



南开大学
Nankai University

南 开 大 学

网 络 空 间 安 全 学 院

网络技术与应用课程报告

简单路由程序设计

学号：2013018

姓名：许健

年级：2020 级

专业：信息安全

2022 年 12 月 24 日

目录

一、 实验内容说明	1
(一) 实验要求	1
(二) 预期实验效果	1
(三) 代码讲解内容	1
二、 实验准备工作	2
(一) 打开网卡获取双 IP	2
(二) 伪造 ARP 报文获取本机 MAC 地址	3
三、 路由表	4
(一) 路由表结构	4
(二) 路由表操作	5
1. 路由表项的添加	5
2. 路由表项的删除	7
3. 路由表项的查找	8
4. 打印路由表	9
四、 路由转发	9
(一) ARP 表	9
(二) 捕获报文过滤条件	10
(三) 捕获报文处理	11
1. 捕获 IP 报文处理	11
2. 捕获 ARP 报文处理	13
(四) 转发函数	13
五、 缓冲区	14
(一) 缓冲区结构	14
(二) 缓冲区相应操作	14
1. 查找缓冲区空余位置	14
2. 缓存数据包	15
3. 处理数据包	15
六、 程序功能演示	16
(一) 连通性测试	16
(二) 日志打印	16
七、 ICMP 报文发送	17
八、 附录	19
(一) 实验中使用的函数	19
(二) 数据报结构定义	19

一、 实验内容说明

(一) 实验要求

1. 设计和实现一个路由器程序，要求完成的路由器程序能和现有的路由器产品（如思科路由器、华为路由器、微软的路由器等）进行协同工作。
2. 程序可以仅实现 IP 数据报的获取、选路、投递等路由器要求的基本功能。可以忽略分片处理、选项处理、动态路由表生成等功能。
3. 需要给出路由表的手工插入、删除方法。
4. 需要给出路由器的工作日志，显示数据报获取和转发过程。
5. 完成的程序须通过现场测试，并在班（或小组）中展示和报告自己的设计思路、开发和实现过程、测试方法和过程。

(二) 预期实验效果

1. 获取并显示本机单网卡的双 IP 地址和 MAC 地址。
2. 可以显示路由表，可以添加路由表项 (206.1.3.0/255.255.255.0/206.1.2.2)，可以删除路由表项但无法删除默认路由表项。
3. 显示工作日志，输出数据报获取和转发过程（至少展示数据报的源 IP、目的 IP、下一跳信息）。
4. 主机 A 和 B 可以互相 ping 通，且显示的 TTL 正确 (126 或 62)。

(三) 代码讲解内容

1. 打开网卡获取双 IP
2. 伪造 ARP 报文获取本机 MAC 地址
3. 自动添加默认路由表项，手动添加删除路由表项，显示路由表
4. 捕获报文的过滤条件 (ARP IP)
5. 捕获 IP 报文的处理
6. 捕获 ARP 报文的处理
7. 缓冲区的超时删除

二、实验准备工作

(一) 打开网卡获取双 IP

使用 `pcap_findalldevs_ex` 函数可以获取本机设备列表，如果返回值为-1 则发生错误。`alldevs` 是指向设备列表的指针，类型为 `pcap_if`，如果是 `nullptr` 说明没有设备发现。可以使用 `for` 循环打印每个网卡的 `name` 和 `description` 信息，最后要使用 `pcap_freealldevs` 函数释放设备列表。

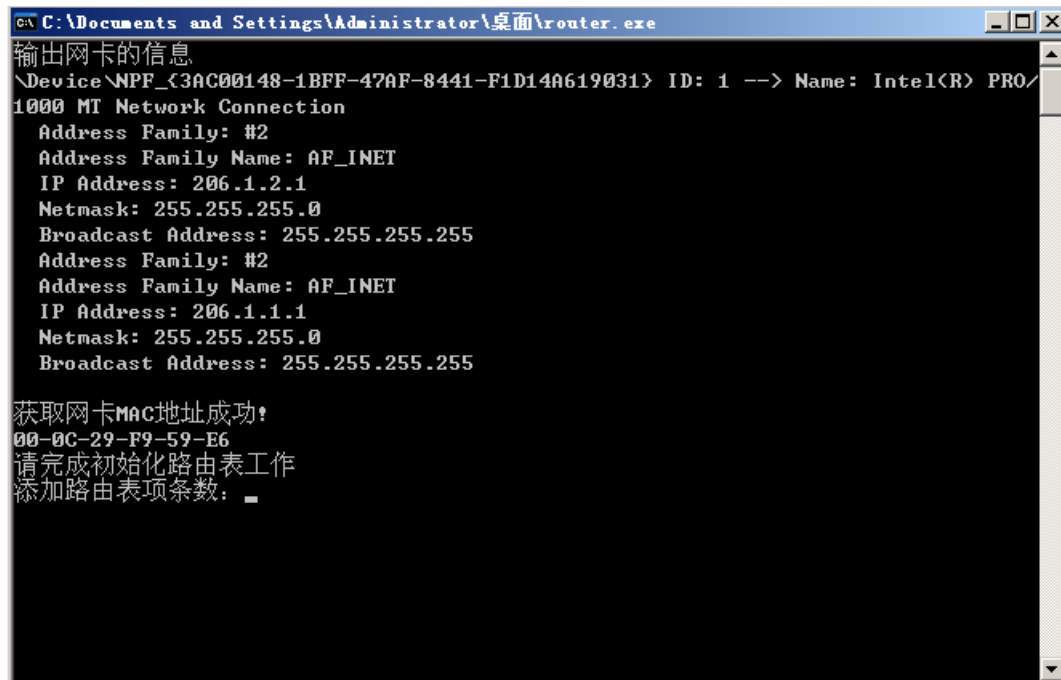


图 1: 打开网卡设备获取双 IP

打开网卡获取双 IP

```

1 void printAlldevs() {
2     int index = 0;
3     cout << "输出网卡的信息" << endl;
4     // 获取本地机器设备列表
5     if (pcap_findalldevs(&alldevs, errbuf) != -1){
6         /* 打印网卡信息列表 */
7         for (d = alldevs; d != NULL; d = d->next)
8         {
9             ++index;
10            if (d->description) {
11                printf("%s ID: %d --> Name: %s \n", d->name, index, d->
                description);
12            }
13            printIf(d);
14        }
15    }
16    pcap_freealldevs(alldevs);

```

```
17 }
18
19 void printIf(pcap_if_t* d) {
20     pcap_addr_t* a;
21     for (a = d->addresses; a; a = a->next)
22     {
23         if (a->addr->sa_family == AF_INET) {
24             printf("  Address Family: #%d\n", a->addr->sa_family);
25             printf("  Address Family Name: AF_INET\n");
26             if (a->addr) {
27                 printf("    IP Address: %s\n", inet_ntoa(((struct sockaddr_in*)
28                     a->addr->sin_addr)));
29             }
30             if (a->netmask) {
31                 printf("    Netmask: %s\n", inet_ntoa(((struct sockaddr_in*)a->
32                     netmask->sin_addr)));
33             }
34             if (a->broadaddr) {
35                 printf("    Broadcast Address: %s\n", inet_ntoa(((sockaddr_in*)
36                     a->broadaddr->sin_addr)));
37             }
38             if (a->dstaddr) {
39                 printf("    Destination Address: %s\n", inet_ntoa(((struct
40                     sockaddr_in*)a->dstaddr->sin_addr)));
41             }
42         }
43     }
44     printf("\n");
45 }
```

(二) 伪造 ARP 报文获取本机 MAC 地址

发送 ARP 包获取本物理网络内其它主机的 MAC 地址，首先需要获取本机的 MAC 地址。我们通过 `voidGetSelfMac(pcap_t*adhandle,constchar*ip_addr,unsignedchar*ip_mac);` 函数获取。其原理是伪造一个假的 ARP 地址解析包，虽然目标主机不存在，但是本机网卡并不知道，依然会给出回应，将 MAC 地址附在 ARP 响应包中发送出去。而我们可以捕获该数据包，从中提取出源 MAC 地址，即为本机 MAC 地址。

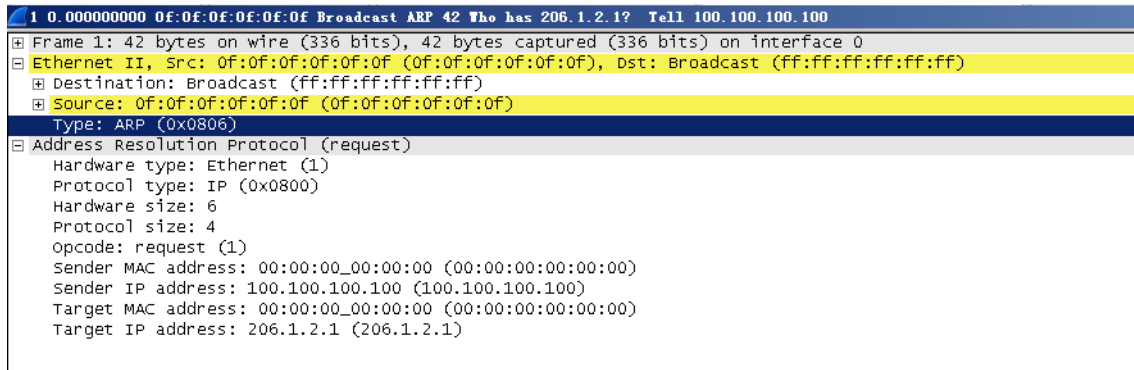


图 2: ARP 伪造数据包

伪造 ARP 报文获取本机 MAC 地址

```

1 void getSelfMac(pcap_t* adhandle, DWORD ip, BYTE* mac) {
2     ARP_t ARPFrame;
3     ARPFrame.FrameHeader = ETH_HRD_DEFAULT; // 以太网帧头
4     ARPFrame.ARPHeader = ARP_HRD_DEFAULT; // ARP帧头
5     ARPFrame.ARPHeader.SendIP = inet_addr("100.100.100.100");
6     ARPFrame.ARPHeader.RecvIP = ip;
7     if (pcap_sendpacket(adhandle, (const u_char*)&ARPFrame, 42) != 0) {
8         printf("error\n");
9         return;
10    }
11    int res;
12    while ((res = pcap_next_ex(adhandle, &pkt_header, &pkt_data)) >= 0) {
13        if (res == 0) continue;
14        if (*(WORD*)(pkt_data + 12) == htons(EH_TYPE)
15            && *(WORD*)(pkt_data + 20) == htons(ARP_REPLY)
16            && *(DWORD*)(pkt_data + 38)
17            == inet_addr("100.100.100.100")) {
18            for (int i = 0; i < 6; i++) {
19                mac[i] = *(BYTE*)(pkt_data + 22 + i);
20            }
21            printf("获取网卡MAC地址成功!\n");
22            break;
23        }
24    }
25 }

```

在图1中可以看到，网卡的 MAC 地址为 00-0C-29-F9-59-E6。

三、 路由表

(一) 路由表结构

路由表项由 mask 掩码、net 网络、nextip 下一跳、type 类型（区分是否为直接投递）四个字段，以及指向下一个路由表项的指针组成。路由表包含头节点、尾节点，以及记录表项数目的

num 字段。路由表采用链表的方式组织, 适合小规模网络。路由表的操作包括: 初始化、添加 (按照掩码长度排序)、删除、打印、查找等功能。

路由表项及路由表结构

```

1  class routeItem {
2  public:
3      DWORD mask;
4      DWORD net;
5      DWORD nextip;
6      int type;
7      routeItem* nextItem;
8      routeItem() {
9          memset(this, 0, sizeof(*this));
10     }
11     routeItem(char* mask, char* net, char* nextip) {
12         this->mask = inet_addr(mask);
13         this->net = inet_addr(net);
14         this->nextip = inet_addr(nextip);
15         this->type = 1;
16     }
17     void printItem();    //打印掩码、目的网络、下一跳IP和类型
18 };
19
20 class routeTable {
21 public:
22     routeItem* head, * tail;
23     int num;
24     //路由表的初始化, 添加直接投递项
25     routeTable();
26     //路由表的添加, 直接投递在最前, 前缀长的在前面
27     void add(routeItem* item);
28     //路由表的删除, 直接投递不可删除
29     void remove(int index);
30     //路由表的打印: mask、net、nextip、type
31     void printTable();
32     //查找最长前缀, 返回下一跳的ip
33     DWORD lookup(DWORD ip);
34     void printNum();
35 };

```

(二) 路由表操作

1. 路由表项的添加

路由表项按照 type 区分是否是直接投递项。直接投递项 type=0, 打开网卡设备时获取, 在路由表的初始化时自动添加到路由表的首部; 手动添加的路由表项 type=1, 会根据子网掩码的长度找到合适位置插入, 掩码越长插入的位置越靠前, 如图3所示。

```

请完成初始化路由表工作
添加路由表项条数: 1
掩码: 255.255.255.0
目的网络: 206.1.3.0
下一跳: 206.1.2.2
添加成功!
当前路由表
掩码:255.255.255.0      目的网络:206.1.1.0      下一跳:0.0.0.0      类型:0
掩码:255.255.255.0      目的网络:206.1.2.0      下一跳:0.0.0.0      类型:0
掩码:255.255.255.0      目的网络:206.1.3.0      下一跳:206.1.2.2      类型:1
路由功能正在启动...
路由功能已经启动!
=====程序功能=====
1:增加路由表项
2:删除路由表项
3:打印路由表
4:ARP查询
5:打印ARP表
请输入要进行的操作:

```

图 3: 初始化路由表项

```

请输入要进行的操作:
1
掩码: 255.255.255.128
目的网络: 1
下一跳: 1
=====程序功能=====
1:增加路由表项
2:删除路由表项
3:打印路由表
4:ARP查询
5:打印ARP表
请输入要进行的操作:
3
掩码:255.255.255.0      目的网络:206.1.1.0      下一跳:0.0.0.0      类型:0
掩码:255.255.255.0      目的网络:206.1.2.0      下一跳:0.0.0.0      类型:0
掩码:255.255.255.128     目的网络:0.0.0.1      下一跳:0.0.0.1      类型:1
掩码:255.255.255.0      目的网络:206.1.3.0      下一跳:206.1.2.2      类型:1
=====程序功能=====
1:增加路由表项
2:删除路由表项
3:打印路由表
4:ARP查询
5:打印ARP表
请输入要进行的操作:

```

图 4: 按照掩码长度插入路由表项

路由表项的添加

```

1 routeTable::routeTable() {
2     head = new routeItem;
3     tail = new routeItem;
4     head->nextItem = tail;
5     tail->type = -1;
6     num = 0;
7     for (int i = 0; i < 2; i++) {
8         routeItem* temp = new routeItem;
9         temp->net = ip[i] & mask[i];
10        temp->mask = mask[i];

```



```

11     temp->nextip = 0;
12     temp->type = 0;
13     this->add(temp);
14 }
15 }
16
17 void routeTable::add(routeItem* item){
18     routeItem* tmp;
19     if (item->type == 0) {
20         item->nextItem = head->nextItem;
21         head->nextItem = item;
22     }
23     else {
24         tmp = head->nextItem;
25         while (tmp->nextItem->type == 0 || item->mask < tmp->nextItem->mask)
26         {
27             if (tmp->nextItem == tail) {
28                 break;
29             }
30             tmp = tmp->nextItem;
31         }
32         item->nextItem = tmp->nextItem;
33         tmp->nextItem = item;
34     }
35     num++;
36 }

```

2. 路由表项的删除

根据索引 index 删除路由表项，判断索引的合法性；若为直接投递项不可删除，否则删除表项，并将 num 字段的值减 1。

```

请输入要进行的操作：
2
掩码:255.255.255.0    目的网络:206.1.1.0    下一跳:0.0.0.0    类型:0
掩码:255.255.255.0    目的网络:206.1.2.0    下一跳:0.0.0.0    类型:0
掩码:255.255.255.128  目的网络:0.0.0.1    下一跳:0.0.0.1    类型:1
掩码:255.255.255.0    目的网络:206.1.3.0    下一跳:206.1.2.2    类型:1
请选择要删除的表项index: 1
直接投递项不可删除

```

图 5: 直接投递项不可删除

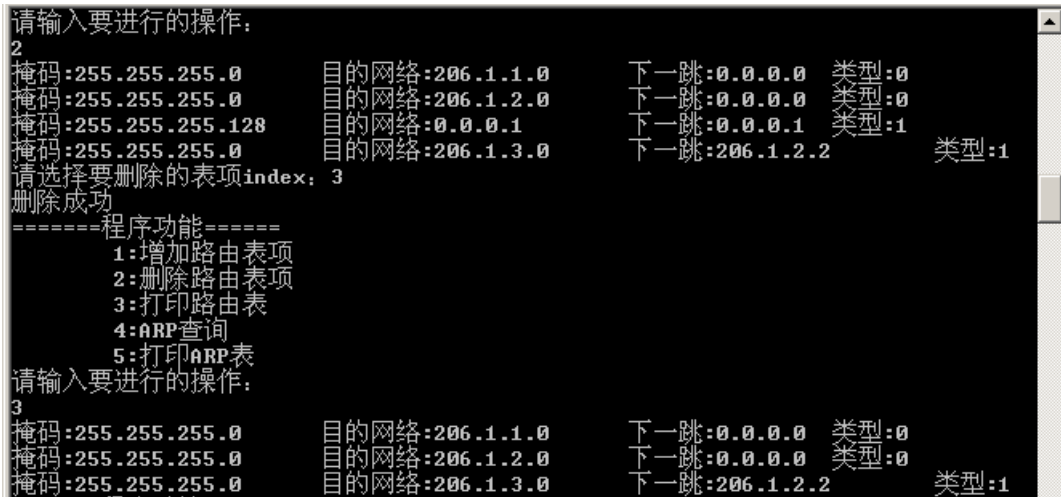


图 6: 删除路由表项

路由表项的删除

```

1 void routeTable::remove(int index){
2     if (index > num) {
3         printf(" 请删除索引正确的表项\n");
4         return;
5     }
6     routeItem* tmp = head;
7     while (--index) {
8         tmp = tmp->nextItem;
9     }
10    if (tmp->nextItem->type == 0) {
11        printf(" 直接投递项不可删除\n");
12        return;
13    }
14    routeItem* item = tmp->nextItem;
15    tmp->nextItem = tmp->nextItem->nextItem;
16    free(item);
17    num--;
18    printf(" 删除成功\n");
19 }

```

3. 路由表项的查找

路由表项按照直接投递项在前，非直接投递项按照子网掩码大小排序，找到的第一个匹配项即为要投递的下一跳。因此将 IP 与子网掩码相与，如果匹配网络号，则返回 nextip。

路由表项的查找

```

1 DWORD routeTable::lookup(DWORD ip) {
2     for (routeItem* tmp = head->nextItem; tmp != tail; tmp = tmp->nextItem) {
3         if ((ip & tmp->mask) == tmp->net) {
4             return tmp->nextip;

```

```

5         }
6     }
7     return -1;
8 }

```

4. 打印路由表

打印路由表项的掩码、目的网络、下一跳、类型，以字符串的形式输出。

打印路由表

```

1 void routeTable::printTable() {
2     for (routeItem* tmp = head->nextItem; tmp != tail; tmp = tmp->nextItem) {
3         tmp->printItem();
4     }
5     return;
6 }
7
8 void routeItem::printItem() {
9     in_addr addr;
10    addr.s_addr = mask;
11    printf("掩码:%s\t", inet_ntoa(addr));
12    addr.s_addr = net;
13    printf("目的网络:%s\t", inet_ntoa(addr));
14    addr.s_addr = nextip;
15    printf("下一跳:%s\t", inet_ntoa(addr));
16    printf("类型:%d\n", type);
17 };

```

四、 路由转发

(一) ARP 表

实验中采用结构数组的方式表示 ARP Table，数组的每一项包括 IP 地址和 MAC 地址，用于存储路由转发时需要的 IP-MAC 对应关系。ARP 表支持插入和查找操作，没有设置表项的超时删除。

ArpTable

```

1 class arpTable {
2 public:
3     DWORD ip;
4     BYTE mac[6];
5     static int num;
6     static void insert(DWORD ip, BYTE mac[6]);
7     static int lookup(DWORD ip, BYTE mac[6]);
8     static void printArp();
9 };
10 arpTable arptable[50];

```

```

11
12 void arpTable::insert(DWORD ip, BYTE mac[6]) {
13     if (num + 1 == 50) {
14         cout << "arp表已满，无法插入" << endl;
15         return;
16     }
17     arptable[num].ip = ip;
18     memcpy(arptable[num].mac, mac, 6);
19     num++;
20 }
21
22 int arpTable::lookup(DWORD ip, BYTE mac[6]) {
23     for (int index = 0; index < num; index++) {
24         if (arptable[index].ip == ip) {
25             memcpy(mac, arptable[index].mac, 6);
26             return index;
27         }
28     }
29     return -1;
30 }

```

(二) 捕获报文过滤条件

线程处理函数 *handlerRequest* 用于捕获数据包处理，对于收到的数据包要判断目的 MAC 是否等于网卡 MAC，如果相等则说明是发送给自己的数据包。然后根据以太网帧的 *FrameType* 字段区分是 IP 数据包 (0x0800) 还是 ARP 数据包 (0x0806)，针对两种数据包进行不同的处理。

线程处理函数 *handlerRequest*

```

1 DWORD WINAPI handlerRequest(LPVOID lparam)
2 {
3     routeTable* table = (routeTable*)(LPVOID)lparam;
4     pcap_pkthdr* pkt_header;
5     const u_char* pkt_data;
6     while (1) {
7         while (1) {
8             int rtn = pcap_next_ex(adhandle, &pkt_header, &pkt_data);
9             if (rtn) break;
10        }
11        IP_t* data = (IP_t*)pkt_data;
12        if (compare(data->FrameHeader.DesMAC, mac) && ntohs(data->FrameHeader
13            .FrameType) == 0x0800) {
14            // 捕获IP报文处理...
15        }
16        else if (compare(data->FrameHeader.DesMAC, mac) && ntohs(data->
17            FrameHeader.FrameType) == 0x0806) {
18            // 捕获ARP报文处理...
19        }
20    }
21 }

```

19

(三) 捕获报文处理

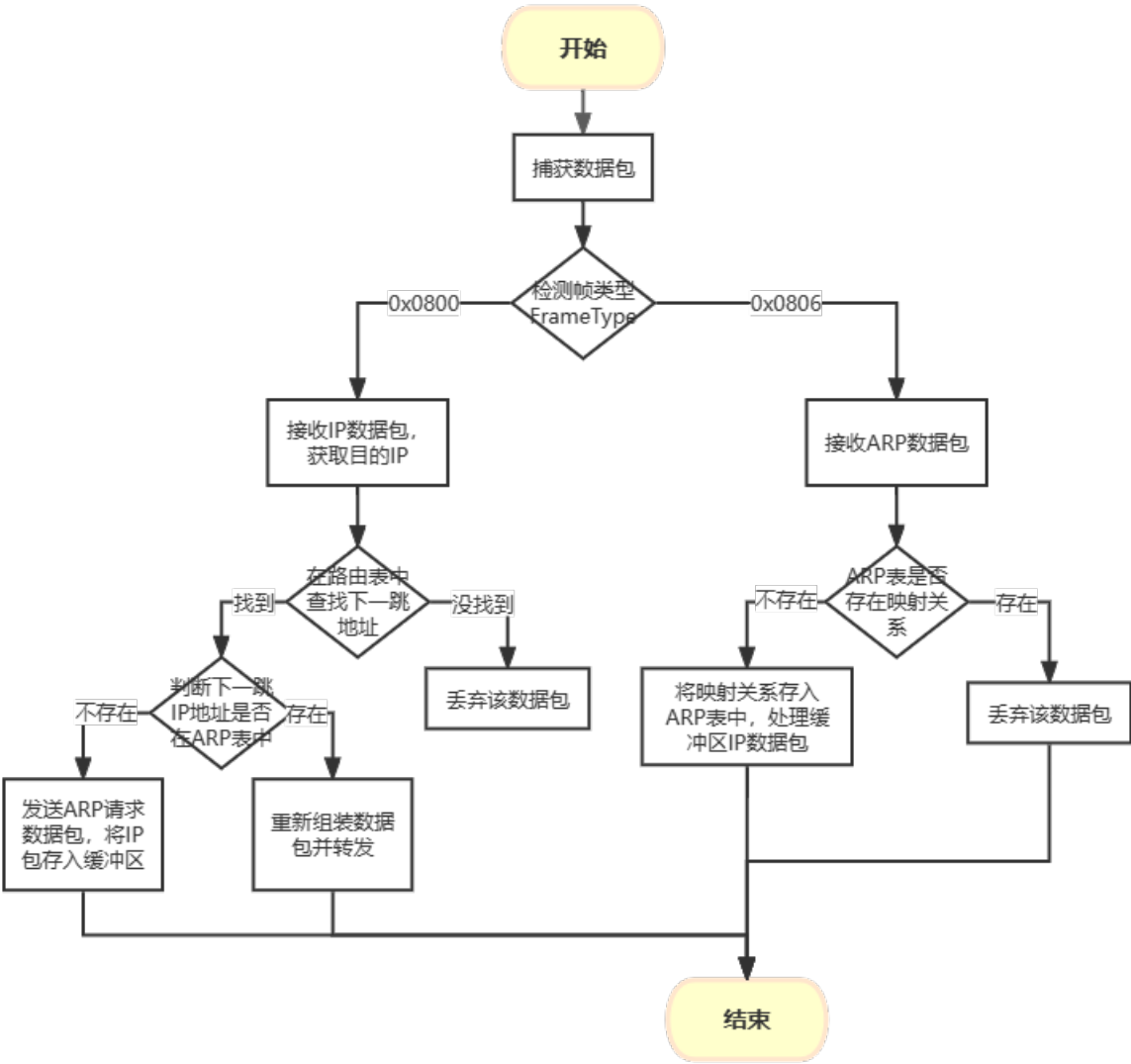


图 7: 处理 IP/ARP 数据包流程

1. 捕获 IP 报文处理

IP 数据包的转发又分为直接投递和非直接投递，重新组装数据包时传入的参数不同。直接投递的目的 MAC 地址就是目的 IP 的 MAC 地址，非直接投递的目的 MAC 地址是下一跳 IP 的 MAC 地址。

对于 ARP 表中找不到 IP-MAC 对应转发的数据包，需要暂时缓存，并发送 ARP 请求数据包。

捕获 IP 报文处理

```
1 printf("[接收]\t源IP: ");
```

```

2  printIp(data->IPHeader.SrcIP);
3  printf("\t目的IP: ");
4  printIp(data->IPHeader.DstIP);
5  printf("\tTTL: %d\n", data->IPHeader.TTL);
6  DWORD dstip = data->IPHeader.DstIP;
7  //查找路由表中是否有对应表项
8  DWORD nextip = table->lookup(dstip);
9  //没有则直接丢弃
10 if (nextip == -1)continue;
11 //校验和不正确直接丢弃
12 if (!Checksum(data))continue;

13 if (data->IPHeader.DstIP != ip[0] && data->IPHeader.DstIP != ip[1]) {

14     //接收者不是自己
15     BYTE broadmac[6] = { 0xff,0xff,0xff,0xff,0xff,0xff };
16     int t1 = compare(data->FrameHeader.DesMAC, broadmac);
17     int t2 = compare(data->FrameHeader.SrcMAC, broadmac);
18     if (!t1 && !t2) { //不是广播消息

19         ICMP_t temp = *(ICMP_t*)pkt_data;
20         BYTE dstmac[6];
21         //直接投递, 查找目的IP的MAC
22         if (nextip == 0)

23             {
24                 if (arpTable::lookup(temp.IPHeader.DstIP, dstmac) == -1) {
25                     sendARP(adhandle, temp.IPHeader.DstIP);
26                     cache(buffer, BufferSize, &temp, temp.IPHeader.DstIP);
27                     continue;
28                 }
29                 resend(adhandle, temp, nextip, dstmac);
30             }
31         //非直接投递, 查找下一条IP的MAC
32         else {

33             if (arpTable::lookup(nextip, dstmac) == -1) {
34                 sendARP(adhandle, nextip);
35                 cache(buffer, BufferSize, &temp, nextip);
36                 continue;
37             }
38             resend(adhandle, temp, nextip, dstmac);
39         }
40     }
41 }

```

2. 捕获 ARP 报文处理

捕获 ARP 报文用于更新 ARP 表项，对于由于缺少 IP-MAC 对应关系而存入缓冲区的 IP 数据包，进行处理转发。

捕获 ARP 报文处理

```

1 ARP_t* arpPacket = (ARP_t*)pkt_data;
2 if (ntohs(arpPacket->ARPHeader.Operation) == ARP_REPLY) {
3     DWORD ip_ = arpPacket->ARPHeader.SendIP;
4     BYTE mac_[6] = { 0 };
5     memcpy(mac_, arpPacket->ARPHeader.SendHa, 6);
6     if (arpTable::lookup(ip_, mac_) == -1) {
7         arpTable::insert(ip_, mac_);
8         printf("[ARP]\tIP: ");
9         printIp(ip_);
10        printf("\tMAC: ");
11        printMac(mac_);
12        printf("\n");
13    }
14    handle(buffer, BufferSize);
15 }

```

(四) 转发函数

resend 函数的工作包括将 TTL 字段减 1，重新计算校验和、打印转发日志，更换源、目的 MAC 地址，重新转发数据包。

resend 函数

```

1 void resend(pcap_t* adhandle, ICMP_t data, DWORD nextip, BYTE dstMac[]) {
2     IP_t* temp = (IP_t*)&data;
3     if (--(temp->IPHeader.TTL) == 0) {
4         //ICMP超时报文发送.....
5         //ICMPPacketProc(11, 0, (u_char*)&data));
6         return;
7     }
8     memcpy(temp->FrameHeader.SrcMAC, temp->FrameHeader.DesMAC, 6);
9     memcpy(temp->FrameHeader.DesMAC, dstMac, 6);
10    setChecksum(temp);
11    int rtn = pcap_sendpacket(adhandle, (const u_char*)temp, sizeof(data));
12    if (rtn == 0) {
13        printf("[转发]\t源IP: ");
14        printIp(temp->IPHeader.SrcIP);
15        printf("\t目的IP: ");
16        printIp(temp->IPHeader.DstIP);
17        printf("\t下一跳: ");
18        if (nextip == 0) {
19            printf("直接投递");
20        }

```

```

21     else {
22         printIp(nextip);
23     }
24     printf("\tTTL: %d\n", temp->IPHeader.TTL);
25 }
26 }

```

五、 缓冲区

(一) 缓冲区结构

数据包缓冲区采用数据实现，数组中每一项包含存放数据包的 Data 区域，目的 IP、缓冲区有效位 valid（转发或超时置为 0，置为 1 表示待处理），时钟 clock（用于数据包超时判断）

Buffer 缓冲区

```

1  class Buffer {
2  public:
3  BYTE Data[DataSize];           //数据部分
4  WORD TargetIP;                 //目的IP
5  bool valid = 1;                //有效位，转发或超时置0
6  clock_t clock;                 //超时判断
7  };
8  extern Buffer buffer[BufferSize];

```

(二) 缓冲区相应操作

针对缓冲区的操作包括：查找缓冲区空余位置、缓存数据包、处理缓冲区

```

1  //查找缓冲区中空余位置
2  int find(Buffer* buffer, int size);
3  //处理缓冲区
4  void handle(Buffer* buffer, int size);
5  //缓存数据包
6  void cache(Buffer* buffer, int size, ICMP_t* data, DWORD TargetIP);

```

1. 查找缓冲区空余位置

对缓冲区数组进行遍历，查找 valid 为 0 的第一个下标，即为可用的位置，否则返回-1。

find 函数

```

1  int find(Buffer* buffer, int size) {
2      for (int i = 0; i < size; i++) {
3          if (buffer[i].valid == 0) {
4              return i;
5          }
6      }

```



```

7     return -1;
8 }

```

2. 缓存数据包

查找缓冲区空余位置，如果找到，将数据包放入数据缓冲区，设置时钟和有效位，记录目的IP，等待处理。

cache 函数

```

1 void cache(Buffer* buffer, int size, ICMP_t* data, DWORD TargetIP) {
2     int index = find(buffer, BufferSize);
3     if (index == -1) {
4         cout << "缓冲区已满，丢弃" << endl;
5         return;
6     }
7     memset(&buffer[index].Data, 0, DataSize);
8     memcpy(&buffer[index].Data, data, sizeof(ICMP_t));
9     buffer[index].TargetIP = data->IPHeader.DstIP;
10    buffer[index].clock = clock();
11    buffer[index].valid = 1;
12 }

```

3. 处理数据包

遍历缓冲区，跳过超时、已经处理的数据包，对于 ARP 表中已经存在映射关系的数据包进行处理，调用 resend 函数转发，并将缓冲区置为空闲（valid 为 0）。

handle 函数

```

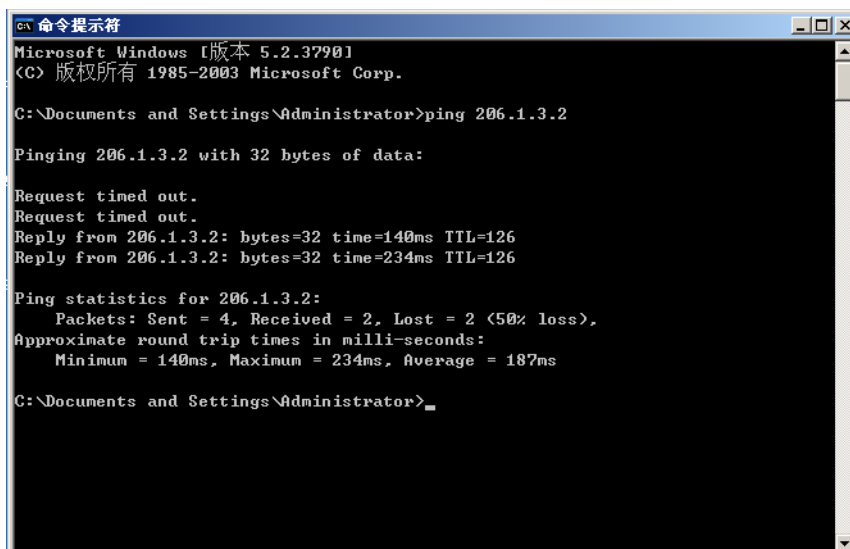
1 void handle(Buffer* buffer, int size) {
2     for (int i = 0; i < size; i++) {
3         if (buffer[i].valid == 0) { continue; }
4         if ((clock() - buffer[i].clock) > BufferTime) {
5             buffer[i].valid = 0;
6             continue;
7         }
8         BYTE mac_[6];
9         if (arpTable::lookup(buffer[i].TargetIP, mac_) != -1) {
10            ICMP_t tmp;
11            memcpy(&tmp, 0, sizeof(tmp));
12            memcpy(&tmp, &buffer[i].Data, sizeof(tmp));
13            resend(adhandle, tmp, buffer[i].TargetIP, mac_);
14            buffer[i].valid = 0;
15        }
16    }
17 }

```

六、 程序功能演示

(一) 连通性测试

由于程序刚运行时 ARP 表为空, IP 数据包需要存入缓冲区等待处理, 因此首次 A ping B 的 ICMP 数据包显示超时, 当 B ping A 时, 路由器已经存储 IP-MAC 映射关系, 不会存在处理超时的问题。



```
命令提示符
Microsoft Windows [版本 5.2.3790]
(C) 版权所有 1985-2003 Microsoft Corp.

C:\Documents and Settings\Administrator>ping 206.1.3.2

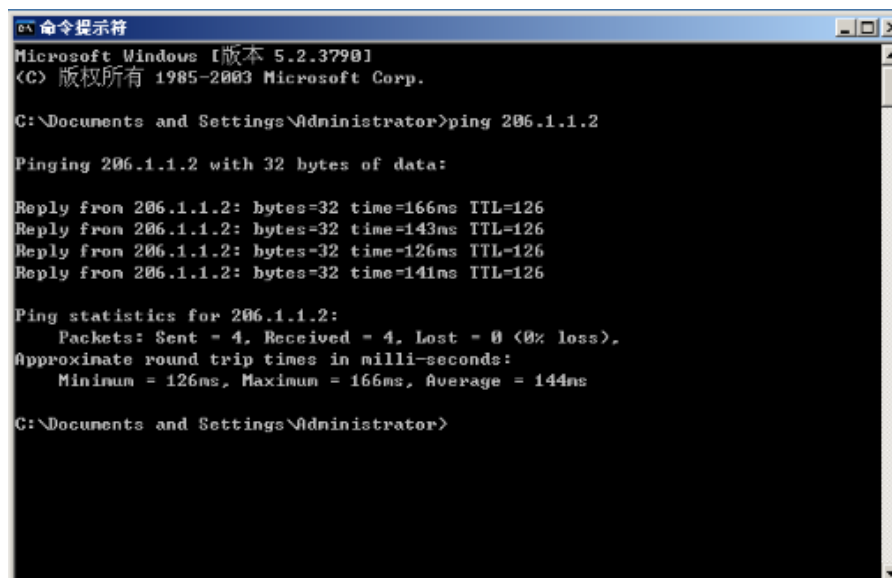
Pinging 206.1.3.2 with 32 bytes of data:

Request timed out.
Request timed out.
Reply from 206.1.3.2: bytes=32 time=140ms TTL=126
Reply from 206.1.3.2: bytes=32 time=234ms TTL=126

Ping statistics for 206.1.3.2:
    Packets: Sent = 4, Received = 2, Lost = 2 (50% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 140ms, Maximum = 234ms, Average = 187ms

C:\Documents and Settings\Administrator>
```

图 8: A ping B



```
命令提示符
Microsoft Windows [版本 5.2.3790]
(C) 版权所有 1985-2003 Microsoft Corp.

C:\Documents and Settings\Administrator>ping 206.1.1.2

Pinging 206.1.1.2 with 32 bytes of data:

Reply from 206.1.1.2: bytes=32 time=166ms TTL=126
Reply from 206.1.1.2: bytes=32 time=143ms TTL=126
Reply from 206.1.1.2: bytes=32 time=126ms TTL=126
Reply from 206.1.1.2: bytes=32 time=141ms TTL=126

Ping statistics for 206.1.1.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 126ms, Maximum = 166ms, Average = 144ms

C:\Documents and Settings\Administrator>
```

图 9: B ping A

(二) 日志打印

日志记录了 IP 数据包接收和转发的信息, 可以看到转发时路由器将 TTL 字段减 1, 日志记录源 IP、目的 IP、下一跳、TTL 字段等信息。对于 ARP 响应包同样有日志打印, 记录 IP-MAC

地址对应关系，已经获得的对应关系会存在 ARP 表中。打印的日志可以和 wireshark 捕获的数据包对应，进一步验证我们的路由转发程序正常工作。

```

[接收] 源IP: 206.1.1.2 目的IP: 206.1.3.2 TTL: 128
[ARP] IP: 206.1.2.2 MAC: 00-0C-29-E5-60-4F
[接收] 源IP: 206.1.1.2 目的IP: 206.1.3.2 TTL: 128
[转发] 源IP: 206.1.1.2 目的IP: 206.1.3.2 下一跳: 206.1.2.2 TTL: 127

[接收] 源IP: 206.1.3.2 目的IP: 206.1.1.2 TTL: 127
[ARP] IP: 206.1.1.2 MAC: 00-0C-29-C2-F2-7D
[接收] 源IP: 206.1.1.2 目的IP: 206.1.3.2 TTL: 128
[转发] 源IP: 206.1.1.2 目的IP: 206.1.3.2 下一跳: 206.1.2.2 TTL: 127

[接收] 源IP: 206.1.3.2 目的IP: 206.1.1.2 TTL: 127
[转发] 源IP: 206.1.3.2 目的IP: 206.1.1.2 下一跳: 直接投递 TTL: 126

[接收] 源IP: 206.1.1.2 目的IP: 206.1.3.2 TTL: 128
[转发] 源IP: 206.1.1.2 目的IP: 206.1.3.2 下一跳: 206.1.2.2 TTL: 127

[接收] 源IP: 206.1.3.2 目的IP: 206.1.1.2 TTL: 127
[转发] 源IP: 206.1.3.2 目的IP: 206.1.1.2 下一跳: 直接投递 TTL: 126

```

图 10: 日志打印

10 223.502040 00:0c:29:c2:f2:7d Broadcast ARP 60 who has 206.1.1.1? Tell 206.1.1.2	
11 223.502087 00:0c:29:c2:f2:7d ARP 42 206.1.1.1 is at 00:0c:29:f9:59:e6	
12 223.502312 206.1.1.2 206.1.3.2 ICMP 74 echo (ping) request 1d-0x0200, seq=256/1, ttl=128 (no response found)	
13 223.543075 00:0c:29:f9:59:e6 Broadcast ARP 42 who has 206.1.2.2? Tell 206.1.1.1	
14 223.543455 00:0c:29:f9:59:e6 ARP 60 206.1.2.2 is at 00:0c:29:e5:60:4f	
15 224.425195 239.168.177.1 239.255.255.250 SSDP 217 M-SEARCH * HTTP/1.1	
16 225.142057 239.168.177.1 239.255.255.250 SSDP 217 M-SEARCH * HTTP/1.1	
17 226.161830 239.168.177.1 239.255.255.250 SSDP 217 M-SEARCH * HTTP/1.1	
18 227.377126 239.168.177.1 239.255.255.250 SSDP 217 M-SEARCH * HTTP/1.1	
19 228.734473 206.1.1.2 206.1.3.2 ICMP 74 echo (ping) request 1d-0x0200, seq=512/2, ttl=128 (no response found)	
20 228.797647 206.1.1.2 206.1.3.2 ICMP 94 echo (ping) request 1d-0x0200, seq=512/2, ttl=127 (no response found) [ETHERNET FRAME CHECK SEQUENCE INCORRECT]	
21 228.797908 00:0c:29:e5:60:4f Broadcast ARP 60 who has 206.1.3.2? Tell 206.1.1.1	
22 228.798105 00:0c:29:e5:60:4f ARP 60 206.1.3.2 is at 00:0c:29:63:18:22	
23 228.798109 206.1.1.2 206.1.3.2 ICMP 94 echo (ping) request 1d-0x0200, seq=512/2, ttl=126 (reply in 24) [ETHERNET FRAME CHECK SEQUENCE INCORRECT]	
24 228.798370 206.1.3.2 206.1.1.2 ICMP 74 echo (ping) reply 1d-0x0200, seq=512/2, ttl=128 (request in 23)	
25 228.798371 206.1.3.2 206.1.1.2 ICMP 74 echo (ping) reply 1d-0x0200, seq=512/2, ttl=127	
26 228.906393 00:0c:29:f9:59:e6 Broadcast ARP 42 who has 206.1.1.1? Tell 206.1.1.1	
27 228.906847 00:0c:29:c2:f2:7d ARP 60 206.1.1.2 is at 00:0c:29:c2:f2:7d	
28 234.237850 206.1.1.2 206.1.3.2 ICMP 74 echo (ping) request 1d-0x0200, seq=768/3, ttl=128 (no response found)	
29 234.267740 206.1.1.2 206.1.3.2 ICMP 94 echo (ping) request 1d-0x0200, seq=768/3, ttl=127 (no response found) [ETHERNET FRAME CHECK SEQUENCE INCORRECT]	
30 234.267927 206.1.1.2 206.1.3.2 ICMP 94 echo (ping) request 1d-0x0200, seq=768/3, ttl=126 (reply in 31) [ETHERNET FRAME CHECK SEQUENCE INCORRECT]	
31 234.268059 206.1.3.2 206.1.1.2 ICMP 74 echo (ping) reply 1d-0x0200, seq=768/3, ttl=128 (request in 30)	
32 234.268222 206.1.3.2 206.1.1.2 ICMP 74 echo (ping) reply 1d-0x0200, seq=768/3, ttl=127	
33 234.378202 206.1.3.2 206.1.1.2 ICMP 94 echo (ping) reply 1d-0x0200, seq=768/3, ttl=126 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]	
34 235.255152 206.1.1.2 206.1.3.2 ICMP 74 echo (ping) request 1d-0x0200, seq=1024/4, ttl=128 (no response found)	
35 235.365868 206.1.1.2 206.1.3.2 ICMP 94 echo (ping) request 1d-0x0200, seq=1024/4, ttl=127 (no response found) [ETHERNET FRAME CHECK SEQUENCE INCORRECT]	
36 235.366391 206.1.1.2 206.1.3.2 ICMP 94 echo (ping) request 1d-0x0200, seq=1024/4, ttl=126 (reply in 37) [ETHERNET FRAME CHECK SEQUENCE INCORRECT]	
37 235.366393 206.1.3.2 206.1.1.2 ICMP 74 echo (ping) reply 1d-0x0200, seq=1024/4, ttl=128 (request in 36)	
38 235.366924 206.1.3.2 206.1.1.2 ICMP 74 echo (ping) reply 1d-0x0200, seq=1024/4, ttl=127	
39 235.489667 206.1.3.2 206.1.1.2 ICMP 94 echo (ping) reply 1d-0x0200, seq=1024/4, ttl=126 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]	

图 11: wireshark 捕获数据包

七、 ICMP 报文发送

对于 TTL 为 0 的数据包、校验和错误的数据包，路由器都要回送 ICMP 数据包，在此给出 ICMPPacket 函数，用于返回 ICMP 报文，这样路由器就可以支持 tracer 指令。

ICMPacket 函数

```

1 void ICMPPacket(BYTE type, BYTE code, const u_char* pkt_data) {
2     ICMP_t packet;
3     //设置FrameHeader_t
4     memcpy(packet.FrameHeader.DesMAC, ((FrameHeader_t*)pkt_data)->SrcMAC,
5            6);
6     memcpy(packet.FrameHeader.SrcMAC, ((FrameHeader_t*)pkt_data)->DesMAC,
7            6);
8     packet.FrameHeader.FrameType = htons(0x0800);
9     //设置IPHeader_t
10    packet.IPHeader.Ver_HLen = ((IPHeader_t*)(pkt_data + 14))->Ver_HLen;

```

```
9      packet.IPHeader.TOS = ((IPHeader_t*)(pkt_data + 14))->TOS;
10     packet.IPHeader.TotalLen = htons(56);
11     packet.IPHeader.ID = ((IPHeader_t*)(pkt_data + 14))->ID;
12     packet.IPHeader.Flag_Segment = ((IPHeader_t*)(pkt_data + 14))->
        Flag_Segment;
13     packet.IPHeader.TTL = 64;
14     packet.IPHeader.Protocol = 1;
15     packet.IPHeader.SrcIP = ip[0];
16     packet.IPHeader.DstIP = ((IPHeader_t*)(pkt_data + 14))->SrcIP;
17     setChecksum((IP_t*)&packet);
18     //设置ICMPHeader_t
19     packet.ICMPHeader.Type = type;
20     packet.ICMPHeader.Code = code;
21     packet.ICMPHeader.Id = 0;
22     packet.ICMPHeader.Sequence = 0;
23     packet.ICMPHeader.Checksum = 0;
24     memcpy(((u_char*)&packet) + 42, (IPHeader_t*)(pkt_data + 14), 20);
25     memcpy(((u_char*)&packet) + 62, (u_char*)(pkt_data + 34), 8);
26     unsigned int sum = 0;
27     WORD* tmp = (WORD*)&(packet.ICMPHeader);
28     for (int i = 0; i < 5; i++) {
29         sum += tmp[i];
30         if (sum & 0xffff0000) {
31             sum &= 0xffff;
32             sum++;
33         }
34     }
35     packet.ICMPHeader.Checksum = ~sum;
36     cout << ~sum << endl;
37     pcap_sendpacket(adhandle, (u_char*)&packet, 70);
38 }
```

八、 附录

(一) 实验中使用的函数

```

bool compare(BYTE mac1[], BYTE mac2[]);           //比较mac地址
bool CheckSum(IP_t* ip_packet);                   //检验checksum
void setCheckSum(IP_t* ip_packet);                 //设置checksum
void printMac(BYTE mac[]);                         //打印mac地址
void printIp(DWORD ip);                           //打印ip
void printAlldevs();                               //打印设备信息
void printIf(pcap_if_t* d);                       //打印网卡信息
pcap_t* open(int choose);                         //打开网卡设备, 获取IP、掩码、
DWORD getNet(DWORD ip, DWORD mask);               //获取网络号
void getSelfMac(pcap_t* adhandle, DWORD ip, BYTE* mac); //欺骗获取网卡MAC
void getOtherMac(pcap_t* adhandle, DWORD ip_, BYTE* mac_); //获取IP对应MAC
void sendARP(pcap_t* adhandle, DWORD ip_);        //发送ARP数据包
void resend(pcap_t* adhandle, ICMP_t data, DWORD nextip, BYTE dstMac[]); //转发数据包
DWORD WINAPI handlerRequest(LPVOID lparam);       //路由转发线程
int find(Buffer* buffer, int size);               //查找缓冲区中空余位置
void handle(Buffer* buffer, int size);             //处理缓冲区
void cache(Buffer* buffer, int size, ICMP_t* data, DWORD TargetIP); //缓存数据包
void ICMPPacket(BYTE type, BYTE code, const u_char* pkt_data); //发送ICMP报文

```

图 12: 函数声明

(二) 数据报结构定义

数据包结构定义

```

1  typedef unsigned char BYTE;
2  typedef unsigned short WORD;
3  typedef unsigned long DWORD;
4
5  #pragma pack(1)
6  struct FrameHeader_t {
7
8      BYTE DesMAC[6];           //目的地址
9      BYTE SrcMAC[6];          //源地址
10     WORD FrameType;           //帧类型
11 };
12
13 struct ARPHeader_t {
14     WORD HardwareType;        //硬件类型
15     WORD ProtocolType;       //协议类型
16     BYTE HLen;                //硬件地址长度
17     BYTE PLen;                //协议地址
18     WORD Operation;           //操作
19     BYTE SendHa[6];           //发送方MAC
20     DWORD SendIP;             //发送方IP
21     BYTE RecvHa[6];           //接收方MAC
22     DWORD RecvIP;             //接收方IP
23 };

```

```
24 struct IPHeader_t {
25     BYTE Ver_HLen;
26     BYTE TOS;
27     WORD TotalLen;
28     WORD ID;
29     WORD Flag_Segment;
30     BYTE TTL; // 生命周期
31     BYTE Protocol; // 协议
32     WORD Checksum; // 校验和
33     DWORD SrcIP; // 源IP
34     DWORD DstIP; // 目的IP
35 };
36
37 struct ICMPHeader_t {
38     WORD Type; // 类型
39     WORD Code; // 代码
40     WORD Checksum; // 校验和
41     WORD Id; // 标识符
42     WORD Sequence; // 序号
43 };
44
45 struct ARP_t {
46     FrameHeader_t FrameHeader; // 帧首部
47     ARPHeader_t ARPHeader; // ARP首部
48 };
49
50 struct IP_t {
51     FrameHeader_t FrameHeader; // 帧首部
52     IPHeader_t IPHeader; // IP首部
53 };
54
55 struct ICMP_t {
56     FrameHeader_t FrameHeader;
57     IPHeader_t IPHeader;
58     ICMPHeader_t ICMPHeader;
59     char buf[50];
60 };
61 #pragma pack()
```