

网络技术与应用课程实验报告

实验5：简化路由器程序设计

姓名：孟启轩 学号：2212452 专业：计算机科学与技术

一、实验内容说明

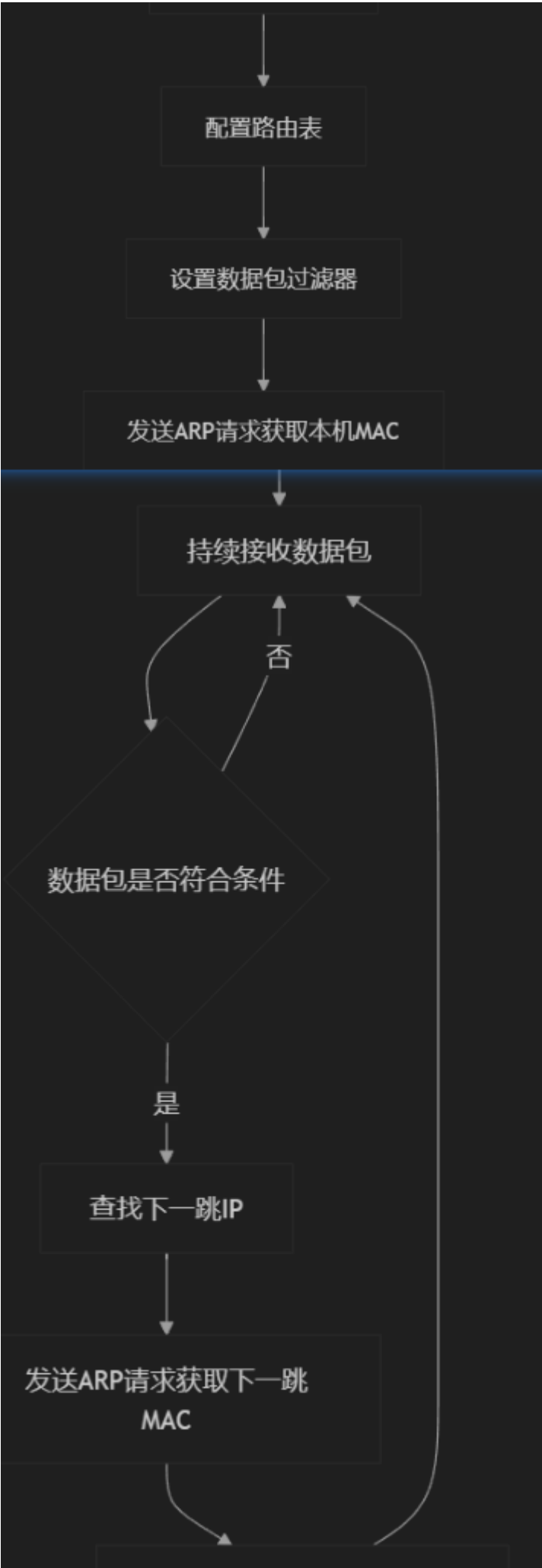
简单路由器程序设计实验的具体要求为：

- 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. 需要给出路由表的建立和形成方式。
- 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，并讲解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、前期准备

(1)流程图如下：





修改数据包MAC地址并转发

(2)与已完成实验的关联

pcap_findalldevs_ex

`pcap_findalldevs_ex` 函数与 `pcap_findalldevs` 是相关联的两个函数。它们的参数类型为 `pcap_if_t`，该类型作为链表头，用于表示可用的网络设备列表。每个链表节点包含两个重要字段：`name` 和 `description`，分别用于描述设备的名称和描述信息。

在 `pcap_if_t` 结构体中，除了存储基本的设备信息之外，还包含了一个 `pcap_addr` 结构体。`pcap_addr` 结构体进一步包含了多个信息，如地址列表、掩码列表、广播地址列表和目的地址列表等。通过使用 `pcap_findalldevs_ex` 函数，用户不仅可以获取本地设备的信息，还能够获取远程适配器的信息，并且它还支持返回指定本地文件夹中的 `pcap` 文件列表。

pcap_open

`pcap_open_live` 函数用于打开计算机上的网络适配器以进行数据包捕获，该函数需要接受五个参数：

- **name (适配器名称/GUID)：** 要打开的网络适配器的唯一标识符或名称。
- **snaplen (捕获数据包的部分长度)：** 定义要捕获数据包中的哪一部分。在某些操作系统中，驱动程序可以配置为仅捕获数据包的初始部分，从而减少应用程序之间复制数据的量，提高捕获效率。在此实验中，`snaplen` 设置为 65535，以确保能够捕获到完整的数据包，比最大的 MTU（最大传输单元）还要大。
- **flags (标志，包含混杂模式开关)：** `flags` 参数主要用于包含混杂模式（PROMISCUOUS mode）开关。通常情况下，适配器只接收发给它自己的数据包，而那些在其他机器之间通讯的数据包将会被丢弃。但是通过启用混杂模式，可以捕获所有经过适配器的数据包，包括发送给其他设备的数据包。
- **to_ms (读取数据的超时时间，以毫秒为单位)：** 指定在使用其他 API 进行读取操作时的超时时间。即使没有数据包到达或者捕获失败，这些函数也会在指定的时间内响应。在统计模式下，`to_ms` 还可以用来定义统计的时间间隔。将其设置为 0 表示没有超时，即如果没有数据包到达，则函数将永远不返回；设置为 -1 表示读取操作立即返回。
- **errbuf (用于存储错误信息字符串的缓冲区)：** 用于存储函数执行过程中产生的错误信息的缓冲区。

该函数执行成功后，将返回一个 `pcap_t` 类型的 handle，该 handle 可以用于后续的数据包捕获和处理操作。

pcap_next_ex

p (pcap_t 类型的 handle): 类型：pcap_t* 描述：该参数是一个 pcap_t 结构体，表示通过 `pcap_open_live` 或类似函数返回的捕获会话的句柄。它用于代表与网络适配器的连接及其捕获参数。**pkt_header (数据包头信息):** 类型：struct pcap_pkthdr 描述：这是一个指向数据包头信息的指针，结构体 `pcap_pkthdr` 包含数据包的时间戳、实际长度及在网络上捕获的长度信息。**pkt_data (数据包内容):** 类型：const u_char* 描述：这是一个指向数据包内容的指针，u_char 为无符号字符型，通常用于表示二进制数据。。**返回值:** 类型：int 描述：函数返回一个整数，代表操作的状态。返回正数表示成功捕获数据包，并且返回值表示捕获到的数据包长度；返回值为 0 表示达到捕获超时，未能捕获到数据包。

(3)ARP协议

ARP请求

任何时候，当主机需要找出这个网络中的另一个主机的物理地址时，它就可以发送一个ARP请求报文，这个报文包好了发送方的MAC地址和IP地址以及接收方的IP地址。因为发送方不知道接收方的物理地址，所以这个查询分组会在网络层中进行广播。

ARP响应

局域网中的每一台主机都会接受并处理这个ARP请求报文，然后进行验证，查看接收方的IP地址是不是自己的地址，只有验证成功的主机才会返回一个ARP响应报文，这个响应报文包含接收方的IP地址和物理地址。这个报文利用收到的ARP请求报文中的请求方物理地址以单播的方式直接发送给ARP请求报文的请求方。

(4)路由器

路由器定义：

路由器（Router）是在网络层实现网络互连，可实现网络层、链路层和物理层协议转换。也就是说，路由器是一种利用协议转换技术将异种网进行互联的设备。

路由器的工作流程：

1. 路由功能：

- 路由器的主要功能之一是选择经过的IP数据报的路由。这涉及到从接收到的IP数据报中提取目的IP地址，然后根据路由表信息选择最优的转发路径。路由器采用表驱动的路由选择算法，其中设备保存一张IP路由表，用于存储关于目的地址以及到达目的地址的最佳路径的信息。通过查询这张路由表，路由器能够确定应该将数据报转发到何处。

2. TTL字段处理：

- 路由器处理IP数据报的TTL（生存时间）字段。它会丢弃TTL值小于等于0的数据报，并将要转发的IP数据报的TTL减1。这是为了确保数据报在网络中不会无限循环。重新计算IP数据报头部的校验和以确保数据的完整性。

3. ICMP报文生成和处理：

- 路由器生成和处理与ICMP相关的报文，包括目的不可达报文和超时报文。当路由器无法为数据包找到适当的路由或主机时，它会丢弃数据报并向源主机发送ICMP目的不可达报文。对于TTL减为零的数据包，路由器会发送ICMP超时报文。这些报文用于通知源主机有关网络状况的变化。

4. 路由协议的实现：

- 实现路由协议，维护静态路由。路由器需要通过路由协议动态地学习和更新路由表，以适应网络拓扑的变化。同时，静态路由允许管理员手动配置特定的路由信息。

5. ARP协议的实现：

- 路由器实现ARP（地址解析协议）以获取下一跳的物理地址。在将IP数据报发送到下一跳之前，路由器需要确定下一跳的物理地址，以便构建数据帧并通过选择的网络接口发送出去。ARP协议用于解析IP地址到MAC地址的映射。

三、实验过程

(1)设计思路

提取目的IP地址并进行路由选择：

使用IP路由选择算法，路由器从捕获到的IP数据报中提取目的IP地址，并查询路由表以选择最优的转发路径。如果路由选择成功，记录下需要将数据报投递到的下一路由器地址；如果选择不成功，简单地丢弃该数据报。

获取下一站路由器接口的MAC地址：

在将路由选择成功的IP数据报发送到相应的接口之前，需要获取下一站路由器接口的MAC地址。为了实现这一步，路由器使用ARP请求方法：

- 获取网络接口卡列表，选择需要捕获MAC地址的网卡A（或选择对应的IP）。
- 构造ARP请求报文S，其中包括ARP请求、广播、伪造源MAC地址和源IP地址，以及目的IP地址为网卡A的IP地址。
- 通过网卡A发送构造的ARP请求报文S。
- 对网卡A进行流量监听，筛选其中的ARP报文（类型为0x806），捕获网卡A的ARP响应报文。在ARP响应报文的帧首部源MAC地址部分可以找到发送该ARP响应的网卡对应的MAC地址。

封装IP数据报为数据帧并发送：

一旦获取了下一站路由器接口的MAC地址，路由器可以将IP数据报封装成数据帧，并通过相应的接口发送出去。

本次实验可以被视为前两个实验的延续，涵盖了获取MAC地址的过程以及对路由选择和数据帧发送的处理。

(2)关键数据结构及关键函数分析

- 帧首部

```
// 数据帧首部
typedef struct FrameHeader_t
{
    BYTE DesMac[6]; // 目的mac地址, 6字节
    BYTE SrcMac[6]; // 源mac地址, 6字节
    WORD FrameType; // 帧类型, 指示帧的协议类型
} FrameHeader_t;
```

- ARP报文和IP报文

```
// ARP帧
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;                 // 协议地址长度
    WORD Operation;            // 操作类型 (请求或响应)
```

```

    BYTE SendHa[6];           // 发送方MAC地址
    DWORD SendIP;             // 发送方IP地址
    BYTE RecvHa[6];           // 接收方MAC地址
    DWORD RecvIP;             // 接收方IP地址
} ARPFrame_t;

// IP数据包的首部
typedef struct IPHeader_t
{
    BYTE Ver_HLen;            // 版本与首部长度
    BYTE TOS;                 // 服务类型
    WORD TotalLen;            // 整个IP数据包的总长度
    WORD ID;                  // 标识
    WORD Flag_Segment;        // 标志和偏移（分段信息）
    BYTE TTL;                 // 生存周期，数据包在网络中可以跳跃的最大次数
    BYTE Protocol;            // 协议类型
    WORD Checksum;            // 校验和
    ULONG SrcIP;              // 源IP地址
    ULONG DstIP;              // 目的IP地址
} IPHeader_t;

```

- 路由器表项：包括掩码、目的网络和下一站路由

```

// 路由表结构
typedef struct router_table
{
    ULONG netmask; // 网络掩码
    ULONG desnet;  // 目的网络
    ULONG nexthop; // 下一站路由
} router_table;

```

- 路由表：采用的结构体是数组的形式，对路由表的操作如下：

```

// 向路由表中添加项（没有做插入时排序的优化）
bool additem(router_table *t, int &tLength, router_table item)
{
    if (tLength == RT_TABLE_SIZE) // 路由表满则不能添加
        return false;
    for (int i = 0; i < tLength; i++)
        if ((t[i].desnet == item.desnet) && (t[i].netmask == item.netmask) &&
            (t[i].nexthop == item.nexthop))
            return false; // 路由表中已存在该项，则不能添加
    t[tLength] = item;      // 添加到表尾
    tLength = tLength + 1;
    return true;
}

// 从路由表中删除项
bool deleteitem(router_table *t, int &tLength, int index)

```

```

{
    if (tLength == 0) // 路由表空则不能删除
        return false;
    for (int i = 0; i < tLength; i++)
        if (i == index) // 当前索引等于目标索引，删除表项
        {
            for (; i < tLength - 1; i++)
                t[i] = t[i + 1];
            tLength = tLength - 1;
            return true;
        }
    return false; // 路由表中不存在该项则不能删除
}

```

- 路由匹配算法：最长匹配算法

这是最常用的路由匹配算法。它选择匹配目标IP地址的最长路由前缀。IP地址通常是按位分割的，而路由表中的每个条目都有一个前缀（网络部分）。最长前缀匹配选择路由表中最长的前缀与目标IP地址匹配，确保选择最具体的路由

```

// 选路 实现最长匹配
// 确保了选择的路由条目是最具体的，从而提高了路由的准确性和效率
ULONG search(router_table *t, int tLength, ULONG DesIP) // 返回下一跳步的IP
{
    ULONG best_desnet = 0; // 最优匹配的目的网络
    int best = -1; // 最优匹配路由表项的下标
    for (int i = 0; i < tLength; i++) // 遍历路由表t
    {
        if ((t[i].netmask & DesIP) == t[i].desnet) // 检查目标IP地址与网络掩码的按位
            // 与结果是否等于目的网络desnet
        {
            if (t[i].desnet >= best_desnet) // 最长匹配
            {
                best_desnet = t[i].desnet; // 保存最优匹配的目的网络
                best = i; // 更新最优匹配路由表项的下标
            }
        }
    }
    if (best == -1)
        return 0xffffffff; // 没有匹配项
    else
        return t[best].nexthop; // 获得匹配项
}

```

主函数逻辑：获取设备列表，然后选择双网卡的设备进行打开，并将刚刚获取的双IP输出，进入一个while循环，对路由表项进行操作，利用arp协议获取本机的mac地址，进入while循环接收数据包，对数据包的目的ip发送mac请求，收到mac请求后将数据包的目的mac修改为对应的mac，发送数据包。

- 获取设备列表：

```
// 获取本机网卡信息
int num = 0; // 记录有几个网络接口卡
if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf) == -1)
{
    cout << "have errors" << endl;
}
```

- 过滤, 只要ARP和IP包:

```
u_int net_mask;
char packet_filter[] = "ip or arp";
struct bpf_program fcode;
net_mask = ((sockaddr_in *) (d->addresses->netmask))->sin_addr.S_un.S_addr;
if (pcap_compile(p, &fcode, packet_filter, 1, net_mask) < 0)
{
    printf("无法编译包过滤器。请检查语法! \n");
    pcap_freealldevs(alldevs);
    return 0;
}
if (pcap_setfilter(p, &fcode) < 0)
{
    printf("过滤器设置错误\n");
    pcap_freealldevs(alldevs);
    return 0;
}
```

- 使用一个虚拟IP来获取本机MAC地址:

```
// 向自己发送arp包, 获取本机的MAC
BYTE scrMAC[6];
ULONG scrIP;
for (i = 0; i < 6; i++)
{
    scrMAC[i] = 0x66;
}
scrIP = inet_addr("112.112.112.112"); // 虚拟IP

for (d = alldevs, i = 0; i < in; i++, d = d->next)
;
for (a = d->addresses; a != NULL; a = a->next)
{
    if (a->addr->sa_family == AF_INET)
    {
        targetIP = inet_addr(inet_ntoa(((struct sockaddr_in *) (a->addr))->sin_addr));
        my_ip = targetIP;
    }
}
```



```

ARPFrame_t ARPFrame;
for (int i = 0; i < 6; i++)
{
    ARPFrame.FrameHeader.DesMac[i] = 0xff;
    ARPFrame.FrameHeader.SrcMac[i] = scrMAC[i];
    ARPFrame.SendHa[i] = scrMAC[i];
    ARPFrame.RecvHa[i] = 0;
}
ARPFrame.FrameHeader.FrameType = htons(0x0806);
ARPFrame.HardwareType = htons(0x0001);
ARPFrame.ProtocolType = htons(0x0800);
ARPFrame.HLen = 6;
ARPFrame.PLen = 4;
ARPFrame.Operation = htons(0x0001);
ARPFrame.SendIP = scrIP;
ARPFrame.RecvIP = targetIP;
int ret_send = pcap_sendpacket(p, (u_char *)&ARPFrame, sizeof(ARPFrame_t));
cout << "向自己发包成功" << endl;

// 截获自己的MAC
pcap_pkthdr *pkt_header1 = new pcap_pkthdr[1500];
const u_char *pkt_data1;
int res;
ARPFrame_t *ARPFrame1;

```

- 获取目的mac为本机mac，目的ip非本机ip的ip数据报

```

WORD RecvChecksum;
WORD FrameType;

IPPacket = (IPData_t *)pkt_data;
ULONG Len = pkt_header->len + sizeof(FrameHeader_t); // 数据包大小包括帧数据部分
长度和帧首部长
u_char *sendAllPacket = new u_char[Len];
for (i = 0; i < Len; i++)
{
    sendAllPacket[i] = pkt_data[i];
}

RecvChecksum = IPPacket->IPHeader.Checksum;
IPPacket->IPHeader.Checksum = 0;
FrameType = IPPacket->FrameHeader.FrameType;
bool desmac_equal = 1; // 目的mac地址与本机mac地址是否相同，相同为1;
for (int i = 0; i < 6; i++)
{
    if (my_mac[i] != IPPacket->FrameHeader.DesMac[i])
    {
        desmac_equal = 0;
    }
}
bool desIP_equal = 0; // 目的IP与本机IP是否相同，不相同为1;
if (IPPacket->IPHeader.DstIP != my_ip)

```

```
{
    desIP_equal = 1;
    targetIP = IPPacket->IPHeader.DstIP;
}
bool Is_ipv4 = 0;
if (FrameType == 0x0008)
{
    Is_ipv4 = 1;
}
```

- 向nextIP发ARP请求，获取下一跳的MAC地址:

```
for (i = 0; i < 6; i++)
{
    scrMAC[i] = my_mac[i];
}
scrIP = my_ip;
targetIP = nextIP;
```

- 组装ARP包:

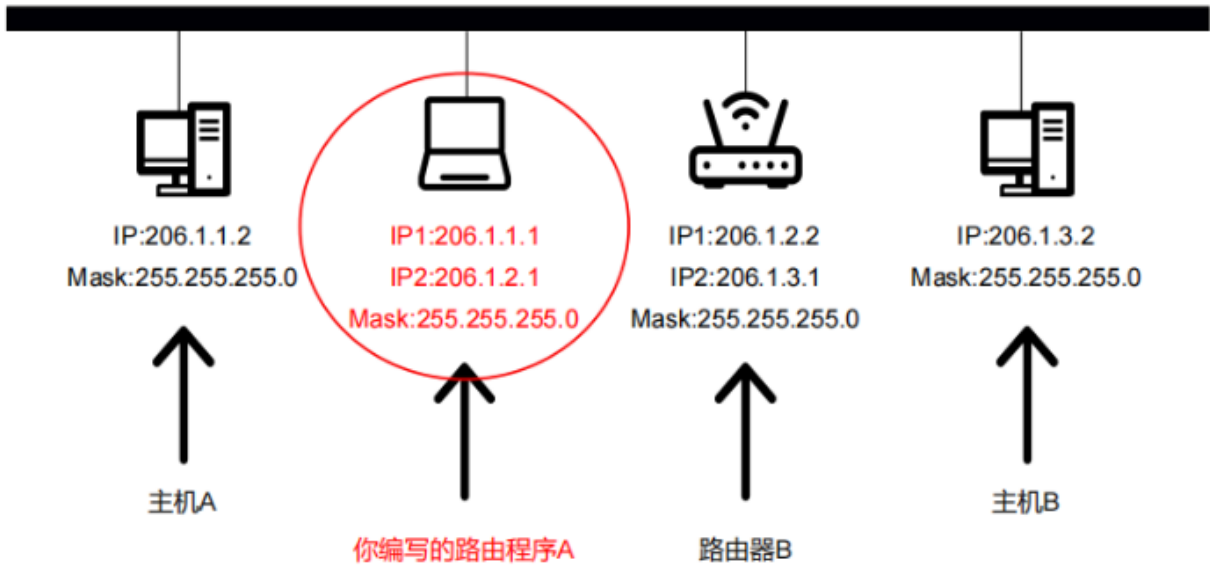
```
for (int i = 0; i < 6; i++)
{
    ARPFrame.FrameHeader.DesMac[i] = 0xff;
    ARPFrame.FrameHeader.SrcMac[i] = scrMAC[i];
    ARPFrame.SendHa[i] = scrMAC[i];
    ARPFrame.RecvHa[i] = 0;
}
ARPFrame.FrameHeader.FrameType = htons(0x0806);
ARPFrame.HardwareType = htons(0x0001);
ARPFrame.ProtocolType = htons(0x0800);
ARPFrame.HLen = 6;
ARPFrame.PLen = 4;
ARPFrame.Operation = htons(0x0001);
ARPFrame.SendIP = scrIP;
```

到目前为止知道了：自己的IP，自己的MAC，要转发的目的IP，要转发的目的MAC。可以向下一跳发送报文了。

(3)配置及运行演示

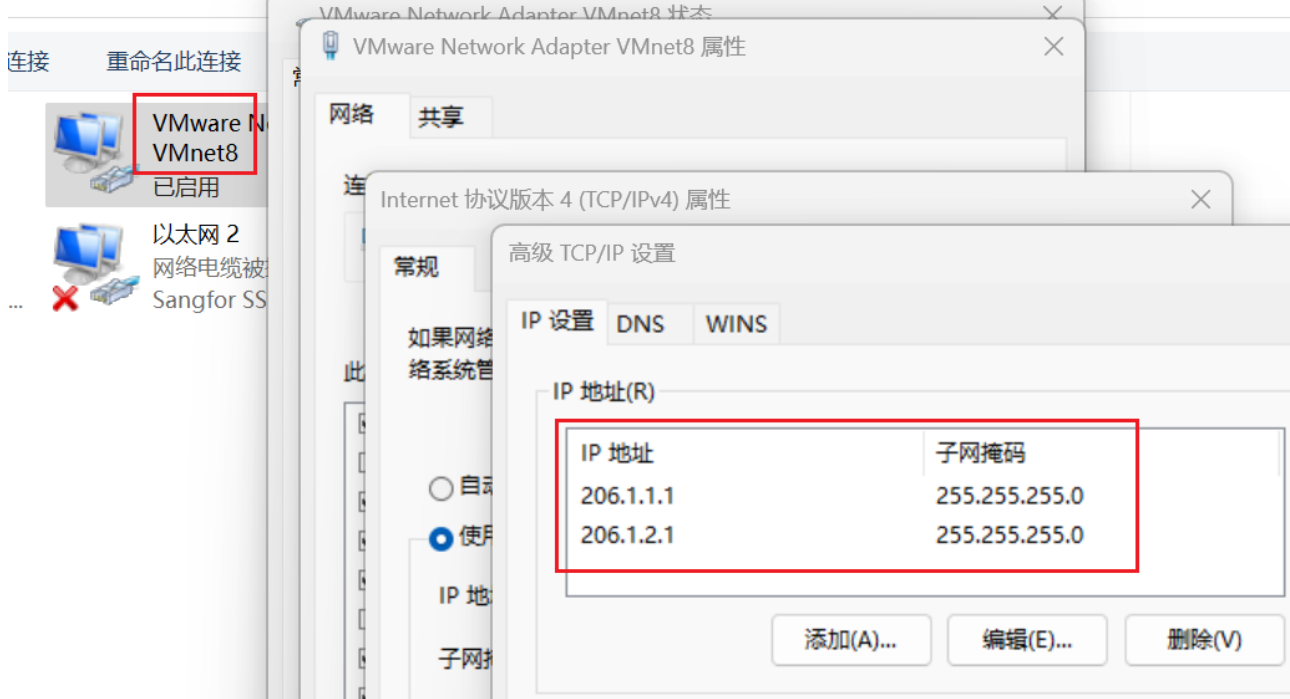
实验运行环境

实验环境拓扑图:







- 1. 路由程序中npcap的配置与前面编程实验相同，以及关闭防火墙等操作，我这里就不过多赘述。
- 2. 配置路由器A的VMnet8双网卡。

网络和 Internet > 网络连接



3. 打开事先配置好的3台虚拟机（分别是主机A， 路由器B， 主机B）。

名称	修改日期	类型
 Windows Server 2003 Enterprise Edition_1	2024/12/5 10:53	文件夹
 Windows Server 2003 Enterprise Edition_2	2024/12/4 22:54	文件夹
 Windows Server 2003 Enterprise Edition_3	2024/12/5 10:53	文件夹
 Windows Server 2003 Enterprise Edition_4	2024/12/5 10:53	文件夹

4. 将3台虚拟机的网络适配器自定义为VMnet8。

虚拟机设置

硬件 选项

设备

 内存

1 GB

 处理器


2

 硬盘 (SCSI)

40 GB

 CD/DVD (IDE)

自动检测

 网络适配器

自定义 (VMnet8 (NAT))

 网络适配器 2


桥接模式 (自动)

 USB 控制器

存在

 声卡

自动检测

 打印机

存在

 显示器

自动检测

设备状态

☒ 已连接(C)

☒ 启动时连接(O)

网络连接

☐ 桥接模式(B): 直接连接物理网络

☐ 复制物理网络连接状态(P)

☐ NAT 模式(N): 用于共享主机的 IP 地址

☐ 仅主机模式(H): 与主机共享的专用网络

☒ 自定义(U): 特定虚拟网络

VMnet8 (NAT 模式)

☐ LAN 区段(L):

```
C:\Documents and Settings\Administrator>route ADD 206.1.1.0 MASK 255.255.255.0 206.1.2.1

C:\Documents and Settings\Administrator>route PRINT

IPv4 Route Table
=====
Interface List
=====
0xc1...00 0c 29 30 80 79 ..... 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

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

5. 添加路由器B的路由表项

6. 启动路由器A程序，选择双IP网卡，添加路由表项。

```
F:\Users\lenovo\source\netw  X  +  v

1: NAME: rpcap://\Device\NPF_{D09A091D-5ED5-4F08-A9DC-A83D8E552330}
DESCRIPTION: Network adapter 'WAN Miniport (Network Monitor)' on local host
2: NAME: rpcap://\Device\NPF_{63AF9373-18F0-4626-955C-C586B24A5415}
DESCRIPTION: Network adapter 'WAN Miniport (IPv6)' on local host
3: NAME: rpcap://\Device\NPF_{9C386B35-17AD-4413-A4E9-B5D6B2BDC5BC}
DESCRIPTION: Network adapter 'WAN Miniport (IP)' on local host
4: NAME: rpcap://\Device\NPF_{703E191A-7CB3-4631-A9FC-19A9521CEA31}
DESCRIPTION: Network adapter 'Bluetooth Device (Personal Area Network)' on local host
IP地址: 169.254.129.68 子网掩码: 255.255.0.0 广播地址: 169.254.255.255

5: NAME: rpcap://\Device\NPF_{0DE51C22-4780-40CE-B634-6D6060D990B2}
DESCRIPTION: Network adapter 'MediaTek Wi-Fi 6 MT7921 Wireless LAN Card' on local host
IP地址: 10.130.84.221 子网掩码: 255.255.128.0 广播地址: 10.130.127.255

6: NAME: rpcap://\Device\NPF_{1AD4FD6C-8C85-4ED4-B59F-0387557785D1}
DESCRIPTION: Network adapter 'VMware Virtual Ethernet Adapter for VMnet8' on local host
IP地址: 206.1.2.1 子网掩码: 255.255.255.0 广播地址: 206.1.2.255

IP地址: 206.1.1.1 子网掩码: 255.255.255.0 广播地址: 206.1.1.255

7: NAME: rpcap://\Device\NPF_{4FD20337-BA64-4663-BF7E-BED6F37554AB}
DESCRIPTION: Network adapter 'VMware Virtual Ethernet Adapter for VMnet1' on local host
IP地址: 192.168.238.1 子网掩码: 255.255.255.0 广播地址: 192.168.238.255

8: NAME: rpcap://\Device\NPF_{BA5221BB-5A38-451A-A8C9-A36F7514DD1C}
DESCRIPTION: Network adapter 'Microsoft Wi-Fi Direct Virtual Adapter #2' on local host
IP地址: 169.254.14.112 子网掩码: 255.255.0.0 广播地址: 169.254.255.255

9: NAME: rpcap://\Device\NPF_{C6E675B7-8065-4AC5-AABD-200E42E46A3D}
DESCRIPTION: Network adapter 'Microsoft Wi-Fi Direct Virtual Adapter' on local host
IP地址: 169.254.166.116 子网掩码: 255.255.0.0 广播地址: 169.254.255.255

10: NAME: rpcap://\Device\NPF_{Loopback}
DESCRIPTION: Network adapter 'Adapter for loopback traffic capture' on local host
IP地址: 127.0.0.1 子网掩码: 255.0.0.0 广播地址: 0.0.0.0

11: NAME: rpcap://\Device\NPF_{D2187D0F-DB20-4C18-8699-5073E9CC5B2A}
DESCRIPTION: Network adapter 'Sangfor SSL VPN CS Support System VNIC' on local host
IP地址: 169.254.221.237 子网掩码: 255.255.0.0 广播地址: 169.254.255.255

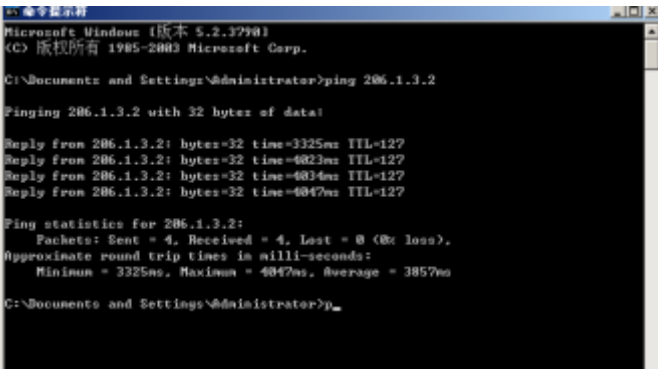
12: NAME: rpcap://\Device\NPF_{E8D78A5E-F933-4C14-922C-BA6C30998832}
DESCRIPTION: Network adapter 'TAP-Windows Adapter V9' on local host
IP地址: 169.254.113.85 子网掩码: 255.255.0.0 广播地址: 169.254.255.255

共有12个网络接口卡
打开第几个网络接口卡?
```

```
11: NAME: rpcap://\Device\NPF_{D2187D0F-DB20-4C18-8699-5073E9CC5B2A}
DESCRIPTION: Network adapter 'Sangfor SSL VPN CS Support System VNIC' on local host
IP地址: 169.254.221.237 子网掩码: 255.255.0.0 广播地址: 169.254.255.255

12: NAME: rpcap://\Device\NPF_{E8D78A5E-F933-4C14-922C-BA6C30998832}
DESCRIPTION: Network adapter 'TAP-Windows Adapter V9' on local host
IP地址: 169.254.113.85 子网掩码: 255.255.0.0 广播地址: 169.254.255.255

共有12个网络接口卡
打开第几个网络接口卡?
6
IP地址: 206.1.2.1 子网掩码: 255.255.255.0 广播地址: 206.1.2.255
IP地址: 206.1.1.1 子网掩码: 255.255.255.0 广播地址: 206.1.1.255
你想要改变路由表吗? y or n
y
添加还是删除? a or d
a
请输入路由表, 输入顺序为: 目的网络号, 子网掩码, 下一跳步
206.1.3.0
255.255.255.0
206.1.2.2
继续添加? y or n
n
网络掩码      目的网络      下一站路由
0      255.255.255.0      206.1.2.0      0.0.0.0
网络掩码      目的网络      下一站路由
1      255.255.255.0      206.1.1.0      0.0.0.0
网络掩码      目的网络      下一站路由
2      255.255.255.0      206.1.3.0      206.1.2.2
n
路由表:
网络掩码      目的网络      下一站路由
0      255.255.255.0      206.1.2.0      0.0.0.0
网络掩码      目的网络      下一站路由
1      255.255.255.0      206.1.1.0      0.0.0.0
网络掩码      目的网络      下一站路由
2      255.255.255.0      206.1.3.0      206.1.2.2
向自己发包成功
本机IP:206.1.1.1
本机MAC:0-50-56-c0-0-8
```



7. 主机Aping主机B, 验证路由器功能及正确性。

8. 查看数据报获取和转发过程的日志输出。

```
F:\Users\lenovo\source\netw  X  +  v
    网络掩码      目的网络      下一站路由
0    255.255.255.0  206.1.2.0      0.0.0.0
    网络掩码      目的网络      下一站路由
1    255.255.255.0  206.1.1.0      0.0.0.0
    网络掩码      目的网络      下一站路由
2    255.255.255.0  206.1.3.0      206.1.2.2
向自己发包成功
本机IP:206.1.1.1
本机MAC:0-50-56-c0-0-8

正为该包进行转发
version=4
headlen=5
tos=0
totallen=60
id=0x212
ttl=128
protocol=1
数据包源地址: 206.1.1.2
数据包目的地址: 206.1.3.2
路由表长度为: 3
nextIP:206.1.2.2
sendIP:206.1.1.1
recvIP:206.1.2.2
发包成功
NextIP的MAC地址:0-c-29-30-80-79
NextIP的IP:206.1.2.2
转发成功
源IP地址: 206.1.1.2      目的IP地址: 206.1.3.2
目的mac: 0-c-29-30-80-79      源mac: 0-50-56-c0-0-8

正为该包进行转发
version=4
headlen=5
tos=0
totallen=60
id=0x113
ttl=127
protocol=1
数据包源地址: 206.1.3.2
数据包目的地址: 206.1.1.2
路由表长度为: 3
nextIP:0.0.0.0
sendIP:206.1.1.1
recvIP:206.1.1.2
发包成功
NextIP的MAC地址:0-c-29-da-62-f0
NextIP的IP:206.1.1.2
转发成功
源IP地址: 206.1.3.2      目的IP地址: 206.1.1.2
目的mac: 0-c-29-da-62-f0      源mac: 0-50-56-c0-0-8
```

四、实验中遇到的问题

(1) 本次实验遇到的最主要问题是主机A ping 主机B得到的TTL值为127，但应该为126，按理说我需要在转发包后将TTL减1，但我这样操作之后主机A就无法ping主机B了，具体的原因我也不是特别清楚，查阅资料也没有得到可靠的信息。

```
// 到目前为止知道了：自己的IP:my_IP;自己的MAC: my_mac[6]; 要转发的包
// 转发包
IPData_t *TempIP;
TempIP = (IPData_t *)sendAllPacket;
// TempIP->IPHeader.TTL -= 1; //TTL-1 只要改TTL就出错?
for (int t = 0; t < 6; t++)
{
    TempIP->FrameHeader.DesMac[t] = its_mac[t]; // 目的mac地址
    TempIP->FrameHeader.SrcMac[t] = my_mac[t];
}
```

(2) 起初我使用机房电脑进行实验，相关配置已经全都配置好，包括机房电脑上的npcap配置，但是在添加路由器B的路由表项时，由于有其他同学已经设置了IP，可能是因为IP的冲突，导致添加路由表项失败，所以我就在虚拟机上完成了此次实验。

(3) 有时实验失败时，过程中通过4台设备互相ping测试进行问题的定位，可以确定是哪两台设备间出了问题。

五、实验感悟

这次实验让我深刻理解了路由器工作的内部机制，尤其是IP数据报如何在网络中被处理和转发的过程。通过亲手构建一个简易的路由器模型，我不仅掌握了网络编程的基础技能，还学到了关于网络协议、路由算法等方面的知识。此外，实验过程中的调试和问题解决锻炼了我的逻辑思维能力，也提高了我对复杂系统的分析和理解能力。最重要的是，我意识到理论学习和实践操作相结合的重要性，这为我未来继续学习计算机网络打下了坚实的基础。