实验5：简单路由器程序的设计

学号：2110049 姓名：张刘明

实验要求：

简单路由器程序设计实验的具体要求为：

（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。

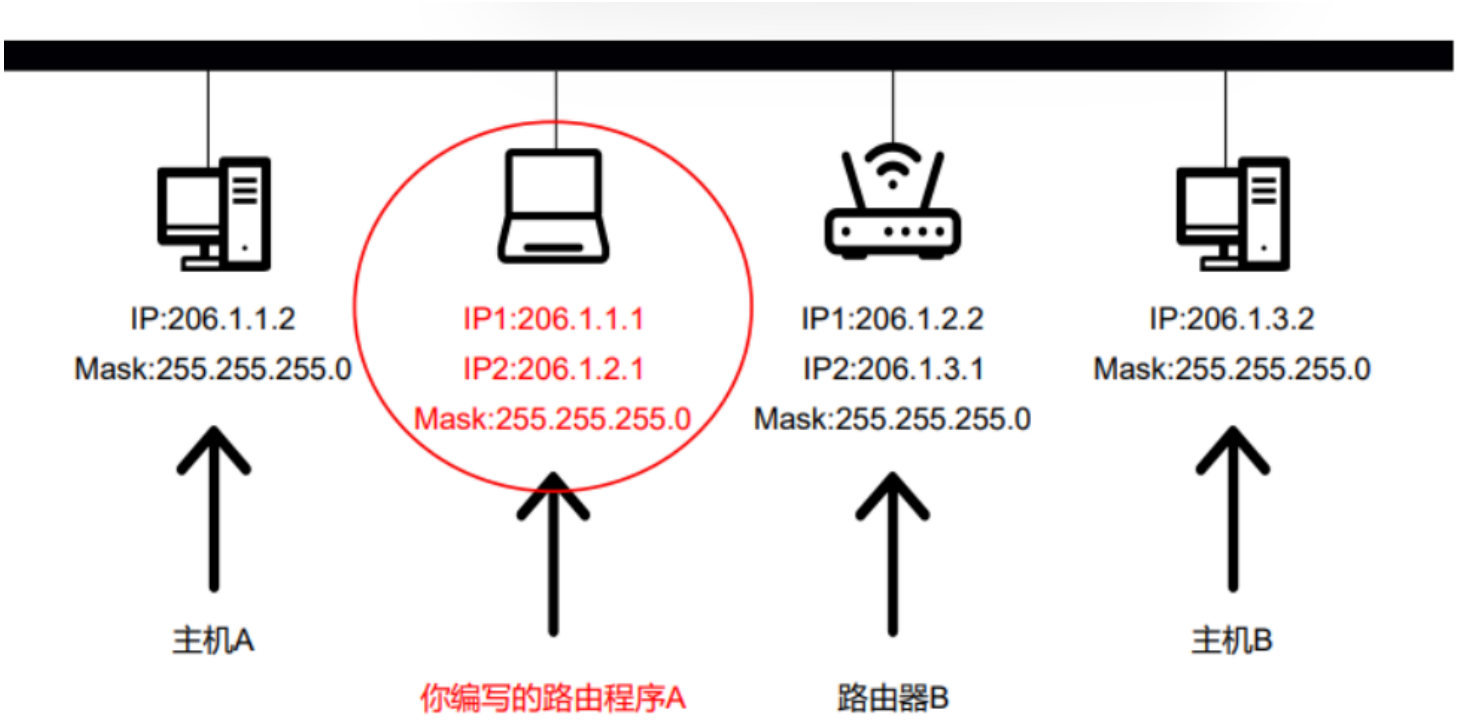
（3）需要给出路由表的手工插入、删除方法。

（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。

实验过程

实验环境



其中，主机A和主机B为终端设备，路由器A为需要运行路由程序的设备，路由器B为另一台路由器。所有设备的IP地址已经全部分配好，所有设备已经安装好x86的VC++运行时环境。其中路由器B的路由功能已经全部开启，但仍需要每次开机后手动添加路由表

项 `route ADD 206.1.1.0 MASK 255.255.255.0 206.1.2.1` 。

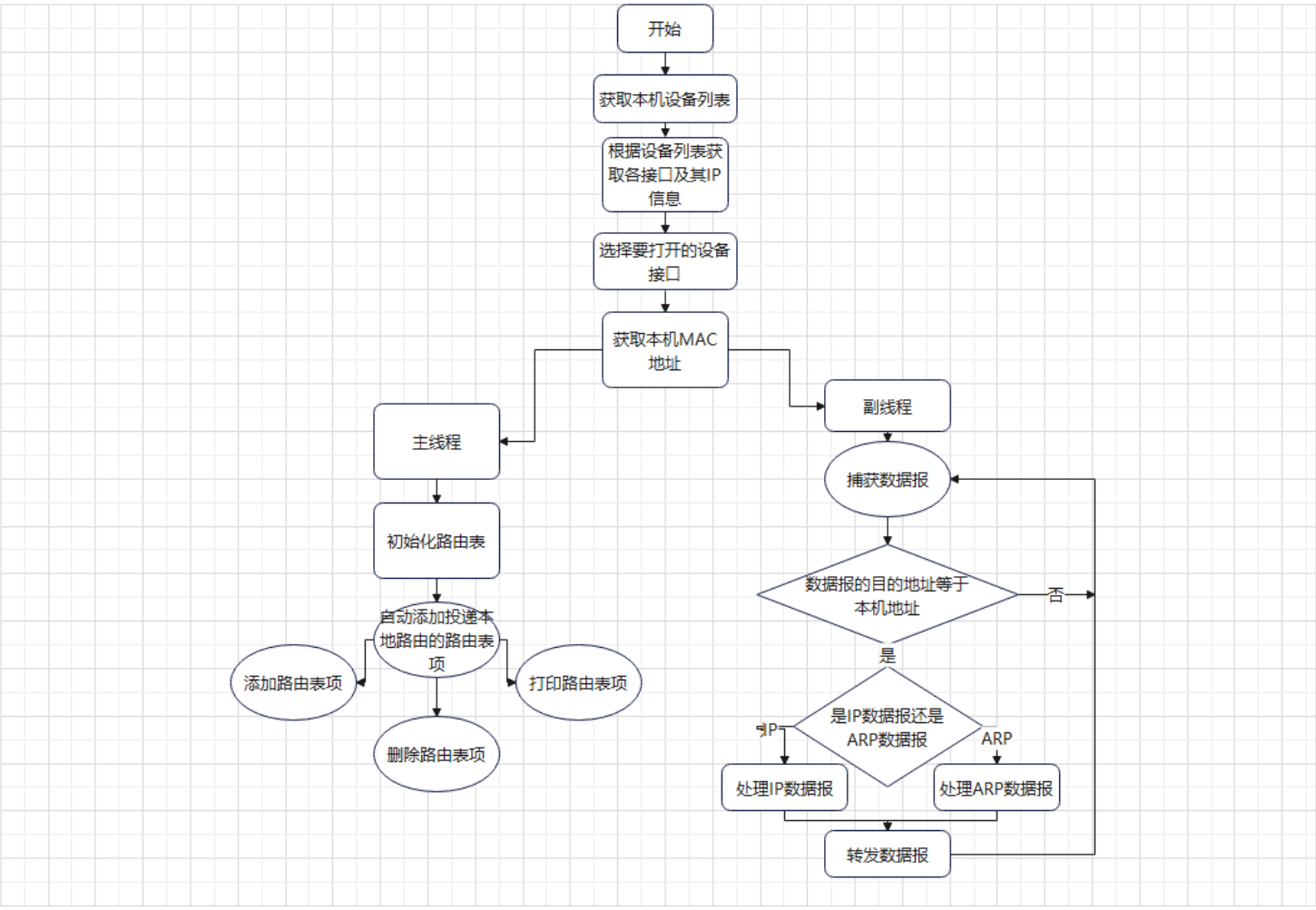
路由器要完成的工作

在本次实验环境中，要实现对接收的数据报进行分析和转发，实现处于不同网络中的主机A和主机B的相互通信（通信通过ping命令执行），路由器需要具备以下功能：

- 为经过的IP数据报选择路由，对接受的IP数据报提取目的IP地址，根据自己的路由表信息为该数据报选择最优的转发路径；
  - 关于路由选择算法，采用表驱动的路由选择算法。需要进行路由选择的设备需要维护一张IP路由表，路由表存储有关目的地址以及怎样到达目的地址的信息，该设备通过查询路由表决定把数据报发往哪里；
- 处理IP数据报的TTL域中的数值，抛弃TTL<=0的数据报，将转发的IP数据报TTL减1；
- 重新计算IP数据报的头部校验和；

- 生成和处理ICMP报头，相关的ICMP报文涉及两种：
  - **目的不可达报文**：当路由器不能为相应的数据包找到路由器或者主机交付数据包时，就丢弃该数据报，向源主机发送ICMP目的不可达报文
  - **超时报文**：路由器在转发数据包时，如果生存周期TTL值减一后为0，就丢弃这个数据包。当丢弃这个数据包的时候，路由器向源主机发出ICMP超时报文
- 实现路由协议，维护静态路由；
- 实现ARP协议，形成数据帧，在将一个IP数据报送往下一跳步之前，路由程序需要获取到下一跳的物理地址，然后形成数据帧从选择的网络接口发送出去。

路由器工作流程



代码实现及功能分析

自定义数据结构

- 帧首部

```
typedef struct FrameHeader_t {
        BYTE DesMAC[6];//目的地址
        BYTE SrcMAC[6];//源地址
        WORD FrameType;//帧类型
}FrameHeader_t;
```

- IP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;
```

- ICMP首部

```
typedef struct ICMPHeader_t {
        BYTE    Type;
        BYTE    Code;
        WORD    Checksum;
        WORD    Id;
        WORD    Sequence;
}ICMPHeader_t;
```

- ARP数据帧

```
typedef struct ARPFrame_t {
        FrameHeader_t FrameHeader;//帧首部
        WORD HardwareType;//硬件类型
        WORD ProtocolType;//协议长度
        BYTE HLen;
        BYTE PLen;
        WORD Operation;
        BYTE SendHa[6];
        DWORD SendIP;
        BYTE RecvHa[6];
        DWORD RecvIP;
}ARPFrame_t;
```

- IP数据包

```
typedef struct IPFrame_t {//包含帧首部和IP首部的数据包
        FrameHeader_t FrameHeader;
        IPHeader_t IPHeader;
}IPFrame_t;
```

- 路由表项

```
class RouteEntry
{
public:
        RouteEntry* next;
        int number;//索引
        DWORD mask;//掩码
        DWORD dst_net;//目的网络
        DWORD next_hop;//下一跳的IP地址
        BYTE nextMAC[6];//下一跳的MAC地址
        int type;//0为直接连接（不可删除），1为用户添加
        RouteEntry() {
                next = NULL;
                number = 0;
                mask = 0;
                dst_net = 0;
                next_hop = 0;
                type = 0;
        }
        RouteEntry(int number, DWORD dstNetwork, DWORD mask, DWORD nextHop, int access)
        {
                this->number = number;
                this->dst_net = dstNetwork;
                this->mask = mask;
                this->next_hop = nextHop;
                this->type = access;
        }
        void print();
};
```

- 路由表

```
class RouteTable {
public:
        RouteEntry* head, * tail;//头节点
        int num;//条数
        RouteTable();//初始化，添加直接相连的网络
        void add(RouteEntry* entry);//添加路由表项，直接相连的在最前面，其余的按照最长匹配原则
        void erase(int number);//删除第i条路由表项，直接相连的不能删除
        void print();
        DWORD search(DWORD dstip);//根据最长匹配原则查找下一跳的ip地址

};
```

- ARP映射表

```
class ArpTable {
public:
        DWORD IP;
        BYTE MAC[6];
        static int num;
        static void add(DWORD IP, BYTE MAC[6]);
        static int search(DWORD IP, BYTE MAC[6]);
};
```

- 转发数据报

```
class PacketBuffer
{
public:
        BYTE                    pktData[MAX_SIZE];// 数据缓存
        int                             totalLen;// 数据包总长度
        ULONG                   targetIP;//目的IP地址
        bool                    valid = 1; //有效位：如果已经被转发或者超时，则置0
        clock_t                 Time;// 超时判断
        PacketBuffer() {};
        PacketBuffer(const PacketBuffer& x)//复制构造函数--->检查！！！
        {
                memcpy(this->pktData, x.pktData, x.totalLen);
                this->totalLen = x.totalLen;
                this->targetIP = x.targetIP;
                this->valid = x.valid;
                this->Time = x.Time;
        }
};
```

- 日志类

```
class Log
{
public:
        Log();//打开文件进行写入
        ~Log();//关闭文件!
        static FILE* my_fp;
        //写入日志
        static void addInfo(const char* str/*日志信息标识*/);
        static void addInfohop(const char* str/*日志信息标识*/,DWORD hop);
        static void ARP_info(const char* str/*日志信息标识*/, ARPFrame_t* p);//arp类型
        static void IP_info(const char* str, IPFrame_t* p);//ip类型
        static void ICMP_info(const char* str);//icmp类型
};
```

路由表相关

- 路由表初始化
  路由表的初始化主要是实现了获取路由主机的双IP相关信息，并将它们添加到路由表当中，类型type 设置为0，表示为直接相连，不可删除

```
RouteTable::RouteTable()
{
        head = new RouteEntry;
        head->next = NULL;
        num = 0;
        //通过得到的双IP的掩码，在路由表中添加直接相连的网络，将类型设置为0，即不可删除项
        for (int i = 0; i < 2; i++)
        {
                RouteEntry* entry = new RouteEntry;
                //添加直接相连的网络
                entry->dst_net = (inet_addr(my_ip[i])) & (inet_addr(my_mask[i]));
                entry->mask = inet_addr(my_mask[i]);
                entry->type = 0;
                this->add(entry);
        }
}
```

- 添加路由表项并重新排序
  采取的是最长匹配原则，对路由表项的存储采用按照掩码排序，掩码越大存的越靠前。
  实现方法为：如果路由表为空，将表项 `RouteEntry* r` 直接添加到 `head` 后面，下一项设置为 `NULL` ;如果路由表非空，从路由表中第一项开始循环比较要插入的表项的掩码与表中已有的掩码，找到合适的位置（掩码小于前一项而大于后一项）插进去。插入后需要从表头遍历，对表中的路由表项重新编号，并且把路由表中的路由表项个数 `num + 1` 。

```
void RouteTable::add(RouteEntry*  r) {
        RouteEntry* temp;
        //当路由表是空的
        if (num == 0) {
                head->next = r;
                r->next = NULL;
        }
        else {
                temp = head->next;
                while (temp != NULL) {
                        //掩码大小小于temp而大于temp->next
                        if (temp->next == NULL || (r->mask < temp->mask && r->mask >= temp->next

                                break;
                        }
                        temp = temp->next;
                }
                if (temp->next == NULL)
                {

                        r->next = NULL;
                        temp->next = r;
                }
                else {
                        r->next = temp->next;
                        temp->next = r;
                }
        }
        RouteEntry* p = head->next;
        //重新编号
        for (int i = 0; p != NULL; i++)
        {
                p->number = i;
                p = p->next;
        }
        num++;
        my_log.addInfo("路由表添加成功!");
        return;
}
```

- 删除路由表项并重新排序
  输入索引为参数，首先先从头遍历，查找是否有对应的索引，如果没有的话，直接报错；如果索引

为第一项，则判断它的类型是否为直接相连，直接相连的路由表项无法删除，同样报错；如果不是的话，只有一项，直接让 head 的下一项为空，有多项的话，先存下第一项 temp ，之后令链表头的下一项直接为 temp 的下一项。如果是在中间位置，遍历到index处,寻找删除结点的前驱结点，存下要删除的点 rem ,让 rem 的前一项的下一项为rem的下一项，完成删除后重新编号。

```
void RouteTable::erase(int index) {
        RouteEntry* temp = new RouteEntry;
        bool hav = false;
        for (RouteEntry* t = head; t->next != NULL; t = t->next)
        {
                if (t->next->number == index)hav = true;
        }
        if (!hav) {
                printf("查无此项，请重新输入！\n");
                return;
        }
        if (index == 0)//删除第一项
        {
                temp = head->next;
                //默认路由表项不能删
                if (temp->type == 0)
                {
                        printf("没有权限删除该项！\n");
                        return;
                }
                else
                {
                        if (num == 1)
                        {
                                head->next = NULL;
                        }
                        else
                        {
                                temp = head->next;
                                head->next = temp->next;
                                printf("已成功删除指定项！\n");
                        }
                }
        }
        else
        {
                temp = head->next;
                for (int i = 0; i < index - 1; i++)//遍历到index处,寻找删除结点的前驱结点
                {
                        temp = temp->next;
                }
                RouteEntry* rem = new RouteEntry;
                rem = temp->next;//要删除的结点x
```

```
                if (rem->type == 0)
                {
                        printf("没有权限删除该项! \n");
                        return;
                }
                if (rem->next == NULL)//尾删
                {
                        temp->next = NULL;
                }
                //中间删
                temp->next = rem->next;
                printf("已成功删除指定项! \n");
        }

        RouteEntry* p = head->next;
        //重新为表项编号
        for (int i = 0; p != NULL; i++)
        {
                p->number = i;
                p = p->next;
        }
        num--;
        my_log.addInfo("路由表删除成功!");
        return ;
}
```

- 查询路由表中的下一跳

查找目的ip地址的下一跳的IP地址，遍历路由表，直到找到网络号匹配的一项（即目的IP地址和掩码相与得到的就是对应的网络号），因为路由表项的排列遵循最长匹配原则，所以找到的第一项就是。如果该项的类型是0，那么说明是直接相连的，直接投递即可，下一跳的地址即为查找的IP地址；如果类型为1，返回表中存储的下一条的地址即 next_hop 。如果没有查询到返回-1，据此，之后可以发送ICMP目的不可达数据报。

```
DWORD RouteTable::search(DWORD dst_ip) {
        DWORD a = -1;
        RouteEntry* t = head->next;
        for (; t!= NULL; t = t->next) {
                if ((t->mask & dst_ip) == t->dst_net) {
                        if (t->type == 0)//直接相连的网络
                        {
                                a= dst_ip;
                        }
                        else
                        a = t->next_hop;
                }
        }
        return a;
}
```

- 打印路由表项

```
void RouteEntry::print()
{
        //打印序号
        printf("%d   ", number);
        in_addr addr;
        //打印掩码
        addr.s_addr = mask;
        //转成主机序
        char* str = inet_ntoa(addr);
        printf("%s\t", str);
        //打印目的主机
        addr.s_addr = dst_net;
        str = inet_ntoa(addr);
        printf("%s\t", str);
        //打印下一跳ip地址
        addr.s_addr = next_hop;
        str = inet_ntoa(addr);
        printf("%s\t", str);
        //用户是否可操作
        if (type == 0)
                printf("non_accessable\n");
        else
                printf("accessable\n");
        printf("\n");
}
```

- 打印路由表

```
void RouteTable::print() {
        RouteEntry* temp = head->next;
        printf("-------------------------路由表----------------------\n");
        int t = 0;
        while (temp!=NULL) {
                temp->print();
                temp = temp->next;
                t++;
        }
        printf("共有路由表项：");
        printf("%d   ", num);
        printf("条/n");
}
```

日志相关

- 下一跳相关信息
  根据参数中的IP，输出下一条的信息，将DWORD类型通过 `inent_ntoa` 转成char*类型的主机序的IP地址便于观察。

```
void Log::addInfohop(const char* str/*日志信息标识*/, DWORD hop) {
        fprintf(my_fp, str);
        if (hop == -1)
                fprintf(my_fp, "%s  \n", "-1");
        in_addr addr;
        addr.s_addr = hop;
        char* str1 = inet_ntoa(addr);
        fprintf(my_fp, "%s  \n", str1);
}
```

- ARP数据报信息
  根据参数中的ARP数据帧类型，对于IP地址，将DWORD类型通过 `inent_ntoa` 转成char*类型的主机序的IP地址；对于MAC地址，通过遍历数组，并转换为16进制输出，不足两位，前面补0输出。

```
void Log::ARP_info(const char* str, ARPFrame_t* p) {
        fprintf(my_fp, str);
        fprintf(my_fp, "----------------------------ARP 数据包------------------------------\n
        in_addr addr;
        addr.s_addr = p->SendIP;
        char* str1 = inet_ntoa(addr);
        fprintf(my_fp, "源IP:  ");
        fprintf(my_fp, "%s  \n", str1);
        fprintf(my_fp, "源MAC:  ");
        for (int i = 0; i < 5; i++)
                fprintf(my_fp, "%02X-", p->SendHa[i]);
        fprintf(my_fp, "%02X\n", p->SendHa[5]);
        in_addr addr2;
        addr2.s_addr = p->RecvIP;
        char* str2 = inet_ntoa(addr2);
        fprintf(my_fp, "目的IP:  ");
        fprintf(my_fp, "%s  \n", str2);
        fprintf(my_fp, "目的MAC:  ");
        for (int i = 0; i < 5; i++)
                fprintf(my_fp, "%02X-", p->RecvHa[i]);
        fprintf(my_fp, "%02X\n", p->RecvHa[5]);
        fprintf(my_fp, "\n");
}
```

IP类型的数据报实现方法类似，这里不再赘述；对于ICMP报文只需输出相关提示信息，如是否超时、是否目的不可达等等。

主函数

- 获取本机网卡ip
  先通过函数 `pcap_findalldevs_ex` 获取网络接口的设备列表，然后通过指向链表首部的 `alldevs` 打印设备的相关信息，包括设备的名字和描述信息，并获得每个网络接口绑定的IP的相关信息，打印IP地址和网络掩码；接下来由用户指定要打开的网卡，打开后，将双IP地址存到 `my_ip` 数组中，把其对应的掩码存到 `my_mask` 中。

```
void  Get_Two_IP() {
        pcap_if_t* dev;//用于遍历网卡信息链表
        pcap_addr_t* add;//用于遍历IP地址信息链表：一个网卡可能有多个IP地址
        if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING,     //获取本机的接口设备
                NULL,                           //无需认证
                &alldevs,                       //指向设备列表首部
                errbuf
        ) == -1)//返回-1表示出错
        {
                printf("There's Error in pcap_findalldevs_ex! Program exit.\n");
                exit(1);
        }
        //显示设备列表信息
        int index = 0;
        for (dev = alldevs; dev; dev = dev->next) {
                printf("%d. %s", ++index, dev->name);
                if (dev->description)
                        printf("( %s )\n", dev->description);
                else
                        printf("(No description )\n");
                //获取该网络接口设备绑定的的IP地址信息
                for (add = dev->addresses; add != NULL; add = add->next) {
                        if (add->addr->sa_family == AF_INET) { //判断该地址是否IP地址
                                //输出相关的信息
                                char str[INET_ADDRSTRLEN];
                                //通过 inet_ntoa将一个网络字节序的IP地址转化为点分十进制的IP地址（字
                                strcpy(str, inet_ntoa(((struct sockaddr_in*)add->addr)->sin_addr
                                printf("IP地址：%s\n", str);
                                strcpy(str, inet_ntoa(((struct sockaddr_in*)add->netmask)->sin_a
                                printf("网络掩码：%s\n", str);
                        }
                }
        }
        if (index == 0)
        {
                printf("\nNo interfaces found!\n");
        }
        printf("-------------------------------------------------------------------------------
        dev = alldevs;
        int num;
        printf("请选择您要打开的网卡：");
        scanf("%d", &num);
        //遍历寻找要打开的网络
```

```
        for (int i = 0; i < num - 1; i++) {
                dev = dev->next;
        }
        //把对应的ip和掩码存上
        int t = 0;
        for (add = dev->addresses; add != NULL && t < 10; add = add->next) {
                if (add->addr->sa_family == AF_INET) {//是IP类型的
                        strcpy(my_ip[t], inet_ntoa(((struct sockaddr_in*)add->addr)->sin_addr));
                        strcpy(my_mask[t], inet_ntoa(((struct sockaddr_in*)add->netmask)->sin_ad
                        t++;
                }
        }
        open_dev = open(dev->name);//pcap_open
        if (open_dev == NULL) {
                pcap_freealldevs(alldevs);
        }
        pcap_freealldevs(alldevs);
}
```

- 打开选中的网络接口

```
pcap_t* open(char* name) {

        pcap_t* temp = pcap_open(name, 65536/*最大值*/, PCAP_OPENFLAG_PROMISCUOUS, 1000, NULL, e
        if (temp == NULL)
                printf("无法打开设备\n");
        return temp;
}
```

- 获取本机MAC地址

```
void Get_Host_Mac(DWORD ip) {
	memset(my_mac, 0, sizeof(my_mac));
	ARPFrame_t ARPF_Send;
	SET_ARP_Frame_HOST(ARPF_Send,ip);
	struct pcap_pkthdr* pkt_header;
	const u_char* pkt_data;
	struct pcap_pkthdr* header = new pcap_pkthdr;
	int k;
	my_log.addInfo("---------------------------------获取本机MAC地址-------------------------
	my_log.ARP_info("发送ARP请求包", &ARPF_Send);
	//发送构造好的数据包
	//用pcap_next_ex()捕获数据包，pkt_data指向捕获到的网络数据包

	//这里可能后续需要修改
	while ((k = pcap_next_ex(open_dev, &pkt_header, &pkt_data)) >= 0) {
		//发送数据包

		pcap_sendpacket(open_dev, (u_char*)&ARPF_Send, sizeof(ARPFrame_t));
		struct ARPFrame_t* arp_message;
		arp_message = (struct ARPFrame_t*)(pkt_data);
		if (k == 0)continue;
		else
		{   //帧类型为ARP，且操作类型为ARP响应

			if (arp_message->FrameHeader.FrameType == htons(0x0806) && arp_message-


				//展示一下包的内容
				my_log.ARP_info("收到ARP应答包", arp_message);
				ShowArp(arp_message);
				//用my_mac记录本机的MAC地址，
				for (int i = 0; i < 6; i++) {
					my_mac[i] = *(unsigned char*)(pkt_data + 22 + i);
				}
				printf("成功获取本机MAC地址\n");
				break;
			}
		}
	}
	my_log.addInfo("---------------------------------成功获取本机MAC地址--------------------
}
```

在获取本机的MAC地址时，需要先利用自定义的 SET_ARP_Frame_HOST 构造要发送的ARP数据包： DesMac 设置为广播地址； RecvHa 设置为0，在请求报文里表示目的地址未知； RecvIP 设置为本机IP地址；帧类型为ARP；硬件类型为以太网；协议类型为IP；硬件地址长度为6；协议地址长度为4；操作为ARP请求.

- 初始化路由表

```
RouteTable::RouteTable()
{
        head = new RouteEntry;
        head->next = NULL;
        num = 0;
        //通过得到的双IP的掩码，在路由表中添加直接相连的网络，将类型设置为0，即不可删除项
        for (int i = 0; i < 2; i++)
        {
                RouteEntry* entry = new RouteEntry;
                //添加直接相连的网络
                entry->dst_net = (inet_addr(my_ip[i])) & (inet_addr(my_mask[i]));
                entry->mask = inet_addr(my_mask[i]);
                entry->type = 0;
                this->add(entry);
        }
}
```

- 静态路由表项的维护

```
int op;
while (1)
{
        printf("\n\n请选择要进行的操作：\n1. 添加路由表项\n2. 删除路由表项\n3. 打印路由表\n");
        scanf("%d", &op);
        RouteEntry* entry = new RouteEntry;
        switch (op) {
        case 1:
                char t[30];
                printf("请输入掩码：");
                scanf("%s", &t);
                entry->mask = inet_addr(t);
                printf("请输入目的网络：");
                scanf("%s", &t);
                entry->dst_net= inet_addr(t);
                printf("请输入下一跳的IP地址：");
                scanf("%s", &t);
                entry->next_hop = inet_addr(t);
                entry->type = 1;
                entry->print();
                my_route.add(entry);
                break;
        case 2:
                my_route.print();
                printf("请输入要删除的表项的索引：");
                int i;
                scanf("%d", &i);
                my_route.erase(i);
                break;
        case 3:
                my_route.print();
                break;
        default:
                printf("无效操作，请重新输入\n");
                break;
        }
}
```

- 捕获数据包
  调用 pcap_next_ex 捕获数据包，格式化收到的包为帧首部，以获取目的MAC地址和帧类型，并且只处理目的MAC是本机MAC的数据包

```
        pcap_pkthdr* pkt_header = NULL;
        const u_char* pkt_data = NULL;
        while (1)
        {
                int ret = pcap_next_ex(open_dev, &pkt_header, &pkt_data);//抓包
                if (ret)break;//接收到消息
        }
        //格式化收到的包为帧首部，以获取目的MAC地址和帧类型
        FrameHeader_t* recv_header = (FrameHeader_t*)pkt_data;
//只处理目的目的mac是自己的包
        if (Compare(recv_header->DesMAC, my_mac))
...
```

- 数据包过滤
  设置捕获报文的过滤条件，通过绑定过滤器，设置只捕获IP和ARP数据包。主要通过调用 pcap_compile 和 pcap_setfilter 函数。

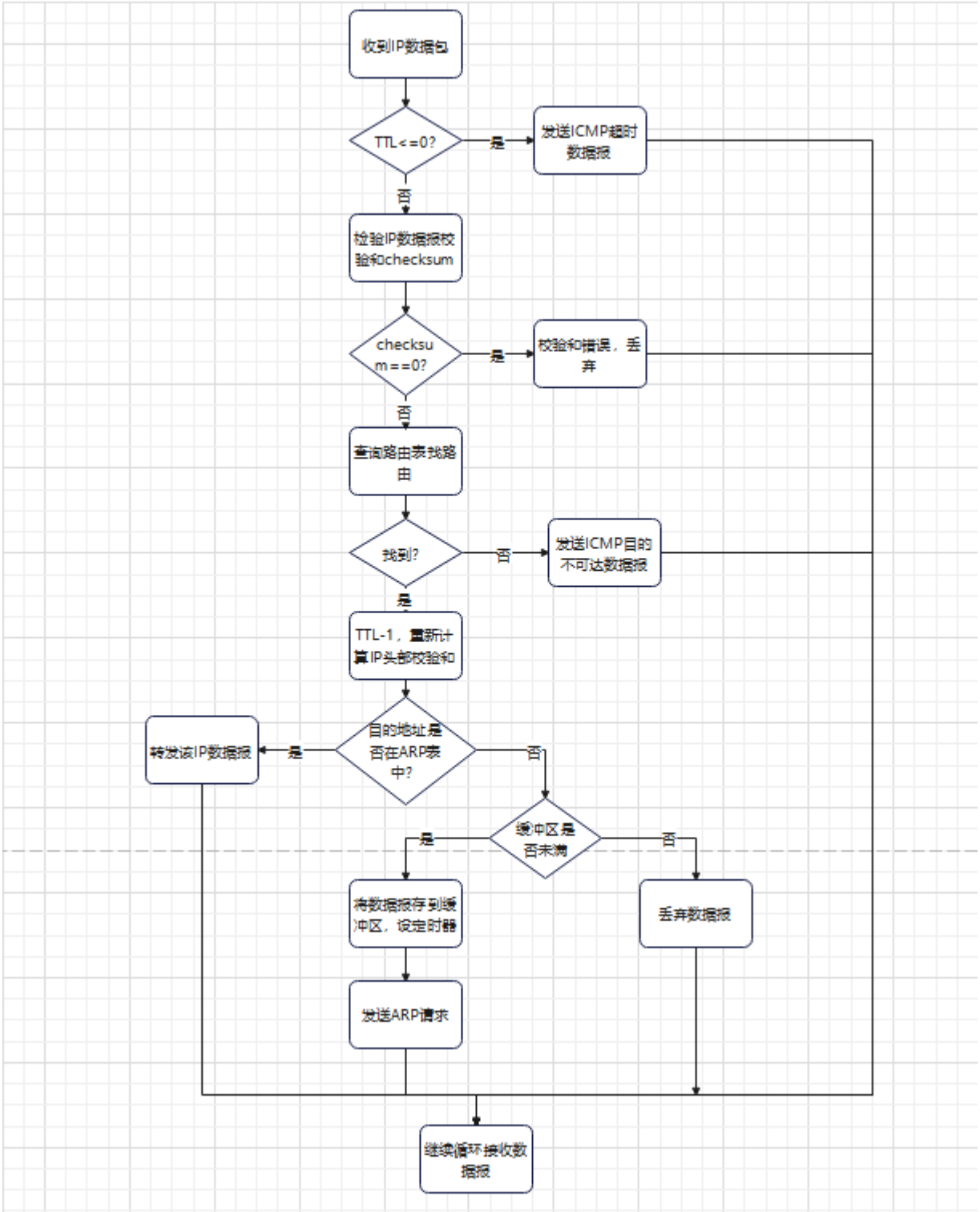
```
if (pcap_compile(open_dev, &fcode, "ip or arp", 1, bpf_u_int32(my_mask[0])) < 0)
{
        fprintf(stderr, "\nError  filter\n");
        system("pause");
        return -1;
}

//根据编译好的过滤码设置过滤条件
if (pcap_setfilter(open_dev, &fcode) < 0)
{
        fprintf(stderr, "\nError setting the filter\n");
        system("pause");
        return -1;
}
```

- 处理收到的IP数据包
  处理的主要思路如下：
  ① 查看报文中的TTL值，如果小于等于0的话，说明超时，发送ICMP超时报文，继续接收下一条报文。
  ② 查询路由表，找路由，找到对应目的地址的下一跳IP，如果查询结果为-1，发送ICMP目的不可达报文，继续接收下一条报文。
  ③ 计算报文中的IP校验和并进行检验，如果校验和错误直接丢弃数据包不处理，继续接收下一条数据包。

④重新封装数据包，将源地址设置为本机mac地址，TTL-1，将IP头部的校验和设置为0，重新计算IP头部校验和

⑤ 查看目的地址是否在ARP表（IP-MAC映射表）中，如果在，则直接转发。

⑥ IP-MAC地址映射表中不存在该映射关系,先缓存IP数据报入缓存队列 my_buffer 中，并设置定时器，发送ARP请求获取目的IP的MAC地址。设置缓冲队列最大为 MAX_BUFFER_COUNT ，如果缓冲区已经满了，则不做处理，提示信息："缓冲区已满，丢弃该IP数据包"。



关于发送ARP请求报文获得目的IP的MAC地址，首先要构造ARP数据报：DesMAC设置为广播地址；SrcMAC和SendHa为本机的MAC地址，用获得的my_mac数组填充；RecvHa设置为0；SendIP为本机的IP地址，用上面得到的ip设置；RecvIP设置为目的主机的IP地址，从参数中获取。

```
void SET_ARP_Frame_DEST(ARPFrame_t& ARPFrame, char ip[20], unsigned char* mac) {
        for (int i = 0; i < 6; i++) {
                ARPFrame.FrameHeader.DesMAC[i] = 0xff;//将APRFrame.FrameHeader.DesMAC设置为广播地
                ARPFrame.RecvHa[i] = 0x00;
                ARPFrame.FrameHeader.SrcMAC[i] = mac[i];//设置为本机网卡的MAC地址
                ARPFrame.SendHa[i] = mac[i];//设置为本机网卡的MAC地址
        }
        ARPFrame.FrameHeader.FrameType = htons(0x0806);//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
        ARPFrame.SendIP = inet_addr(ip);//将ARPFrame->SendIP设置为本机网卡上绑定的IP地址

}
```

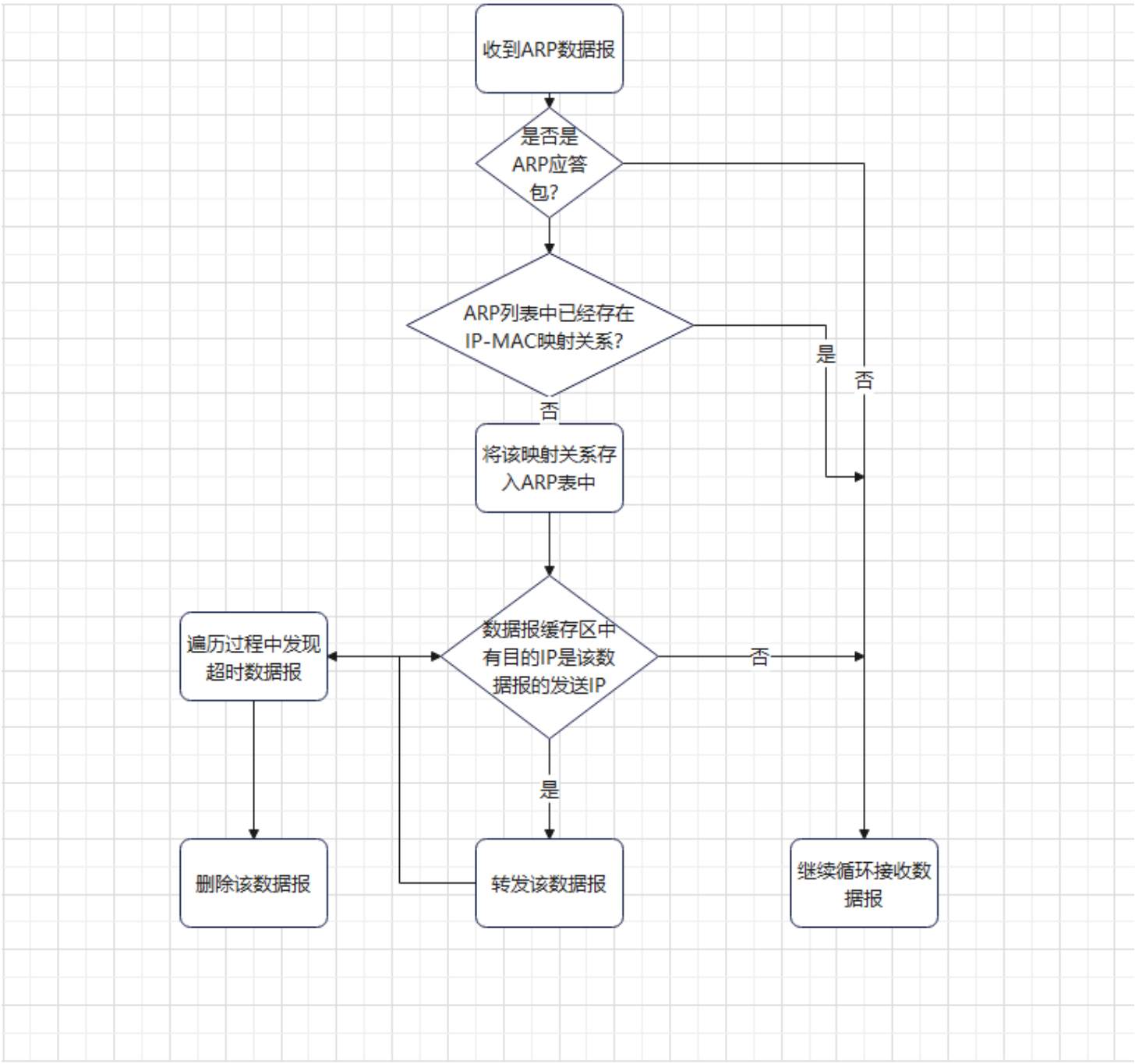
然后调用 `pcap_sendpacket` 发送构造好的数据报

```
if (ntohs(recv_header->FrameType) == 0x800)
{
        //格式化收到的包为帧首部+IP首部类型
        IPFrame_t* data = (IPFrame_t*)pkt_data;

        my_log.IP_info("接收IP数据报\n", data);
        //获取目的IP地址并在路由表中查找，并获取下一跳ip地址



        // ICMP超时
        if (data->IPHeader.TTL <= 0)
        {
                //发送超时报文
                Send_ICMP_Pac(11, 0, pkt_data);
                my_log.addInfo("发送ICMP超时数据包!");
                continue;
        }

        IPHeader_t* IpHeader = &(data->IPHeader);
        // 检验校验和，数据报损坏或是出错
        if (check_checksum(data) == 0)
        {
                my_log.IP_info("校验和错误，丢弃", data);
                continue;
        }
        DWORD dstip = data->IPHeader.DstIP;

        DWORD nexthop = router_table.search(dstip);

        my_log.addInfohop("接收到的IP数据报目的为", dstip);
        my_log.addInfohop("接收到的IP数据报下一跳为",nexthop);
        //无匹配项
        //目的不可达
        if (nexthop == -1)
        {
                Send_ICMP_Pac(3, 0, pkt_data);// ICMP目的不可达
                my_log.addInfo("发送ICMP目的不可达数据报!");
                continue;
        }
        else
        {
```

```
PacketBuffer packet;
packet.targetIP = nexthop;

//重新封装MAC地址(源MAC地址变为my_mac)
for (int t = 0; t < 6; t++)
{
        data->FrameHeader.SrcMAC[t] = my_mac[t];
}

data->IPHeader.TTL -= 1;// TTL减1
// 设IP头中的校验和为0
data->IPHeader.Checksum = 0;
unsigned short buff[sizeof(IPHeader_t)];

memset(buff, 0, sizeof(IPHeader_t));
IPHeader_t* header = &(data->IPHeader);
memcpy(buff, header, sizeof(IPHeader_t));

// 计算IP头部校验和
// data->IPHeader.Checksum = cal_checksum(check_buff, si
data->IPHeader.Checksum = calCheckSum1(header);

// IP-MAC地址映射表中存在该映射关系
//根据nexthop取ARP映射表中查看是否存在映射

if (my_arptable->search(nexthop, data->FrameHeader.DesMA
{
        //查找到了数据报可以直接转发
        memcpy(packet.pktData, pkt_data, pkt_header->le
        packet.totalLen = pkt_header->len;
        if (pcap_sendpacket(open_dev, (u_char*)packet.pk
        {
                // 错误处理
                continue;
        }
        my_log.addInfo("--------------------------------
        my_log.IP_info("转发", data);
}


// IP-MAC地址映射表中不存在该映射关系,获取
//先缓存IP数据报
//设置缓冲区my_buffer
```

```
                else
                {
                        //最多存50条
                        if (pktNum < MAX_BUFFER_COUNT)          // 存入缓
                        {
                                packet.totalLen = pkt_header->len;
                                // 将需要转发的数据报存入缓冲区
                                memcpy(packet.pktData, pkt_data, pkt_hea

                                my_buffer[pktNum++] = packet;

                                packet.Time = clock();

                                my_log.IP_info("缓存IP数据报\n", data);
                                // 发送ARP请求
                                Get_Other_Mac(packet.targetIP);
                        }
                        else
                        {
                                my_log.addInfo("缓冲区已满，丢弃该IP数据包
                                my_log.IP_info("缓冲区溢出，丢弃", data);
                        }
                }
        }
}
```

- 处理ARP数据包
  处理思路如下：
  ① 接收到ARP数据报后，先判断是否为响应报文，如果不是的话，什么也不做，直接继续接收新的报文。
  ② 在IP-MAC映射表 `my_arptable` 中查找是否已经存在对应的映射关系，如果存在的话，直接继续接收新的报文。
  ③ 在IP-MAC映射表 `my_arptable` 中不存在对应的映射关系，通过 `my_arptable->add(data->SendIP,data->SendHa)` 将映射关系存入，然后遍历缓冲区中是否有目的IP是该MAC对应的IP地址的数据报，有则进行转发。
  ④ 在遍历缓冲区的时候，如果发现了超时的数据报，则将其从缓冲区中删除，调用自定义函数 `Delete_Buffer` ，将已经占用缓冲区的总数据包个数 `pktNum -1` 。


收到ARP数据报
是否是ARP应答包?
ARP列表中已经存在IP-MAC映射关系?
是
否
否
将该映射关系存入ARP表中
遍历过程中发现超时数据报
数据报缓存区中有目的IP是该数据报的发送IP
否
是
删除该数据报
转发该数据报
继续循环接收数据报

```
else if (ntohs(recv_header->FrameType) == 0x806)
            {
                ARPFrame_t* data = (ARPFrame_t*)pkt_data;//格式化收到的包为帧首部-
                my_log.ARP_info("接收ARP响应包", data);
                //收到ARP响应包
                //处理响应报文
                if (data->Operation == ntohs(0x0002)) {
                    BYTE tmp_mac[6] = { 1 };
                    if (my_arptable->search(data->SendIP, tmp_mac)) {//该映射
                    }
                    else {
                        DWORD tmp_ip = data->SendIP;
                        for (int i = 0; i < 6; i++) {
                            tmp_mac[i] = data->SendHa[i];
                        }
                        //IP-MAC对应关系存表
                        my_arptable->add(data->SendIP, data->SendHa);
                        //遍历缓冲区，看是否有可以转发的包
                        for (int i = 0; i < pktNum; i++)
                        {
                            PacketBuffer packet = my_buffer[i];
                            if (packet.valid == 0)continue;
                            if (clock() - packet.Time >= 6000) {//超
                                // packet.valid = 0;
                                // my_buffer[i].valid = 0;
                                Delete_Buffer(my_buffer, i);
                                pktNum -= 1;
                                continue;
                            }
                            ////往此IP地址转发
                            if (packet.targetIP == data->SendIP)
                            {
                                IPFrame_t* ipframe = (IPFrame_t*
                                //重新封装IP包
                                for (int i = 0; i < 6; i++) {
                                    ipframe->FrameHeader.Src
                                    ipframe->FrameHeader.Des
                                }
                                // 发送IP数据包
                                pcap_sendpacket(open_dev, (u_cha
                                my_buffer[i].valid = 0;
                                my_log.addInfo("----------------
                                my_log.IP_info("转发", ipframe);
```

```
                    my_log.addInfo("----------
                }
            }
        }
    }
}
```

其他函数

- 计算校验和
  校验和计算的主要思路为：
  - 将校验和字段设置为0（传进来的是已经为0的）
  - 按顺序对每16位（1个字=2字节）进行加法运算；
  - 如果溢出就将进位加到最低位
  - 对累加的结果取反——就是头部校验和值

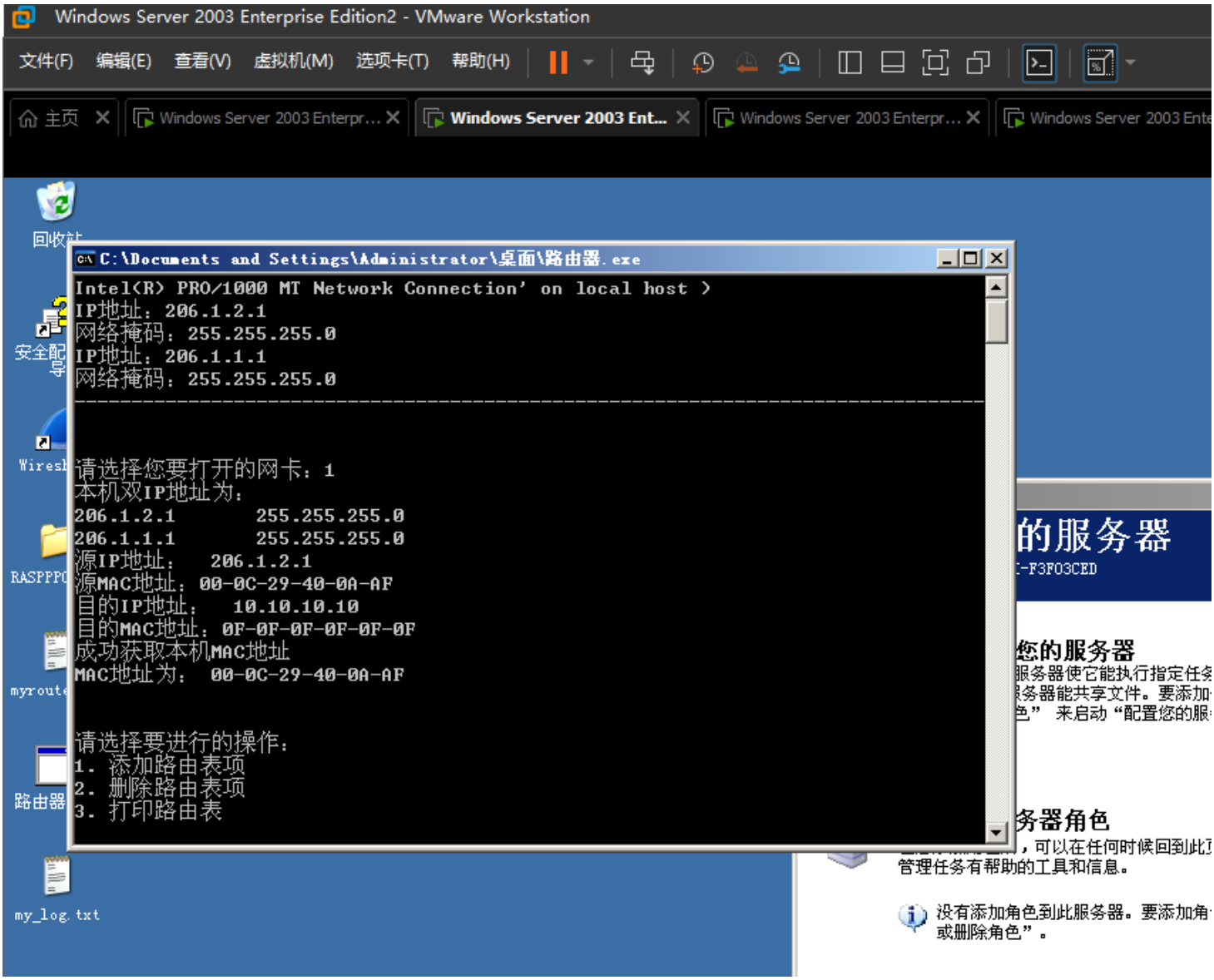
```
unsigned short calCheckSum1(IPHeader_t* temp)
{
	// temp->IPHeader.Checksum = 0;
	unsigned int sum = 0;
	//WORD* t = (WORD*)&temp->IPHeader;//每16位为一组
	WORD* t = (WORD*)temp;
	for (int i = 0; i < sizeof(IPHeader_t) / 2; i++) {
		sum += t[i];
		while (sum >= 0x10000) {
		//如果溢出，则进行回卷
			int s = sum >> 16;
			sum -= 0x10000;
			sum += s;
		}
	}

	// temp->Checksum = ~sum;//结果取反
	return (unsigned short)~sum;
}

unsigned short calCheckSum2(unsigned short* pBuffer, int nSize) {
	///*
	//* 计算方法：将校验和字段设置为0（传进来的是已经为0的）
	//* 按顺序对每16位（1个字=2字节）进行加法运算；
	//* 如果溢出就将进位加到最低位
	//* 对累加的结果取反—就是头部校验和值
	//*/
	unsigned long ulCheckSum = 0;
	while (nSize > 1)
	{
		ulCheckSum += *pBuffer++;
		nSize -= sizeof(unsigned short);//每16位一组
	}
	if (nSize)
	{
		ulCheckSum += *(unsigned short*)pBuffer;
	}

	ulCheckSum = (ulCheckSum >> 16) + (ulCheckSum & 0xffff);
	ulCheckSum += (ulCheckSum >> 16);
	return (unsigned short)(~ulCheckSum);
}
```

- 检验校验和

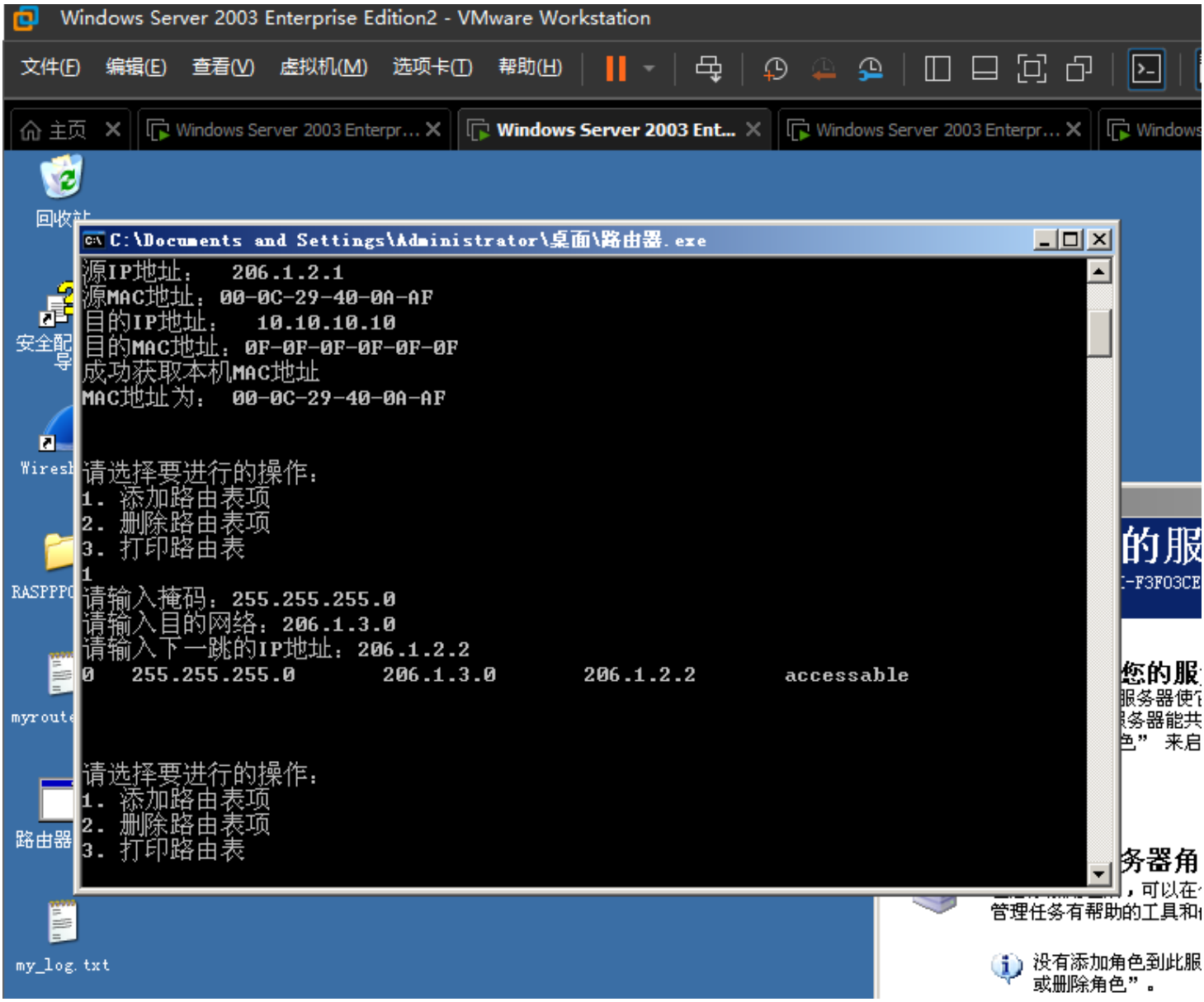
```
bool check_checksum(IPFrame_t* temp) {
        unsigned int sum = 0;
        WORD* t = (WORD*)&temp->IPHeader;//每16位为一组
        for (int i = 0; i < sizeof(IPHeader_t) / 2; i++) {
                sum += t[i];
                while (sum >= 0x10000) {
                        //包含原校验和一起进行相加
                        int s = sum >> 16;
                        sum -= 0x10000;
                        sum += s;
                }
        }

        if (sum == 65535)return 1;//全1，校验和正确
        return 0;
}
```

实验结果

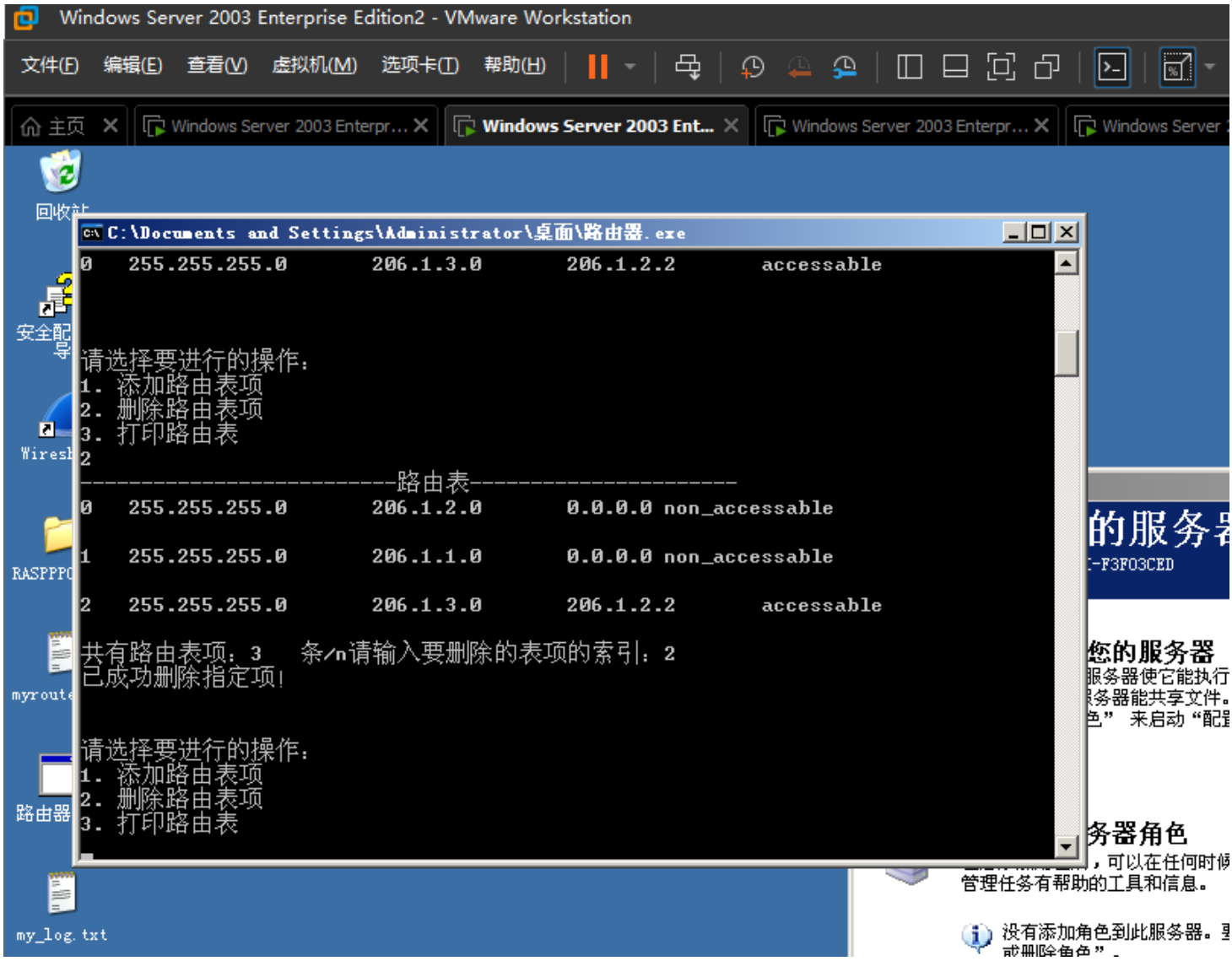
- 显示本机网卡的双IP地址并获取MAC地址


Windows Server 2003 Enterprise Edition2 - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
主页
Windows Server 2003 Enterpr...
Windows Server 2003 Ent...
Windows Server 2003 Enterpr...
Windows Server 2003 Ente
回收站
安全配置向导
Wiresh
RASPPPO
myroute
路由器
my_log.txt
C:\Documents and Settings\Administrator\桌面\路由器.exe
Intel(R) PRO/1000 MT Network Connection' on local host)
IP地址：206.1.2.1
网络掩码：255.255.255.0
IP地址：206.1.1.1
网络掩码：255.255.255.0
请选择您要打开的网卡：1
本机双IP地址为：
206.1.2.1 255.255.255.0
206.1.1.1 255.255.255.0
源IP地址： 206.1.2.1
源MAC地址：00-0C-29-40-0A-AF
目的IP地址： 10.10.10.10
目的MAC地址：0F-0F-0F-0F-0F-0F
成功获取本机MAC地址
MAC地址为： 00-0C-29-40-0A-AF
请选择要进行的操作：
1. 添加路由表项
2. 删除路由表项
3. 打印路由表
的服务器
您的服务器
务器角色
管理任务有帮助的工具和信息。
没有添加角色到此服务器。要添加角
或删除角色”。

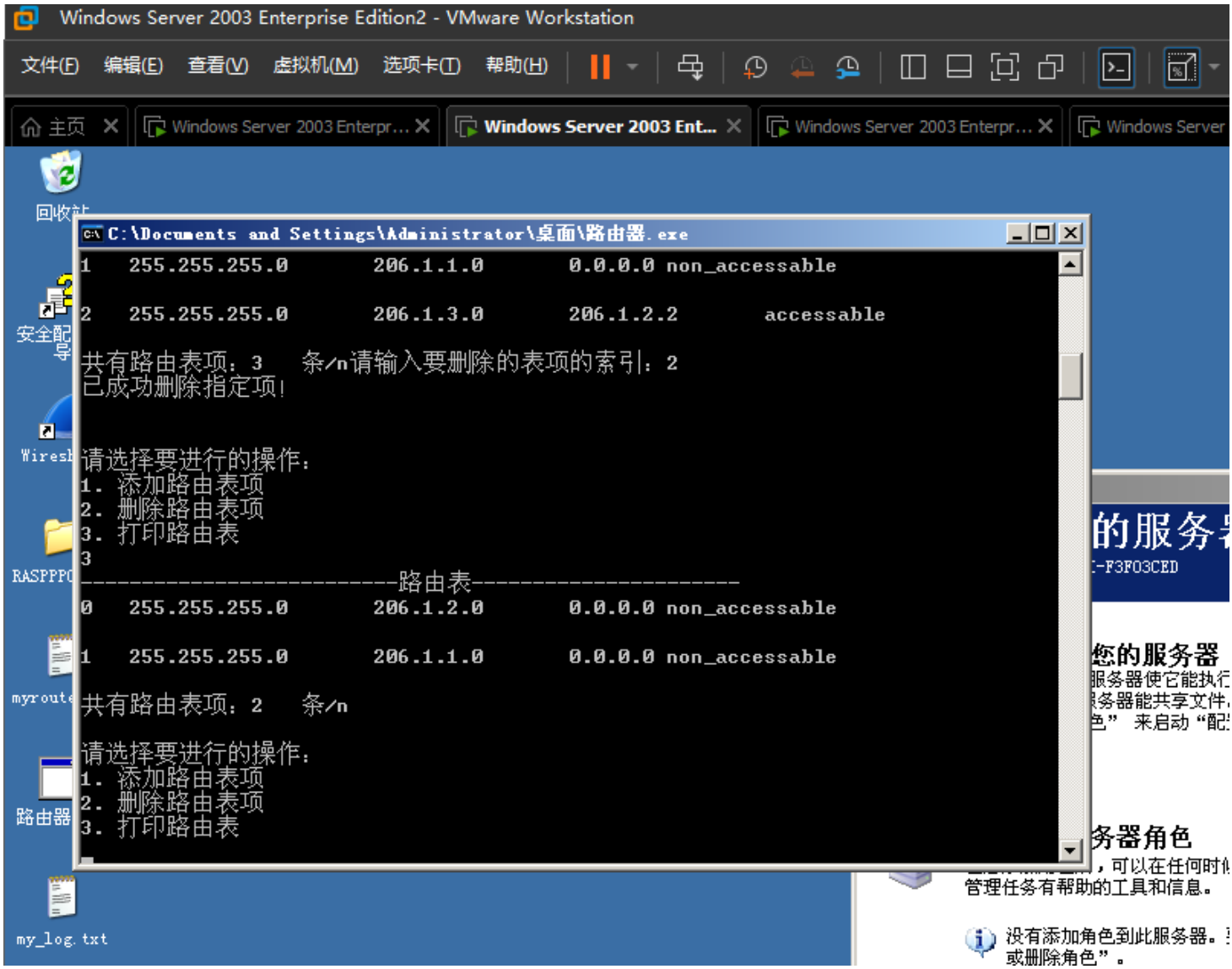
- 路由表项的添加


Windows Server 2003 Enterprise Edition2 - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
主页
Windows Server 2003 Enterpr...
Windows Server 2003 Ent...
Windows Server 2003 Enterpr...
C:\Documents and Settings\Administrator\桌面\路由器.exe
源IP地址： 206.1.2.1
源MAC地址：00-0C-29-40-0A-AF
目的IP地址： 10.10.10.10
目的MAC地址：0F-0F-0F-0F-0F-0F
成功获取本机MAC地址
MAC地址为： 00-0C-29-40-0A-AF
请选择要进行的操作：
1. 添加路由表项
2. 删除路由表项
3. 打印路由表
1
请输入掩码：255.255.255.0
请输入目的网络：206.1.3.0
请输入下一跳的IP地址：206.1.2.2
0 255.255.255.0 206.1.3.0 206.1.2.2 accessable
请选择要进行的操作：
1. 添加路由表项
2. 删除路由表项
3. 打印路由表
回收站
安全配置向导
RASPPPO
myroute
路由器
my_log.txt
没有添加角色到此服务器
或删除角色”。

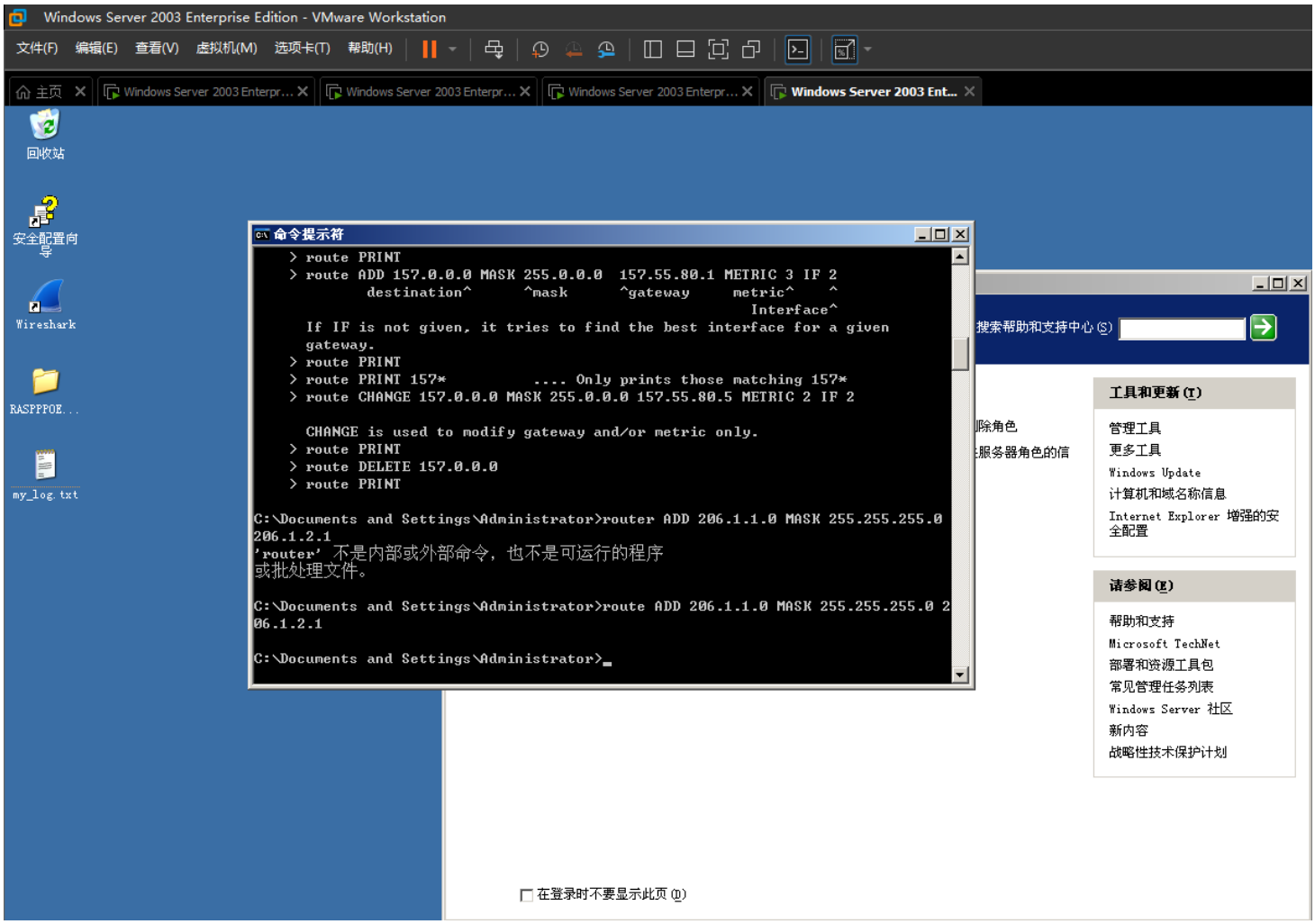
- 路由表项的删除


Windows Server 2003 Enterprise Edition2 - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
主页
Windows Server 2003 Enterpr...
Windows Server 2003 Ent...
Windows Server 2003 Enterpr...
回收站
安全配置向导
Wiresh
RASPPPO
myroute
路由器
my_log.txt
C:\Documents and Settings\Administrator\桌面\路由器.exe
0 255.255.255.0 206.1.3.0 206.1.2.2 accessable
请选择要进行的操作:
1. 添加路由表项
2. 删除路由表项
3. 打印路由表
2
----------路由表----------
0 255.255.255.0 206.1.2.0 0.0.0.0 non_accessable
1 255.255.255.0 206.1.1.0 0.0.0.0 non_accessable
2 255.255.255.0 206.1.3.0 206.1.2.2 accessable
共有路由表项: 3 条/n请输入要删除的表项的索引: 2
已成功删除指定项!
请选择要进行的操作:
1. 添加路由表项
2. 删除路由表项
3. 打印路由表
的服务器
您的服务器
务器角色
管理任务有帮助的工具和信息。
没有添加角色到此服务器。

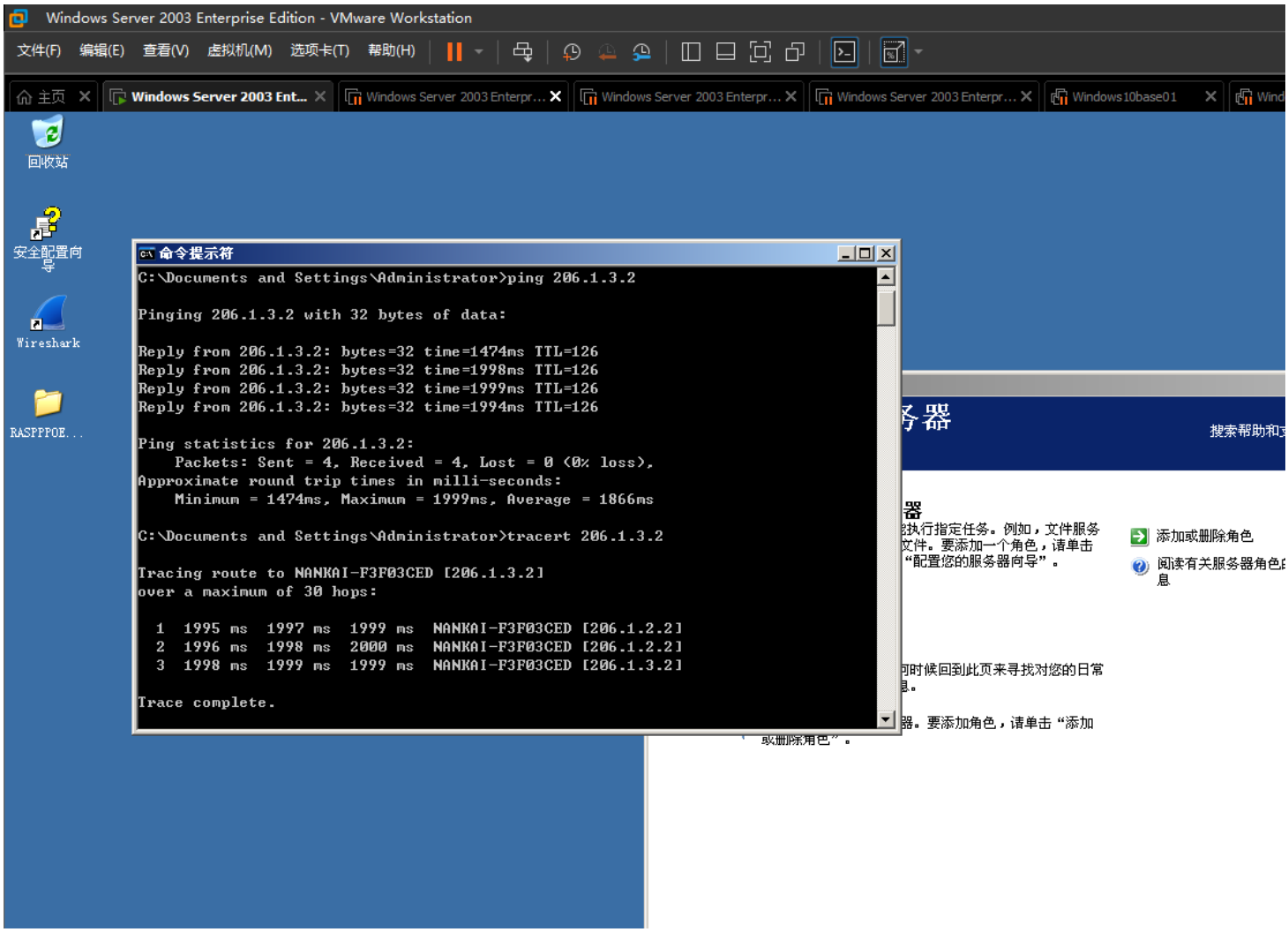
- 路由表项的打印



- 路由器2添加静态路由表项

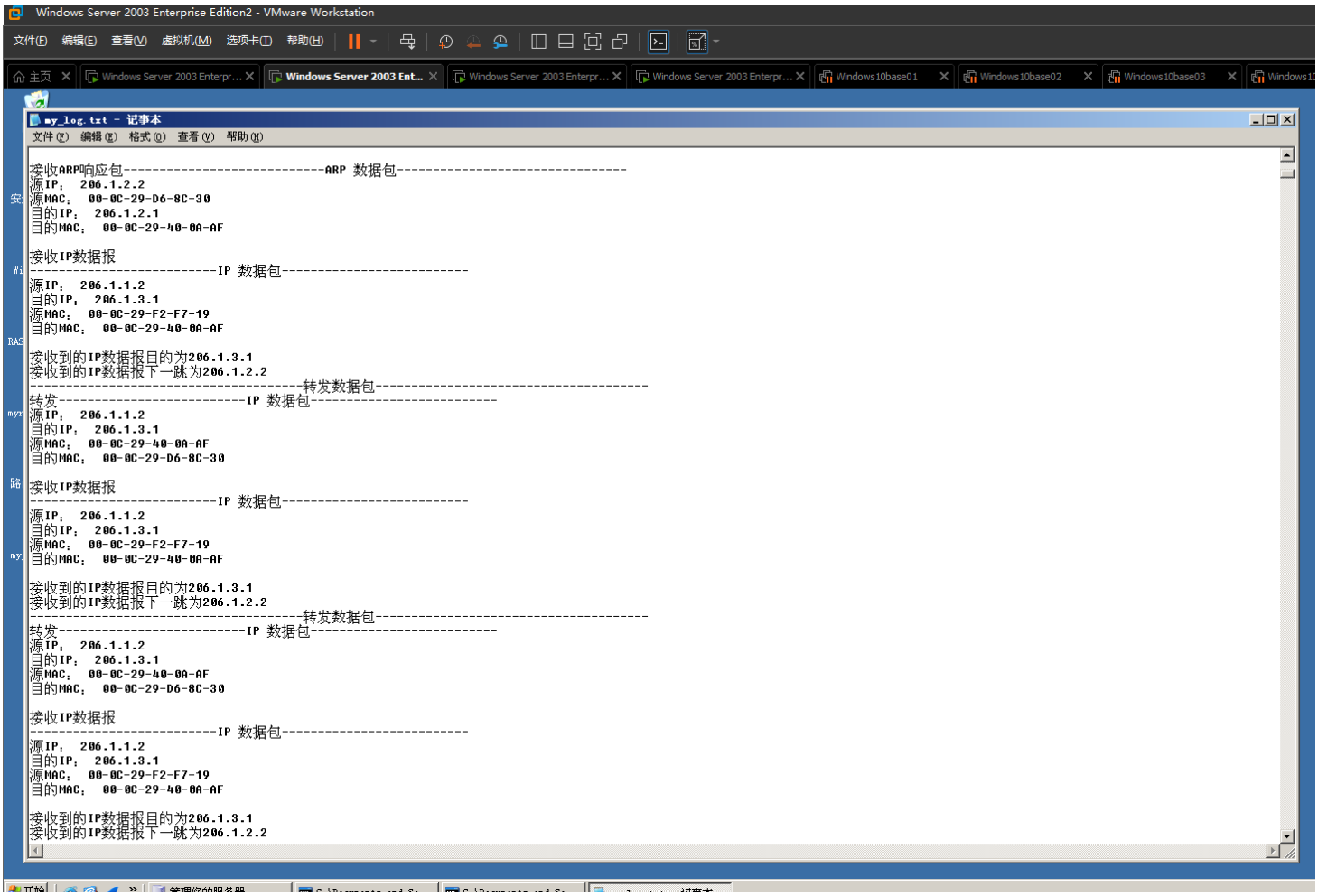

Windows Server 2003 Enterprise Edition - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
主页
Windows Server 2003 Enterpr...
Windows Server 2003 Enterpr...
Windows Server 2003 Enterpr...
Windows Server 2003 Ent...
回收站
安全配置向导
Wireshark
RASPPPOE...
my_log.txt
命令提示符
> route PRINT
> route ADD 157.0.0.0 MASK 255.0.0.0 157.55.80.1 METRIC 3 IF 2
destination^ ^mask ^gateway metric^ ^
Interface^
If IF is not given, it tries to find the best interface for a given
gateway.
> route PRINT
> route PRINT 157* .... Only prints those matching 157*
> route CHANGE 157.0.0.0 MASK 255.0.0.0 157.55.80.5 METRIC 2 IF 2
CHANGE is used to modify gateway and/or metric only.
> route PRINT
> route DELETE 157.0.0.0
> route PRINT
C:\Documents and Settings\Administrator>router ADD 206.1.1.0 MASK 255.255.255.0
206.1.2.1
'router' 不是内部或外部命令，也不是可运行的程序
或批处理文件。
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>
搜索帮助和支持中心 (S)
工具和更新 (T)
管理工具
更多工具
Windows Update
计算机和域名称信息
Internet Explorer 增强的安全配置
请参阅 (E)
帮助和支持
Microsoft TechNet
部署和资源工具包
常见管理任务列表
Windows Server 社区
新内容
战略性技术保护计划
在登录时不要显示此页 (D)

- 主机A ping 主机B


Windows Server 2003 Enterprise Edition - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
主页
Windows Server 2003 Ent...
Windows Server 2003 Enterpr...
Windows Server 2003 Enterpr...
Windows Server 2003 Enterpr...
Windows10base01
回收站
安全配置向导
Wireshark
RASPPPOE...
命令提示符
C:\Documents and Settings\Administrator>ping 206.1.3.2
Pinging 206.1.3.2 with 32 bytes of data:
Reply from 206.1.3.2: bytes=32 time=1474ms TTL=126
Reply from 206.1.3.2: bytes=32 time=1998ms TTL=126
Reply from 206.1.3.2: bytes=32 time=1999ms TTL=126
Reply from 206.1.3.2: bytes=32 time=1994ms TTL=126
Ping statistics for 206.1.3.2:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 1474ms, Maximum = 1999ms, Average = 1866ms
C:\Documents and Settings\Administrator>tracert 206.1.3.2
Tracing route to NANKAI-F3F03CED [206.1.3.2]
over a maximum of 30 hops:
1 1995 ms 1997 ms 1999 ms NANKAI-F3F03CED [206.1.2.2]
2 1996 ms 1998 ms 2000 ms NANKAI-F3F03CED [206.1.2.2]
3 1998 ms 1999 ms 1999 ms NANKAI-F3F03CED [206.1.3.2]
Trace complete.
添加或删除角色

成功!

- 日志打印


Windows Server 2003 Enterprise Edition2 - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
主页
Windows Server 2003 Enterpr...
Windows Server 2003 Ent...
Windows Server 2003 Enterpr...
Windows Server 2003 Enterpr...
Windows10base01
Windows10base02
Windows10base03
my_log.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
接收ARP响应包----------ARP 数据包----------
源IP： 206.1.2.2
源MAC： 00-0C-29-D6-8C-30
目的IP： 206.1.2.1
目的MAC： 00-0C-29-40-0A-AF
接收IP数据报
----------IP 数据包----------
源IP： 206.1.1.2
目的IP： 206.1.3.1
源MAC： 00-0C-29-F2-F7-19
目的MAC： 00-0C-29-40-0A-AF
接收到的IP数据报目的为206.1.3.1
接收到的IP数据报下一跳为206.1.2.2
----------转发数据包----------
转发----------IP 数据包----------
源IP： 206.1.1.2
目的IP： 206.1.3.1
源MAC： 00-0C-29-40-0A-AF
目的MAC： 00-0C-29-D6-8C-30
接收IP数据报
----------IP 数据包----------
源IP： 206.1.1.2
目的IP： 206.1.3.1
源MAC： 00-0C-29-F2-F7-19
目的MAC： 00-0C-29-40-0A-AF
接收到的IP数据报目的为206.1.3.1
接收到的IP数据报下一跳为206.1.2.2
----------转发数据包----------
转发----------IP 数据包----------
源IP： 206.1.1.2
目的IP： 206.1.3.1
源MAC： 00-0C-29-40-0A-AF
目的MAC： 00-0C-29-D6-8C-30
接收IP数据报
----------IP 数据包----------
源IP： 206.1.1.2
目的IP： 206.1.3.1
源MAC： 00-0C-29-F2-F7-19
目的MAC： 00-0C-29-40-0A-AF
接收到的IP数据报目的为206.1.3.1
接收到的IP数据报下一跳为206.1.2.2