南開大學

网络技术与应用课程实验报告

实验五：简单路由器程序的设计

专　　业　信息安全

学　　号　2113662

姓　　名　张丛

班　　级　信息安全一班

一、 实验目的

(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、实验过程

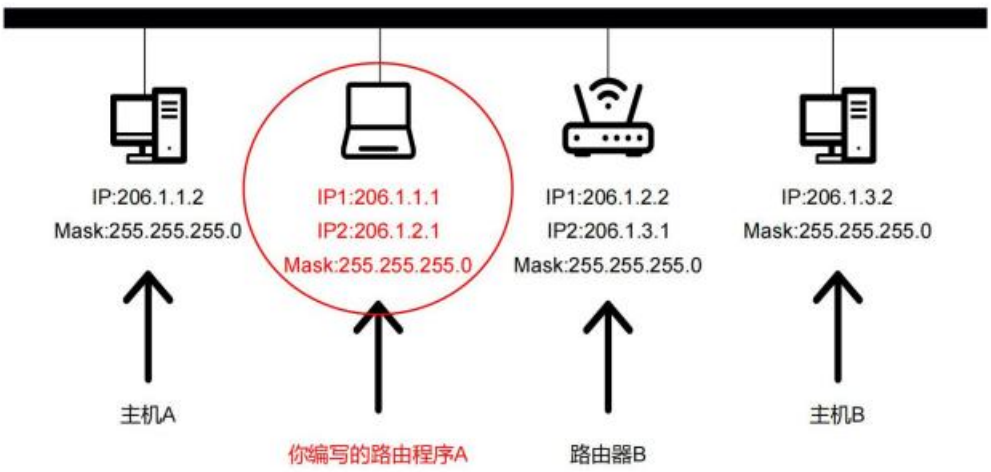
在实验 4 中，我们曾在实际环境中完成了按静态路由方式配置了路由器和主机，然后进行连通测试。

此次实验，就是我们自己编写路由程序，在四台虚拟机中再次完成静态路由配置，使两台主机可以连通。

虚拟环境：

编号	IP	NetMask	说明
1	206.1.1.2	255.255.255.0	终端设备
2	206.1.1.1, 206.1.2.1	255.255.255.0	路由器
3	206.1.2.2, 206.1.3.1	255.255.255.0	路由器
4	206.1.3.2	255.255.255.0	终端设备

网络拓扑：



实验的重难点：编写路由程序。

实现路由程序的**思路流程**：

（一）准备工作

- 捕获设备，打开网卡，获取双 IP
- 伪造 ARP 报文，获取本机 MAC
- 自动添加默认路由表项，手动添加（或删除）路由表项

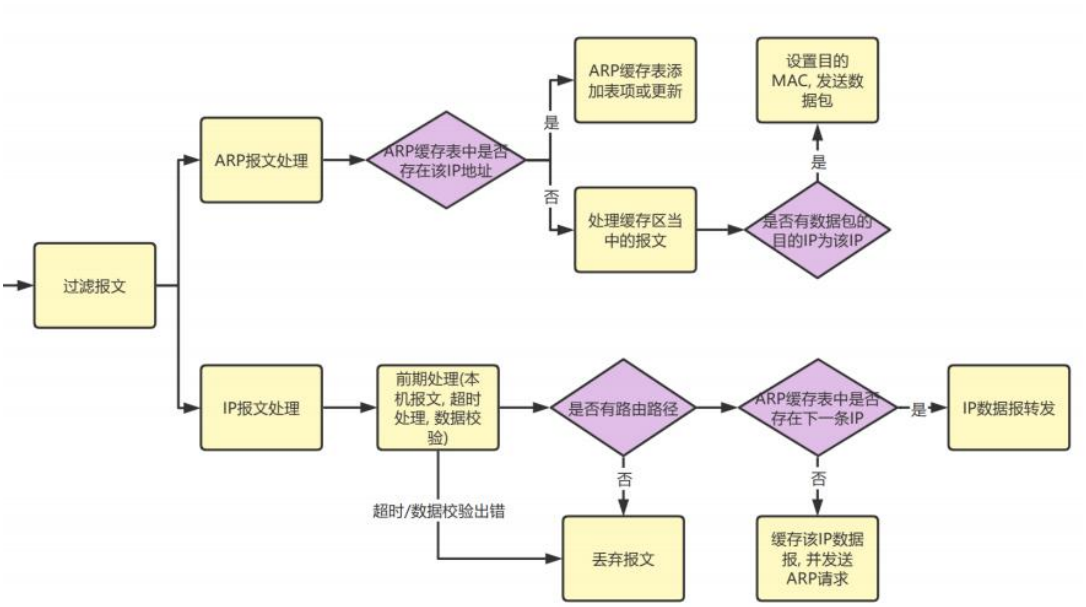
（二）接收消息并处理

- 捕获报文
- 捕获 IP 报文的处理
- 捕获 ARP 报文的处理

（三）转发

- MAC 地址的修改
- TTL 的修改
- 重新设置校验和

可以用下图表示第二三步:



重要的**数据结构**:

之前的实验使用过的 **ARP 报文和 IP 报文**:

```
//  ARP报文格式
typedef struct ARPFrame_t {
    FrameHeader_t FrameHeader;  //帧首部
    WORD HardwareType;          //硬件类型
    WORD ProtocolType;          //协议类型
    BYTE HLen;                  //硬件地址长度
    BYTE PLen;                  //协议地址
    WORD Operation;             //操作
    BYTE SendHa[6];             //发送方MAC
    DWORD SendIP;               //发送方IP
    BYTE RecvHa[6];             //接收方MAC
    DWORD RecvIP;               //接收方IP
}ARPFrame_t;
```

```
//  IP报文首部
typedef struct IPHeader_t {
    BYTE Ver_HLen;
    BYTE TOS;
    WORD TotalLen;
    WORD ID;
    WORD Flag_Segment;
    BYTE TTL;                   //生命周期
    BYTE Protocol;
    WORD Checksum;              //校验和
    ULONG SrcIP;                //源IP
    ULONG DstIP;                //目的IP
}IPHeader_t;
```

路由表：

```
// 路由表结构体 —— 链表存储路由表项
class Route_table {
public:
    Route_item* head, * tail;   //采用链表 —— 支持最多添加50转发表
    int num;                    //数量
    Route_table();              //初始化，添加直接连接的网络
    //路由表的添加 —— 1.直接投递在最前 2.其余按最长匹配原则
    void add(Route_item* a);
    //删除，type = 0不能删除
    void remove(int index);
    //路由表的打印 mask net next type
    void Print_file();
    //查找 —— 最长匹配原则返回下一跳的ip
    DWORD lookup(DWORD ip);
};
```

路由表是以路由表项为节点的链表，定义了三个成员函数：添加、删除、查找。

其中路由表项定义如下：

```
//  路由表表项
class Route_item {
public:
    DWORD mask;             //掩码
    DWORD net;              //目的网络
    DWORD nextip;           //下一跳
    BYTE nextMAC[6];        //下一跳的MAC地址
    int index;              //第几条
    int type;               //0为直接连接(不可删除)，1为用户添加
    Route_item* nextitem;
    Route_item() {
        memset(this, 0, sizeof(*this));
    }
    //  打印表项内容，打印出掩码、目的网络和下一跳IP、类型（是否是直接 投递）
    void Print_item();
};
```

除了必须有的掩码、目的、下一跳，还定义了 index 来标识是否是直接投递的默认路由（不可手动删除）。

ARP 表:

```
class Arp_table {
public:
    DWORD ip;           //IP地址
    BYTE mac[6];        //MAC地址
    static int num;     //表项数量
    static void insert(DWORD ip, BYTE mac[6]);  //插入表项
    static int lookup(DWORD ip, BYTE mac[6]);   //删除表项
}atable[50];
```

ARP 表是数组结构，ARP 表项由 ip 和对应的 mac 组成。

除了上面的数据结构，还定义了一些方便输出日志的结构体。

代码流程：

首先获取设备，打开网卡，**获取 ip**。

```
//  获取本机ip
find_alldevs();
//  输出此时存储的IP地址与MAC地址
for (int i = 0; i < 2; i++) {
    printf("%s\t", ip[i]);
    printf("%s\n", mask[i]);
}
```

```
'  获取对应网卡网卡号与IP地址
)id find_alldevs() {
  if (pcap_findalldevs_ex(pcap_src_if_string, NULL, &alldevs, errbuf) == -1) {
      printf("%s", "error");
  }
  else {
      int i = 0;
      //  获取该网络接口设备的ip地址信息
      for (d = alldevs; d != NULL; d = d->next) {
          if (i == index) {
              net[i] = d;                                          全局数组net的相应位置
              int t = 0;
              for (a = d->addre
                  if (((struct sockaddr_in*)a->addr)->sin_family == AF_INET && a->addr)   //确保是IPv4地址
                  {
                      printf("%d ", i);
                      printf("%s\t", d->name, d->description);
                      printf("%s\t%s\n", "IP地址:", inet_ntoa(((struct sockaddr_in*)a->addr)->sin_addr));
                      //  存储对应IP地址与MAC地址
                      strcpy(ip[t], inet_ntoa(((struct sockaddr_in*)a->addr)->sin_addr));          //将IP地
                      strcpy(mask[t], inet_ntoa(((struct sockaddr_in*)a->netmask)->sin_addr));     //将子网
                      t++;
                  }
              }
              //  打开该网卡
              ahandle = open(d->name);
          }
          i++;
      }
  }
  pcap_freealldevs(alldevs);
```

struct sockaddr_in
Microsoft Windows Extended data types

然后**获取本机 MAC**,通过伪造 ARP 报文来实现:

```
//  获取本机MAC
Get_SelfMac(inet_addr(ip[0]));
//  打印本机MAC
Print_Mac(selfmac);
```

```
//  获得本地IP地址以及对应的MAC地址
void Get_SelfMac(DWORD ip) {
    memset(selfmac, 0, sizeof(selfmac));
    ARPFrame_t ARPFrame;
    for (int i = 0; i < 6; i++) {
        //  将APRFrame.FrameHeader.DesMAC设置为广播地址
        ARPFrame.FrameHeader.DesMAC[i] = 0xff;
        //  将APRFrame.FrameHeader.SrcMAC设置为本机网卡的MAC地址
        ARPFrame.FrameHeader.SrcMAC[i] = 0x0f;
        //  将ARPFrame.SendHa设置为本机网卡的MAC地址
        ARPFrame.SendHa[i] = 0x0f;
        //  将ARPFrame.RecvHa设置为0
        ARPFrame.RecvHa[i] = 0;
    }
    ARPFrame.FrameHeader.FrameType = htons(0x806);  //帧类型为ARP
    ARPFrame.HardwareType = htons(0x0001);          //硬件类型为以太网
    ARPFrame.ProtocolType = htons(0x0800);          //协议类型为IP
    ARPFrame.HLen = 6;                              //硬件地址长度为6
    ARPFrame.PLen = 4;                              //协议地址长为4
    ARPFrame.Operation = htons(0x0001);             //操作为ARP请求
    //  将ARPFrame.SendIP设置为本机网卡上绑定的IP地址
    ARPFrame.SendIP = inet_addr("122.122.122.122");
    ARPFrame.RecvIP = ip;
    u_char* h = (u_char*)&ARPFrame;
```

在 ARP 报文的响应中获取 MAC:

```
if (pcap_sendpacket(ahandle, (u_char*)&ARPFrame, sizeof(ARPFrame_t)) != 0) {
    //  发送错误处理
    printf("senderror\n");
}
else {
    //  发送成功
    while (1) {
        pcap_pkthdr* pkt_header;
        const u_char* pkt_data;
        int rtn = pcap_next_ex(ahand                    kt_data);
        if (rtn == 1) {
            ARPFrame_t* IPPacket = (ARPFrame_t*)pkt_data;
            //  输出目的MAC地址
            if (ntohs(IPPacket->FrameHeader.FrameType) == 0x806) {
                if (!Compare_MAC(IPPacket->FrameHeader.SrcMAC, ARPFrame.FrameHeader.SrcMAC) && Comp
                    ltable.write2log_arp(IPPacket);
                    //  输出源MAC地址，源MAC地址即为所需MAC地址
                    for (int i = 0; i < 6; i++)
                        selfmac[i] = IPPacket->FrameHeader.SrcMAC[i];
                    //  已经捕获到了MAC地址，因此退出
                    break;
```

struct ARPFrame_t
ARP报文格式

接下来**初始化路由表**，在前面我们已经得到了需要的 ip:

```
BYTE mac[6];
int Choice;
//  初始化路由表结构体
Route_table Router;
//  初始化路由表项
Route_item Item;
```

```
//  初始化路由表，添加默认路由
Route_table::Route_table() {
    head = new Route_item;
    tail = new Route_item;
    head->nextitem = tail;
    num = 0;
    for (int i = 0; i < 2; i++) {
        Route_item* temp = new Route_item;
        //  本机网卡的ip 和掩码进行按位与即为所在网络
        temp->net = (inet_addr(ip[i])) & (inet_addr(mask[i]));
        temp->mask = inet_addr(mask[i]);
        temp->type = 0;     //0表示直接投递的网络，不可删除
        this->add(temp);    //添加表项
    }
```

接下来我们设立多线程，在主函数里面我们可以对路由表进行添加和删除的操作，以及随机打印输出路由表：

```
while (1) {
    printf("==================== 1. 添加路由表项 ====================\n");
    printf("==================== 2. 删除路由表项 ====================\n");
    printf("==================== 3.  打印路由表  ====================\n");
    printf("==================== 4.  退出程序  ====================\n");
    printf("\n");
```

设立接收并转发数据包的线程：

```
hThread = CreateThread(NULL, NULL, handlerRequest, LPVOID(&Router), 0, &dwThreadId);
```

在这个线程里，我们将完成路由器的主要功能：

首先，捕获数据包：

```
//  接收和处理线程函数
DWORD WINAPI handlerRequest(LPVOID lparam) {
    Route_table rtable = *(Route_table*)(LPVOID)lparam;
    while (1) {
        pcap_pkthdr* pkt_header;
        const u_char* pkt_data;
        //  通过pcap_next_ex()函数对本机网卡接收到的数据包进行循环捕获
        while (1) {
            int rtn = pcap_next_ex(ahandle, &pkt_header, &pkt_data);
            //  接收到消息 —— 跳出
            if (rtn)
                break;
        }
```

对捕获的报文进行判断：

如果捕获报文的目的 MAC 不是本机 MAC，直接丢弃；

如果在路由表中查找不到对应的路由表项，也直接丢弃。

```
FrameHeader_t* header = (FrameHeader_t*)pkt_data;
//  判断捕获报文的目的MAC是本机MAC
if (Compare_MAC(header->DesMAC, selfmac)) {
    //  收到IP格式数据报
    if (ntohs(header->FrameType) == 0x800) {
        Data_t* data = (Data_t*)pkt_data;
        //  写入日志
        ltable.write2log_ip("[receive IP]", data);
        DWORD ip1_ = data->IPHeader.DstIP;  // 将ip1设置为目的Ip头首部

        //  判断是否能查找到下一跳ip ip1
        DWORD ip_ = rtable.lookup(ip1_);    //查找是否有对应表项
        if (ip_ == -1)                      //如果没有找到下一跳IP地址则丢弃
            continue;
```

其中，查找路由表的函数如下：

```
//  查找路由表对应表项 —— 并给出下一跳的ip地址
DWORD Route_table::lookup(DWORD ip) {
    Route_item* t = head->nextitem;
    for (; t != tail; t = t->nextitem) {
        //  目的IP和掩码 确定目的网络 —— 再返回下一跳
        if ((t->mask & ip) == t->net)
            return t->nextip;
    }
    return -1;
}
```

因为在添加路由表项时，是按照最长匹配原则，按照掩码长度从大到小的顺序排列：

```
//  添加路由表项
void Route_table::add(Route_item* item) {
    Route_item* pointer;
    // 默认路由添加在路由表链表头部
    if (item->type == 0) {
        item->nextitem = head->nextitem;
        head->nextitem = item;
        item->type = 0;
    }
    //其它，按照最长匹配原则，按照掩码长度从大到小的顺序排列
    else {
        for (pointer = head->nextitem; pointer != tail && pointer->nextitem != tail; pointer = pointer->next
            if (item->mask < pointer->mask && item->mask >= pointer->nextitem->mask || pointer->nextitem ==
                break;
        }
        //插入到合适位置
        item->nextitem = pointer->nextitem;
        pointer->nextitem = item;
        //a->type = 1;
```

所以在查找路由表项时也符合最长匹配原则。

接下来，在知道下一跳的情况下，且校验和无误时，（若不知道下一跳的 MAC）需要查找 ARP 表。

```
//  判断校验和 — 校验和不正确，则直接丢弃不进行处理
if (Check_checksum(data)) {
    if (data->IPHeader.DstIP != inet_addr(ip[0]) && data->IPHeader.DstIP != inet_addr(ip[1])
        int t1 = Compare_MAC(data->FrameHeader.DesMAC, broadcast);
        int t2 = Compare_MAC(data->FrameHeader.SrcMAC, broadcast);
        if (!t1 && !t2) {
            //  ICMP报文包含IP数据包报头和其它内容
            ICMP_t* temp_ = (ICMP_t*)pkt_data;
            ICMP_t temp = *temp_;
            BYTE mac[6];

            //  如果查到下一跳IP地址
            //直接投递
            if (ip_ == 0) {
                //如果ARP表中没有所需内容，则需要获取ARP
                if (!Arp_table::lookup(ip1_, mac))
                    Arp_table::insert(ip1_, mac);
                //printMac(mac);
                resend(temp, mac);
            }
            //非直接投递
            else if (ip_ != -1) {
                //  没在表中查找到 — 添加并转发
                if (!Arp_table::lookup(ip_, mac))
                    Arp_table::insert(ip_, mac);
                resend(temp, mac);
```

获取下一跳 MAC 与获取本机 MAC 的过程类似，都需要发送 ARP 报文，根据 ARP 的响应来获得 MAC。（且之前的 ARP 的实验中也做过，不赘述了）

当所有工作都处理好之后，进行数据报转发：

```
//  数据报转发
void resend(ICMP_t data, BYTE dmac[]) {
    Data_t* temp = (Data_t*)&data;
    //  修改MAC地址
    memcpy(temp->FrameHeader.SrcMAC, temp->FrameHeader.DesMAC, 6);  //源MAC为本机MAC
    memcpy(temp->FrameHeader.DesMAC, dmac, 6);                      //目的MAC为下一跳MAC
    //  修改TTL值
    temp->IPHeader.TTL -= 1;            //TTL-1
        //  超时则丢弃
    if (temp->IPHeader.TTL < 0)
        return;//丢弃
    setchecksum(temp);                                              //重新设置校验和
    int rtn = pcap_sendpacket(ahandle, (const u_char*)temp, sizeof(temp));  //发送数据报
    if (rtn == 0)
        ltable.write2log_ip("[forward IP]", temp);//写入日志
}
```

于是完成了路由功能。

三、实验结果

先在虚拟机 3 中手动添加路由：

```
C:\Documents and Settings\Administrator>route add 206.1.1.0 MASK 255.255.255.0 2
06.1.2.1

C:\Documents and Settings\Administrator>route PRINT

IPv4 Route Table
===========================================================================
Interface List
0x1 ........................... MS TCP Loopback interface
0x10003 ...00 0c 29 d9 70 cd ...... Intel(R) PRO/1000 MT Network Connection
===========================================================================
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
        127.0.0.0        255.0.0.0        127.0.0.1        127.0.0.1      1
        206.1.1.0    255.255.255.0        206.1.2.1        206.1.2.2      1
        206.1.2.0    255.255.255.0        206.1.2.2        206.1.2.2     10
        206.1.2.2  255.255.255.255        127.0.0.1        127.0.0.1     10
      206.1.2.255  255.255.255.255        206.1.2.2        206.1.2.2     10
        206.1.3.0    255.255.255.0        206.1.3.1        206.1.2.2     10
        206.1.3.1  255.255.255.255        127.0.0.1        127.0.0.1     10
      206.1.3.255  255.255.255.255        206.1.3.1        206.1.2.2     10
        224.0.0.0        240.0.0.0        206.1.2.2        206.1.2.2     10
  255.255.255.255  255.255.255.255        206.1.2.2        206.1.2.2      1
===========================================================================
Persistent Routes:
  None
```

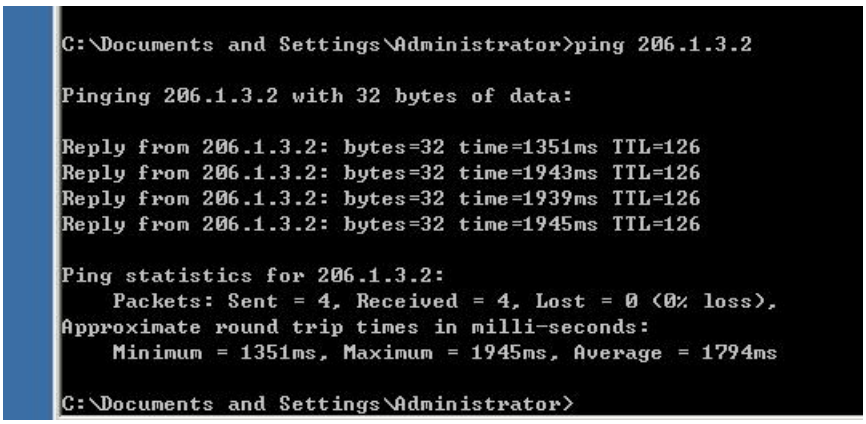
在虚拟机 2 中运行我们的路由程序:

```
C:\Documents and Settings\Administrator\桌面\路由器.exe
0 rpcap://\Device\NPF_{3AC00148-1BFF-47AF-8441-F1D14A619031}    IP地址: 206.1.2.
1
0 rpcap://\Device\NPF_{3AC00148-1BFF-47AF-8441-F1D14A619031}    IP地址: 206.1.1.
1
206.1.2.1       255.255.255.0
206.1.1.1       255.255.255.0
MAC地址为:  00-0C-29-3C-77-FD
```

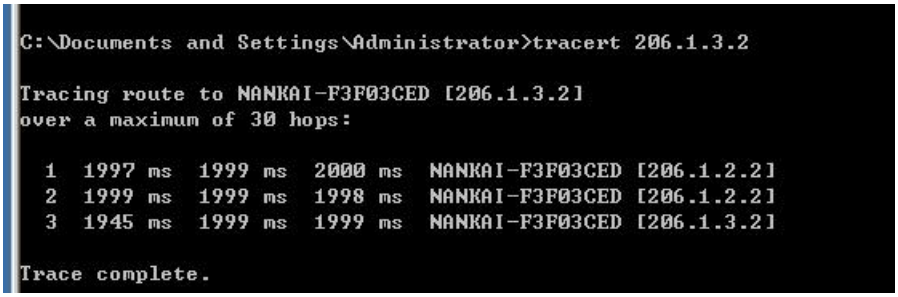
在路由程序中添加路由:

```
请输入操作序号 : 1

请输入掩码: 255.255.255.0
请输入目的网络: 206.1.3.0
请输入下一跳地址: 206.1.2.2

==================== 1. 添加路由表项 ====================
==================== 2. 删除路由表项 ====================
==================== 3.  打印路由表  ====================
==================== 4.  退出程序  ====================

请输入操作序号 : 3

     掩码             目的IP           下一跳
255.255.255.0       206.1.1.0      0.0.0.0
255.255.255.0       206.1.2.0      0.0.0.0
255.255.255.0       206.1.3.0      206.1.2.2
```

在虚拟机 1 中 ping 虚拟机 4:



虚拟机 1 tracert 虚拟机 4:



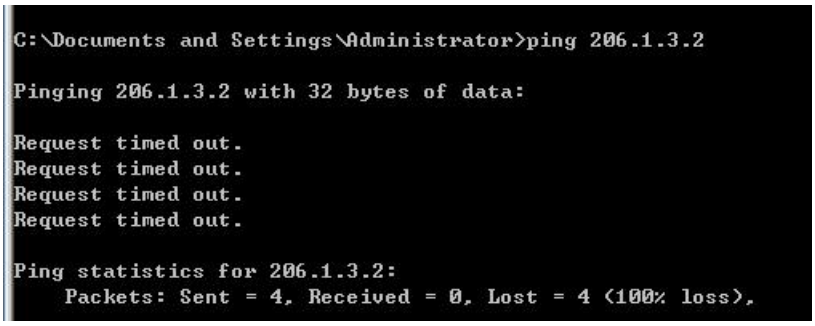
查看日志:

```
Log_File.txt - 记事本
文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)
[ ARP ] DesIP:  206.1.2.1  DesMAC:  00-0C-29-AA-6A-82
[receive IP]  SrcIP:  206.1.1.2  DesIP:  206.1.3.2  SrcMAC:  00-0C-29-E1-12-76  DesMAC:  00-0C-29-AA-6A-82
[ ARP ] DesIP:  206.1.2.2  DesMAC:  00-0C-29-B0-56-24
[forward IP]  SrcIP:  206.1.1.2  DesIP:  206.1.3.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-B0-56-24
[receive IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-B0-56-24  DesMAC:  00-0C-29-AA-6A-82S
[ ARP ] DesIP:  206.1.1.2  DesMAC:  00-0C-29-E1-12-76
[forward IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-E1-12-76
[receive IP]  SrcIP:  206.1.1.2  DesIP:  206.1.3.2  SrcMAC:  00-0C-29-E1-12-76  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.1.2  DesIP:  206.1.3.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-B0-56-24
[receive IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-B0-56-24  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-E1-12-76
[receive IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-B0-56-24  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-E1-12-76
[receive IP]  SrcIP:  206.1.1.2  DesIP:  206.1.3.2  SrcMAC:  00-0C-29-E1-12-76  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.1.2  DesIP:  206.1.3.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-B0-56-24
[receive IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-B0-56-24  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-E1-12-76
[receive IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-B0-56-24  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-E1-12-76
[receive IP]  SrcIP:  206.1.1.2  DesIP:  206.1.3.2  SrcMAC:  00-0C-29-E1-12-76  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.1.2  DesIP:  206.1.3.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-B0-56-24
[receive IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-B0-56-24  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-E1-12-76
[receive IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-B0-56-24  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-E1-12-76
[ ARP ] DesIP:  206.1.2.1  DesMAC:  00-0C-29-AA-6A-82
[receive IP]  SrcIP:  206.1.1.2  DesIP:  206.1.3.2  SrcMAC:  00-0C-29-E1-12-76  DesMAC:  00-0C-29-AA-6A-82
[ ARP ] DesIP:  206.1.2.2  DesMAC:  00-0C-29-B0-56-24
[forward IP]  SrcIP:  206.1.1.2  DesIP:  206.1.3.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-B0-56-24
[receive IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-B0-56-24  DesMAC:  00-0C-29-AA-6A-82
[ ARP ] DesIP:  206.1.1.2  DesMAC:  00-0C-29-E1-12-76
[forward IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-E1-12-76
[receive IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-B0-56-24  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-E1-12-76
[receive IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-B0-56-24  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-E1-12-76
[receive IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-B0-56-24  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-E1-12-76
[receive IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-B0-56-24  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-E1-12-76
[receive IP]  SrcIP:  206.1.1.2  DesIP:  206.1.3.2  SrcMAC:  00-0C-29-E1-12-76  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.1.2  DesIP:  206.1.3.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-B0-56-24
[receive IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-B0-56-24  DesMAC:  00-0C-29-AA-6A-82
[forward IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-AA-6A-82  DesMAC:  00-0C-29-E1-12-76
[receive IP]  SrcIP:  206.1.3.2  DesIP:  206.1.1.2  SrcMAC:  00-0C-29-B0-56-24  DesMAC:  00-0C-29-AA-6A-82
```

在路由程序中删除路由表项:

```
请输入操作序号 : 2

请输入删除表项编号:
2

==================== 1. 添加路由表项 ====================
==================== 2. 删除路由表项 ====================
==================== 3.  打印路由表  ====================
==================== 4.  退出程序  ====================

请输入操作序号 : 3

     掩码              目的IP           下一跳
255.255.255.0       206.1.1.0       0.0.0.0
255.255.255.0       206.1.2.0       0.0.0.0
```

在虚拟机 1 ping 虚拟机 4:



可见，删除路由表项后无法连通了。

四、总结与思考

此次实验，的确是最难的，其实路由程序的思路流程是比较容易想清楚的，但代码的工作量比较大。

有些地方的代码不是最重要的但很繁杂，于是我没有截图进报告里。

github 链接： nku_network_technique/实验五 简单路由 at main • zciszrry/nku_network_technique (github.com)