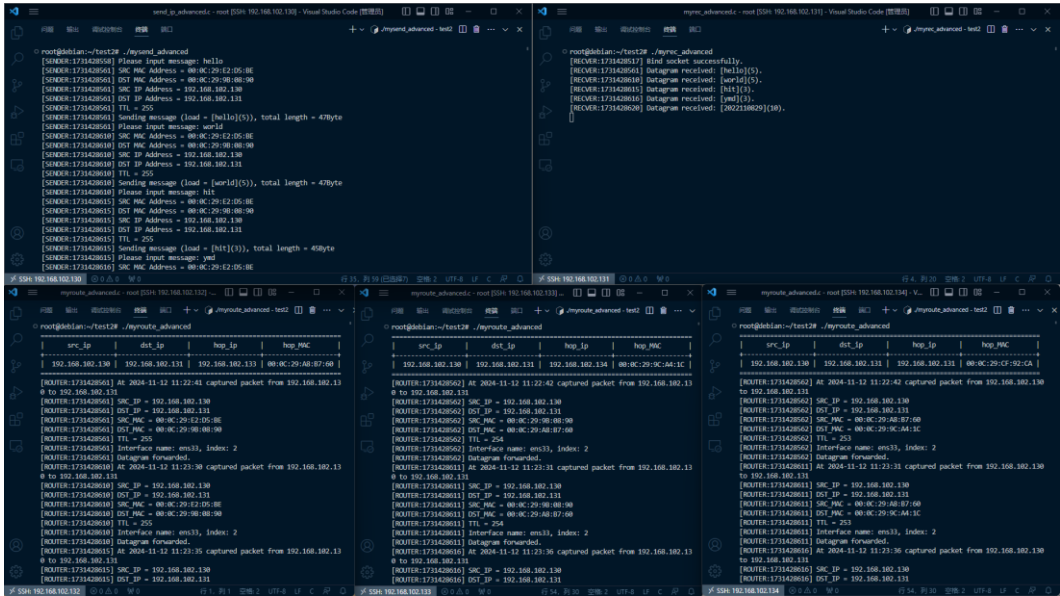


图 2-2 基于单网口主机的多 IP 数据报转发与收发

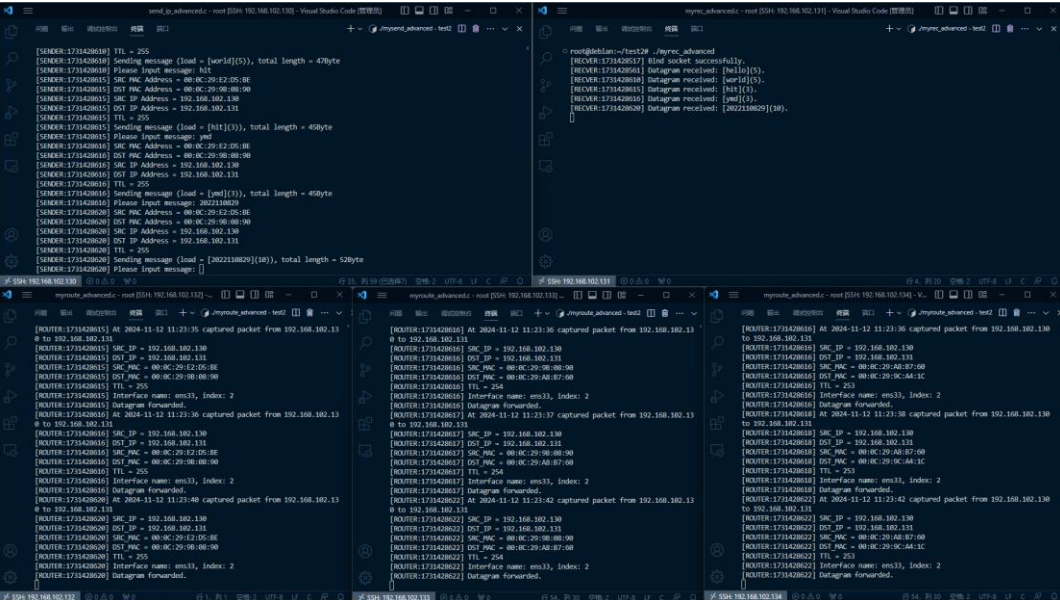


图 2-3 基于单网口主机的多 IP 数据报转发与收发

同时打开源主机、路由机 1、路由机 2、路由机 3、目的主机，依次运行./myrecv\_advanced、./myroute\_advanced、./myroute\_advanced、./myroute\_advanced、./mysend\_advanced 通过观察五个主机的结果，发现五个主机每条消息前面都有[SENDER][ROUTER][RECVER]的头信息，这里面携带每条消息的时间戳；源主机和路由主机都有源 IP 地址、目的 IP 地址、源 MAC 地址、目的 MAC 地址、TTL 时间和相关数据。

每次输入待发送数据后，发现源主机打印 Sent message，即发送成功；路由机打印 Captured packet from to 和 Datagram forward，即捕获和转发成功；目的主机打印 Data received，即接收成功。

### 三、基于双网口主机的路由转发

#### (1) 必做内容

构造了静态路由表，并实现了不同子网间的 IP 数据报查表转发过程。



```
root@debian:~/test3# ./send
Message sent to 192.168.152.128:12345
```

图 3-1 基于双网口主机的单数据报路由转发（源主机）

```
root@debian:~/test3# ./recv
Received message: Hello, this is a test message.
```

图 3-2 基于双网口主机的单数据报路由转发（目的主机）

```
root@debian:~/test3# ./route
-----IP Address: 192.168.152.128
-----IP Address: 192.168.152.128
Captured packet from 192.168.109.128 to 192.168.152.128
Interface name: ens37, index: 3
```

图 3-3 基于双网口主机的单数据报路由转发（路由主机）

同时打开源主机、路由主机、目的主机，依次运行./recv、./route、./send 通过观察三个主机结果，发现源主机打印 Message sent to 192.168.152.128:12345，即发送成功；路由主机打印 IP 地址和 Captured packet from to，即捕获和转发成功；目的主机打印 Received message，即接收成功。

## （2）选做内容

通过完善路由表，改进示例程序实现双向传输。

```
root@debian:~/test3# ./send_advanced
[RECV:1731429715] Bind socket successfully.
[SENDER:1731429715] Please input message: hello
[SENDER:1731429717] SRC MAC Address = 00:0C:29:A3:44:AE
[SENDER:1731429717] DST MAC Address = 00:0C:29:08:3C:53
[SENDER:1731429717] SRC IP Address = 192.168.109.128
[SENDER:1731429717] DST IP Address = 192.168.152.128
[SENDER:1731429717] TTL = 255
[SENDER:1731429717] Sending message (load = [hello](5)), total length = 47Byte
[RECV:1731429717] Try to receive message.
[RECV:1731429725] Datagram received: [world](5).
[SENDER:1731429725] Please input message: hit
[SENDER:1731429742] SRC MAC Address = 00:0C:29:A3:44:AE
[SENDER:1731429742] DST MAC Address = 00:0C:29:08:3C:53
[SENDER:1731429742] SRC IP Address = 192.168.109.128
[SENDER:1731429742] DST IP Address = 192.168.152.128
[SENDER:1731429742] TTL = 255
[SENDER:1731429742] Sending message (load = [hit](3)), total length = 45Byte
[RECV:1731429742] Try to receive message.
[RECV:1731429750] Datagram received: [yumd](3).
[SENDER:1731429750] Please input message:
```

图 3-4 基于双网口主机的多数据报双向路由转发（源主机）

```
root@debian:~/test3# ./recv_advanced
[RECV:1731429701] Bind socket successfully.
[RECV:1731429701] Try to receive message.
[RECV:1731429717] Datagram received: [hello](5).
[SENDER:1731429717] Please input message: world
[SENDER:1731429724] SRC MAC Address = 00:0C:29:0E:33:8E
[SENDER:1731429724] DST MAC Address = 00:0C:29:08:3C:5D
[SENDER:1731429724] SRC IP Address = 192.168.152.128
[SENDER:1731429724] DST IP Address = 192.168.109.128
[SENDER:1731429724] TTL = 255
[SENDER:1731429724] Sending message (load = [world](5)), total length = 47Byte
[RECV:1731429724] Try to receive message.
[RECV:1731429742] Datagram received: [hit](3).
[SENDER:1731429742] Please input message: ymd
[SENDER:1731429750] SRC MAC Address = 00:0C:29:0E:33:8E
[SENDER:1731429750] DST MAC Address = 00:0C:29:08:3C:5D
[SENDER:1731429750] SRC IP Address = 192.168.152.128
[SENDER:1731429750] DST IP Address = 192.168.109.128
[SENDER:1731429750] TTL = 255
[SENDER:1731429750] Sending message (load = [ymd](3)), total length = 45Byte
[RECV:1731429750] Try to receive message.
```

图 3-5 基于双网口主机的多数据报双向路由转发（目的主机）

```

root@debian:~/test3# ./route_advanced
MAC Table:
=====
|      addr      |      mac      |
+-----+-----+
| 192.168.109.128 | 00:0C:29:A3:44:AE |
| 192.168.152.128 | 00:0C:29:8E:33:8E |
+-----+-----+
Route Table:
=====
|      addr      |      mask      | ifr |
+-----+-----+-----+
| 192.168.109.0  | 255.255.255.0  | 1   |
| 192.168.152.0  | 255.255.255.0  | 2   |
+-----+-----+-----+
[ROUTER:1731429718] At 2024-11-12 11:41:58 captured packet from 192.168.109.128 to 192.168.152.128
[ROUTER:1731429718] SRC_IP = 192.168.109.128
[ROUTER:1731429718] DST_IP = 192.168.152.128
[ROUTER:1731429718] SRC_MAC = 00:0C:29:A3:44:AE
[ROUTER:1731429718] DST_MAC = 00:0C:29:08:3C:53
[ROUTER:1731429718] TTL = 255
[ROUTER:1731429718] Interface name: ens37, index: 3
[ROUTER:1731429718] Datagram forwarded.
[ROUTER:1731429725] At 2024-11-12 11:42:05 captured packet from 192.168.152.128 to 192.168.109.128
[ROUTER:1731429725] SRC_IP = 192.168.152.128
[ROUTER:1731429725] DST_IP = 192.168.109.128
[ROUTER:1731429725] SRC_MAC = 00:0C:29:8E:33:8E
[ROUTER:1731429725] DST_MAC = 00:0C:29:08:3C:5D
[ROUTER:1731429725] TTL = 255
[ROUTER:1731429725] Interface name: ens33, index: 2
[ROUTER:1731429725] Datagram forwarded.
[ROUTER:1731429742] At 2024-11-12 11:42:22 captured packet from 192.168.109.128 to 192.168.152.128
[ROUTER:1731429742] SRC_IP = 192.168.109.128
[ROUTER:1731429742] DST_IP = 192.168.152.128
[ROUTER:1731429742] SRC_MAC = 00:0C:29:A3:44:AE
[ROUTER:1731429742] DST_MAC = 00:0C:29:08:3C:53
[ROUTER:1731429742] TTL = 255
[ROUTER:1731429742] Interface name: ens37, index: 3
[ROUTER:1731429742] Datagram forwarded.
[ROUTER:1731429750] At 2024-11-12 11:42:30 captured packet from 192.168.152.128 to 192.168.109.128
[ROUTER:1731429750] SRC_IP = 192.168.152.128
[ROUTER:1731429750] DST_IP = 192.168.109.128
[ROUTER:1731429750] SRC_MAC = 00:0C:29:8E:33:8E
[ROUTER:1731429750] DST_MAC = 00:0C:29:08:3C:5D
[ROUTER:1731429750] TTL = 255
[ROUTER:1731429750] Interface name: ens33, index: 2
[ROUTER:1731429750] Datagram forwarded.
    
```

图 3-6 基于双网口主机的多数据报双向路由转发（路由主机）

依次运行 ./recv\_advanced、./route\_advanced、./send\_advanced，同时打开源主机、路由主机、目的主机，通过观察三个主机的结果，发现三个主机每条消息前面都有[SENDER][ROUTER][RECV]的头信息，这里面携带每条消息的时间戳；源主机和路由主机都有源 IP 地址、目的 IP 地址、源 MAC 地址、目的 MAC 地址、TTL 时间和相关数据。

在源主机向目的主机发信息时，每次输入待发送数据后，发现源主机打印 Sent message，即发送成功；路由机打印 Captured packet from to 和 Datagram forward，即捕获和转发成功；目的主机打印 Data received，即接收成功。

在目的主机向源主机发信息时，每次输入待发送数据后，发现目的主机打印 Sent message，即发送成功；路由机打印 Captured packet from to 和 Datagram forward，即捕获和转发成功；源主机打印 Data received，即接收成功。

```

root@debian:~/test3# ./send_advanced
[RECV:1731430413] Bind socket successfully.
[SENDER:1731430413] Please input message: Hello,this is a test message.
[SENDER:1731430431] SRC MAC Address = 00:0C:29:A3:44:AE
[SENDER:1731430431] DST MAC Address = 00:0C:29:08:3C:53
[SENDER:1731430431] SRC IP Address = 192.168.109.128
[SENDER:1731430431] DST IP Address = 192.168.152.128
[SENDER:1731430431] TTL = 255
[SENDER:1731430431] Sending message (load = [Hello,this is a test message.](29)), total length = 71Byte
[RECV:1731430431] Try to receive message.
[RECV:1731430471] Datagram received: [Hello! Got your message loud and clear.](39).
    
```

图 3-7 基于双网口主机的多数据报双向路由转发（源主机）

```

root@debian:~/test3# ./recv_advanced
[RECV:1731430407] Bind socket successfully.
[RECV:1731430407] Try to receive message.
[RECV:1731430430] Datagram received: [Hello,this is a test message.](29).
[SENDER:1731430430] Please input message: Hello! Got your message loud and clear.
[SENDER:1731430470] SRC MAC Address = 00:0C:29:8E:33:8E
[SENDER:1731430470] DST MAC Address = 00:0C:29:08:3C:5D
[SENDER:1731430470] SRC IP Address = 192.168.152.128
[SENDER:1731430470] DST IP Address = 192.168.109.128
[SENDER:1731430470] TTL = 255
[SENDER:1731430470] Sending message (load = [Hello! Got your message loud and clear.](39)), total length = 81Byte
[RECV:1731430470] Try to receive message.
    
```

图 3-8 基于双网口主机的多数据报双向路由转发（目的主机）

```

root@debian:~/test3# ./route_advanced
MAC Table:
=====
|      addr      |      mac      |
+-----+-----+
| 192.168.109.128 | 00:0C:29:A3:44:AE |
| 192.168.152.128 | 00:0C:29:8E:33:8E |
+-----+-----+
Route Table:
=====
|      addr      |      mask      | ifr |
+-----+-----+-----+
| 192.168.109.0  | 255.255.255.0  | 1   |
| 192.168.152.0  | 255.255.255.0  | 2   |
+-----+-----+-----+
[ROUTER:1731430431] At 2024-11-12 11:53:51 captured packet from 192.168.109.128 to 192.168.152.128
[ROUTER:1731430431] SRC_IP = 192.168.109.128
[ROUTER:1731430431] DST_IP = 192.168.152.128
[ROUTER:1731430431] SRC_MAC = 00:0C:29:A3:44:AE
[ROUTER:1731430431] DST_MAC = 00:0C:29:08:3C:53
[ROUTER:1731430431] TTL = 255
[ROUTER:1731430431] Interface name: ens37, index: 3
[ROUTER:1731430431] Datagram forwarded.
[ROUTER:1731430471] At 2024-11-12 11:54:31 captured packet from 192.168.152.128 to 192.168.109.128
[ROUTER:1731430471] SRC_IP = 192.168.152.128
[ROUTER:1731430471] DST_IP = 192.168.109.128
[ROUTER:1731430471] SRC_MAC = 00:0C:29:8E:33:8E
[ROUTER:1731430471] DST_MAC = 00:0C:29:08:3C:5D
[ROUTER:1731430471] TTL = 255
[ROUTER:1731430471] Interface name: ens33, index: 2
[ROUTER:1731430471] Datagram forwarded.
    
```

图 3-9 基于双网口主机的多数据报双向路由转发（路由主机）

#### 问题讨论：

（1）针对虚拟机界面无法做到多界面直观感受到多主机数据报的转发情况，可以用SSH服务器连接。用本地的VS Code连接虚拟机。步骤如下：

##### 1) 安装OpenSSH服务器

1. `sudo apt update`
2. `sudo apt install openssh-server`

##### 2) 启动SSH服务

1. `sudo systemctl start ssh`

##### 3) 设置开机启动

1. `sudo systemctl enable ssh`

##### 4) 检查SSH服务状态

1. `sudo systemctl status ssh`

##### 5) 配置防火墙

1. `sudo ufw allow ssh`
2. `sudo ufw status`

#### 6) 连接到SSH服务器

1. `ssh username@server_ip`

(2) 在“基于双网口主机的路由转发”实验中，第一次正常运行，而将虚拟机关机，再次打开运行程序时，会出现 `Network unreachable` 的报错，这是由于源主机和目的主机的网关设置是一次性的，每次开机需要重新配置。

1. `ip route add default via 192.168.109.2`
2. `ip route add default via 192.168.152.2`

(3) 在进行SSH连接是，输入`ssh root@192.168.***.***`时，回出现连接失败的问题，这是由于本着最小权限原则，为安全考虑，默认root账户没有SSH权限，如果要将权限解除。需要进行如下步骤：

- 1) 登录root状态
- 2) 编辑sshd的配置文件

1. `vim /etc/ssh/sshd_config`

#### 3) 把PermitRootLogin选项改成yes

```
Include /etc/ssh/sshd_config.d/*.conf

#Port 22
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::

#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key

# Ciphers and keying
#RekeyLimit default none

# Logging
#SyslogFacility AUTH
#LogLevel INFO

# Authentication:

#LoginGraceTime 2m
#PermitRootLogin prohibit-password
PermitRootLogin yes
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

#PubkeyAuthentication yes
```

图1 修改配置文件

#### 4) 重启SSH

1. `service sshd restart`

心得体会：

通过本次实验，我完成了使用 VMware Workstation Player 创建 Debian 虚拟机，使用虚拟机实现多主机间的 UDP 数据报收发及转发，基于单网口主机的 IP 数据转发及收发，基于双网口主机的路由转发。了解原始套接字的基本概念和使用方法，掌握路由器进行 IP 数据报转发的基本原理，实现于原始套接字的 IP 数据报的发送和接收，实现基于原始套接字的 IP 数据报转发，包括 AF\_INET 和 AF\_PACKET 原始套接字的应用。对数据报的收发及转发有了充分的了解，并且锻炼了 Linux Debian 环境操作能力及 SSH 任务学习。