南　　开　　大　　学

网络空间安全学院

网络技术与应用实验报告

实验 5：简单路由器程序的设计

姓名:申杰予

学号:2111030

专业 :物联网工程

2023 年 12 月 23 日

目录

一、 实验要求 **2**

二、 实验过程 **2**

（一） 实验拓扑图 2

（二） 实验流程图 2

（三） 具体实现 4

1. 打开网卡并获取 MAC 4
2. 路由表项的操作 5
3. IP 包的处理 7
4. ARP 包的处理 9
5. 日志的输出 10

三、 实验结果 **11**

四、 实验总结 **12**

1. 遇到的问题 12
2. 感悟 12

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。

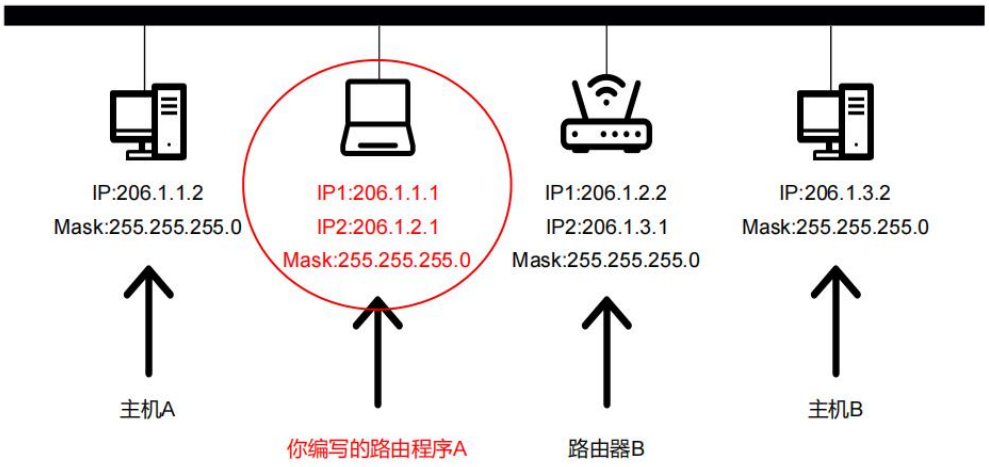
3. 需要给出路由表的手工插入、删除方法。

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、 实验过程

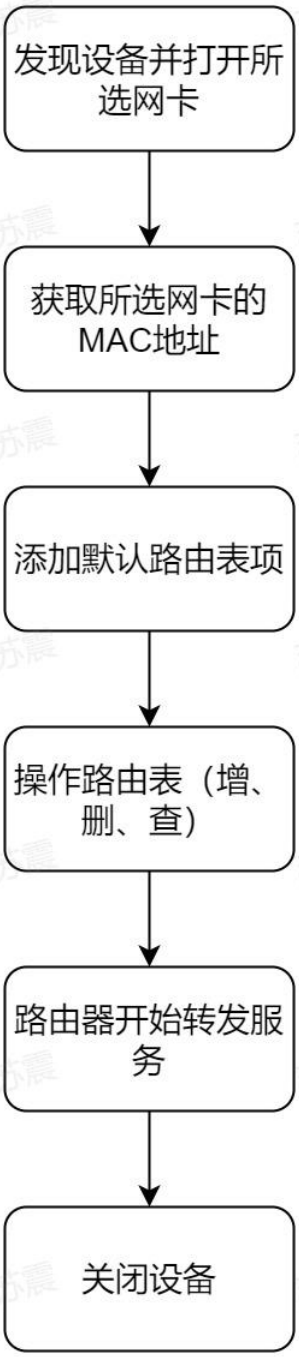
（一） 实验拓扑图



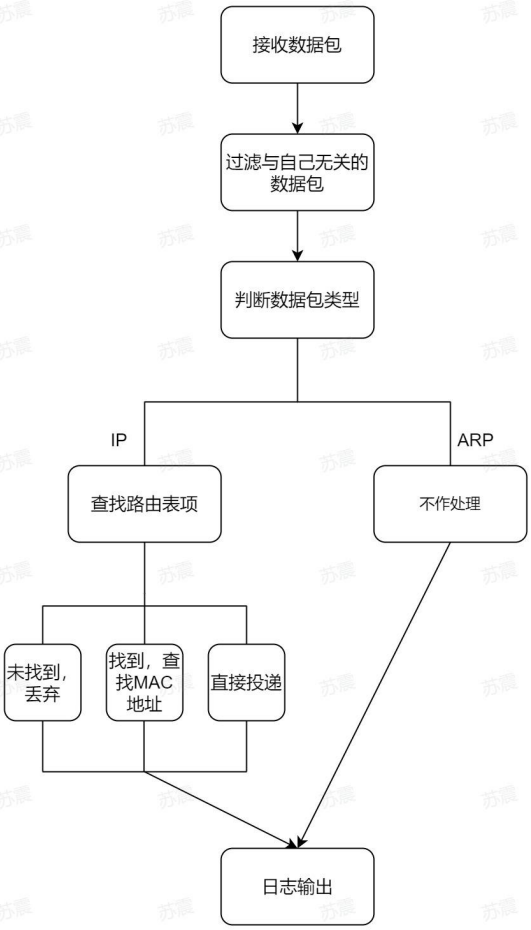
在本次实验中，我们需要部署四台虚拟机，并且在每台虚拟机上设置IP 地址（主机要设计默认网关）。而我们所写的路由器程序，则需布置在第二个虚拟机上，使第二台虚拟机起到路由器的作用（不需要开启**Routing and Remote Access** 服务），而第三台虚拟机则需开启**Routing and Remote Access** 服务。由于我所写代码中包括了高版本的函数，所下发的虚拟机版本库中不包含这些函数，所以**我在微软官方下载了Win10 映像，以兼容我所写的代码**。同时为防止程序出现问题，我将四台虚拟机的防火墙服务均关闭。

（二） 实验流程图

本次实验大体流程图如下：



而转发服务的具体流程图如下：



（三） 具体实现

在实现部分，我维护了两个表，一个是实验所要求的**路由表**，另一个则是 **IP-MAC 表**。维护 **IP-MAC 表**的原因是：如果我们曾经给这个 MAC地址发送给数据包，第二次发送就不需要再获取一遍 MAC 地址了，因为获取 MAC 地址的过程比较费时，需要构造数据包发送并等待回收，维护这样一个表大大提高了时间效率。

由于代码部分比较冗长，所以这里我只贴重点的代码方便讲解。

1. 打开网卡并获取 MAC

打开网卡并获取 MAC

```
//捕获回复包，获取选取网卡的MAC地址
while (1) {
        pcap__pkthdr* header;
        const u__char* content;
        int result = pcap__next__ex(hand, &header, &content);
        if (result == 1) {
                //强制类型转换
                First__Packet__Reply = (ARP__Packet*)content;
                if (First__Packet__Reply->RecvIP == inet__addr("192.192.192.192"
```

```
                    )) {     //ARP类型
                        bool compare__flag = true;
                        for (int i = 0; i < 6; i++)
                            if (First__Packet__Reply->FrameHeader.DesMAC[i]
                                != First__Packet.FrameHeader.SrcMAC[i]) {
                                compare__flag = false;
                                break;
                            }

                        //捕捉到的包的源MAC应是虚拟主机的MAC地址
                        if (compare__flag) {
                            for (int i = 0; i < 6; i++)
                                local__MAC[i] = First__Packet__Reply->
                                    FrameHeader.SrcMAC[i];

                            cout << "MAC地址为:";
                            for (int i = 0; i < 5; i++) {
                                printf("%02x:", First__Packet__Reply->
                                    FrameHeader.SrcMAC[i]);
                            }
                            printf("%02x", First__Packet__Reply->
                                FrameHeader.SrcMAC[5]);
                            break;
                        }
                    }
                }
            }
```

我们首先模拟了一个 ip 为 **192.192.192.192** 的虚拟主机向所选网卡发送 ARP 包，随后用捕获本机网卡的应答包，从中得到 MAC 地址。

2. 路由表项的操作

首先我们看一下路由表项的数据结构：

路由表项数据结构

```
//路由表项
struct Router__Table__Item {
        u__long DesIP;      // 目的IP
        u__long Netmask;    //子网掩码
        u__long Nextjump;   //下一跳步
        int is__default;
        Router__Table__Item() {
            memset(this, 0, sizeof(*this));
        }

        Router__Table__Item(u__long ip, u__long mask, u__long nextjump, int
            Is__default) {
            this->DesIP = ip;
```

```
        this->Netmask = mask ;
        this->Nextjump = nextjump ;
        this->is__default = Is__default ;
    }
};
```

路由表项由典型的三元组组成，并且额外增加了一个变量 **is__default**，是为了判断某项路由表项是否是默认的路由表项，如果是默认的路由表项，则无法删除。

随后我们看对路由表操作的代码：

路由表的操作

```
//添加路由表项
void add__item(u__long ip, u__long mask, u__long jump) {
    Router__Table__Item item(ip, mask, jump, 0);
    router__table.push__back(item);
    sort(router__table.begin(), router__table.end(), cmp);
    return ;
}


//删除路由表项
void delete__item(u__long ip, u__long mask, u__long jump) {
    for (int i = 0; i < router__table.size(); i++) {
        if (router__table[i].DesIP == ip && router__table[i].Netmask ==
            mask && router__table[i].Nextjump == jump) {
            if (router__table[i].is__default == 0) {
                router__table.erase(router__table.begin() + i);
            }
            else {
                cout << "目标为路由表项, 删除失败!" << endl;
                return ;
            }
        }
        else {
            cout << "没有找到相应的路由表项, 删除失败!" << endl;
            return ;
        }
    }
}


//输出路由表
void show__item() {
    cout << setw(15) << "目的IP" << setw(15) << "子网掩码" << setw(15) <<
        "下一跳步" << endl;
    cout << "
        ========================================================================
        " << endl;
```

```
        for (int i = 0; i < router__table.size(); i++)
                cout << setw(15) << getIP(router__table[i].DesIP) << setw(15)
                    << getIP(router__table[i].Netmask) << setw(15) << getIP(
                    router__table[i].Nextjump) << endl;
        return;
}


//查找路由表
int find__item(u__long dstip) {
        for (int i = 0; i < router__table.size(); i++) {
                if ((dstip & router__table[i].Netmask) == router__table[i].
                    DesIP) {
                        if (router__table[i].Nextjump != 0)
                                return router__table[i].Nextjump;
                        else
                                return -1;      //直接投递
                }
        }

        //没有查到
        return 0;
}
```

我的路由表是基于 STL 库的 vector 来实现的，非常方便，效率也很高。这里对路由表的操作有四个：

- 路由表的插入：普通路由表项的删除会用到此函数。在此函数中，添加的路由表项 **is__default** 项为 0，代表并非默认路由表项，可以删除。添加的方法就是将新的路由表项 **push__back** 到已有的路由表中。重点在于，**每当我们添加路由表项时，应该按照子网掩码从大到小排序，这样符合最长匹配原则，即找到的第一个符合条件的路由表项的子网掩码一定是最大的。**而从大到小排序的原因为：**我们在查找路由表是从前向后一个个遍历查找的**。
- 路由表的删除：路由表的删除也是遍历，如果找到符合条件的项且不为默认路由表项，则可以删除，否则不可以删除。
- 路由表的输出：在这里我用到了**setw()** 函数，便于格式化输出，使得输出的路由表简洁美观。
- 路由表的查找：在查找中，我们从 vector 中从前向后遍历，由于我们的路由表项是按照子网掩码大小排序，所以这样找到的路由表项符合最长匹配原则。当目标 ip 与路由表项相与时，如果下一跳步不为 0，证明不算直接投递，则返回下一跳的IP；如果下一跳步为 0，则直接投递（**返回值为-1**）；如果找不到，则函数返回值为 0.

3. IP 包的处理

IP 包的处理

```
//数据包为IP类型
if (ntohs(et__header->FrameType) == 0x0800) {
```

```
IP__Packet* ip__packet = new IP__Packet;
ip__packet = (IP__Packet*)content;
if (ip__packet->IPHeader.SrcIP != inet__addr("206.1.1.2") && ip__packet
    ->IPHeader.SrcIP != inet__addr("206.1.3.2") && ip__packet->IPHeader
    .DstIP != inet__addr("206.1.3.2") && ip__packet->IPHeader.DstIP !=
    inet__addr("206.1.1.2"))
        continue;
i++;


//检查校验和
if (!verify__checksum(&(ip__packet->IPHeader))) {
        cout << "校验和错误!" << endl;
        continue;
}

print__iplog(ip__packet);

int result = find__item(ip__packet->IPHeader.DstIP);

//查不到
if (result == 0) {
        cout << "查询不到对应的路由表项!" << endl;
        continue;
}

//直接投递
else if (result == -1) {
        cout << "直接投递!" << endl;
        u__char* dstMAC = find__mac(ip__packet->IPHeader.DstIP);

        //查不到MAC地址
        if (dstMAC == 0) {
                cout << "表中查询不到MAC地址, 开始获取" << endl;
                dstMAC = get__mac(ip__packet->IPHeader.DstIP);
                if (dstMAC == 0) {
                        cout << "获取MAC失败!" << endl;
                        continue;
                }
        }

        //查到了MAC地址
        for (int i = 0; i < 6; i++)
                ip__packet->FrameHeader.DesMAC[i] = dstMAC[i];
        for (int i = 0; i < 6; i++)
                ip__packet->FrameHeader.SrcMAC[i] = local__MAC[i];

        //计算校验和
```

```
48                  calculate__checksum(&ip__packet->IPHeader);
49
50                  if (pcap__sendpacket(hand, (u__char*)ip__packet, header->len) !=
                        0)
51                      cout << "转发失败!" << endl;
52                  else
53                      cout << "转发成功!" << endl;
54                  print__iplog(ip__packet);
55          }
56
57          //查到对应表项
58          else {
59                  cout << "查到对应路由表项!" << endl;
60                  u__char* dstMAC = find__mac(ip__packet->IPHeader.DstIP);
61                  if (dstMAC == 0) {
62                      cout << "表中查询不到MAC地址, 开始获取" << endl;
63                      dstMAC = get__mac(ip__packet->IPHeader.DstIP);
64                      if (dstMAC == 0) {
65                          cout << "获取MAC失败!" << endl;
66                          continue;
67                      }
68                  }
69
70                  for (int i = 0; i < 6; i++)
71                      ip__packet->FrameHeader.DesMAC[i] = dstMAC[i];
72                  for (int i = 0; i < 6; i++)
73                      ip__packet->FrameHeader.SrcMAC[i] = local__MAC[i];
74
75                  if (pcap__sendpacket(hand, (u__char*)ip__packet, header->len) !=
                        0)
76                      cout << "转发失败!" << endl;
77                  else
78                      cout << "转发成功!" << endl;
79                  print__iplog(ip__packet);
80          }
81  }
```

如果判断捕获到的包是 IP 数据包，我们则先检查校验和，随后进行 IP 包日志的输出。接下来我们去查询路由表项，如果找到了或者是直接投递，接下来我们就去**IP-MAC 表**中找目的 IP 的 MAC 地址，如果存在，则直接取 MAC 地址，大大节省了时间，提高了效率。如果找不到，则获取 MAC 地址（**原理与获取本地MAC 是一致的**）。而找不到对应的路由表项，则无法转发此数据包。

4. ARP 包的处理

ARP 包的处理

```
//数据包类型为ARP类型
1   if (ntohs(et__header->FrameType) == 0x0806) {
2
```

```
        ARP__Packet* arp__packet = new ARP__Packet;
        arp__packet = (ARP__Packet*)content;
        if (arp__packet->SendIP != inet__addr("206.1.1.2") && arp__packet->
            SendIP != inet__addr("206.1.3.2") && arp__packet->RecvIP !=
            inet__addr("206.1.3.2") && arp__packet->RecvIP != inet__addr("
            206.1.1.2"))
                continue;
        i++;
        print__arplog(arp__packet);
}
```

由于 ARP 包不能穿透，所以这里我们只进行 ARP 包日志的输出。

5. 日志的输出

日志输出

```
//IP数据报日志输出
void print__iplog(IP__Packet* ip__packet) {
        get__time(t);
        cout << "
            ================================================================================
            " << endl;
        cout << "[log]" << t << endl;
        cout << "数据包类型:" << "IP" << endl;
        cout << "源MAC:";
        for (int i = 0; i < 5; i++) {
                printf("%02x:", ip__packet->FrameHeader.SrcMAC[i]);
        }
        printf("%02x", ip__packet->FrameHeader.SrcMAC[5]);

        cout << endl;

        cout << "目的MAC:";
        for (int i = 0; i < 5; i++) {
                printf("%02x:", ip__packet->FrameHeader.DesMAC[i]);
        }
        printf("%02x", ip__packet->FrameHeader.DesMAC[5]);

        cout << endl;

        cout << "源IP:";
        cout << getIP(ip__packet->IPHeader.SrcIP) << endl;
        cout << "目的IP:";
        cout << getIP(ip__packet->IPHeader.DstIP) << endl;
        cout << "
            ================================================================================
            " << endl;
        return;
```

```
}


//ARP数据报日志输出
void print__arplog(ARP__Packet* arp__packet) {
        get__time(t);
        cout << "
            ================================================================================
            " << endl;
        cout << "[log]" << t << endl;
        cout << "数据包类型:" << "ARP" << endl;
        cout << "源MAC:";
        for (int i = 0; i < 5; i++) {
                printf("%02x:", arp__packet->FrameHeader.SrcMAC[i]);
        }
        printf("%02x", arp__packet->FrameHeader.SrcMAC[5]);

        cout << endl;

        cout << "目的MAC:";
        for (int i = 0; i < 5; i++) {
                printf("%02x:", arp__packet->FrameHeader.DesMAC[i]);
        }
        printf("%02x", arp__packet->FrameHeader.DesMAC[5]);

        cout << endl;

        cout << "源IP:";
        cout << getIP(arp__packet->SendIP) << endl;
        cout << "目的IP:";
        cout << getIP(arp__packet->RecvIP) << endl;
        cout << "
            ================================================================================
            " << endl;
        return;
}
```

其实 IP 包和 ARP 包的日志输出很相似，都是输出了源 MAC 和目的 MAC 地址以及源 IP 和目的 IP 地址。日志输出的目的是：便于我们观察数据包流动的过程，以便于我们更好地理解整个路由转发过程。

三、 实验结果

在实现代码后，我们通过测试来验证路由程序的正确性。在这里，我们用主机 A 去 ping 主机 B，得到的结果如下图：

可以看到，主机 A 是可以 ping 通主机 B 的。而第一次 ping 通时有两个包会丢失，而第二次 ping 没有包丢失，这是因为第一次并没有得到 MAC 地址以及对应的路由表项，而在第一次ping 通时，MAC 地址和路由表项会存入已有的表中，因此不会出现超时，所有包也不会丢失。日志输出这里就不作展示了。

四、 实验总结

1. 遇到的问题

在这次实验中代码部分其实没有遇到什么问题，因为许多代码在之前的实验都已经实现过，这次主要是将他们拼凑在一起形成一个整体。而问题在于测试部分。首先我用的某些函数版本比较高，在助教所发的 2003 版虚拟机无法运行，因为库里不包括相应的函数，所以我选择了使用4 台 Win10 的虚拟机。其次就是虚拟机卡顿问题，当我开 4 台虚拟机时，我的电脑整个会卡死，黑屏，或是鼠标动不了，重启了好多次，搞了一天最后才成功，大费周折。

2. 感悟

由于这次作业是大作业，所以要实现的代码量其实还不小，整个项目是一个大项目。在之前写稍微难一些的作业时，我总是会写完后再慢慢 debug，直到程序可以跑通。而这次实验我强迫自己每写完一个函数或者是完成一个部分，就要对相应的部分立即 debug，因为代码量过大，再去程序中一行行寻找 bug 是一件很困难的事儿，所以这次我也做出了改变，也深深感到了这种方法的优点。也是课程的最后了，想说句：助教辛苦，老师辛苦，谢谢你们！