

# 实验5：简单路由器程序的设计

学号 2012522

姓名 郭坤昌

年级 2020

专业 计算机科学与技术

2022 年 12月22日

## 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。

## 实验准备

- 网络拓扑图



- 虚拟机配置

编号	IP	子网掩码	说明
----	----	------	----

编号	IP	子网掩码	说明
1	206.1.1.2	255.255.255.0	主机A
2	206.1.1.1,206.1.2.1	255.255.255.0	路由器A
3	206.1.2.2,206.1.3.1	255.255.255.0	路由器B
4	206.1.3.2	255.255.255.0	主机B

其中，1号设备和4号设备为终端设备，2号设备为需要运行路由程序的设备，3号设备为另一台路由器。所有4台设备的IP地址已经全部分配好，所有4台设备已经安装好x86的VC++运行时环境。其中3号设备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
```

即开启四台虚拟机以后，需要：

1. 为3号设备手动添加路由表项。
2. 在2号设备上运行路由程序。

环境即搭建完毕。

- 编程环境

使用VC6进行编程和配置

## 实验过程

### 路由器工作流程

路由器使用混杂模式捕获所有经过网卡的数据包，处理基于IP协议的数据包，这里只考虑ICMP报文，在完成前期打开网卡，获取本机mac地址后，工作过程如下：

#### 1. 接收线程——过滤

##### 1.1 过滤本机发送的数据包—

##### 1.2 对于ICMP数据包

- 过滤目的地址为本机IP地址（不需要转发）的数据包
- 过滤目的mac地址不为本机mac地址的数据包（不经过路由程序的数据包，这里不考虑组播和广播的处理，详见第三节问题处理部分）
- 过滤校验和字段出错的数据包
- 将满足条件的ICMP数据包加入缓冲区等待转发

##### 1.3 对于ARP数据包

- 过滤ARP请求数据包。原因如下：向本机进行ARP请求的数据包由操作系统自动进行处理；若跨网段进行ARP请求，则不予转发
- 过滤不在同一网段下的ARP相应数据包。由于使用虚拟机，仍能在混杂模式下捕获到与接口不在同一网段下的ARP响应数据包，这是不合理的，因此还必须进行过滤。

- 将ARP相应数据包中的源IP地址和源mac地址的映射关系添加到ARP缓存中，并记录添加的时间
- 2. 转发线程——维护缓冲区，转发数据包
  - 2.1 遍历转发缓存，对每一个数据包，若TTL为0，丢弃并回传ICMP差错报文，否则进入下一步
  - 2.2 检查数据包是否超时（缓存清理部分），若未超时进入下一步，否则移出缓冲区
  - 2.3 检查目的IP地址是否与网卡接口在同一网段，若是，则转发IP为目的IP，进入2.5步，否则进入2.4步
  - 2.4 在路由表中查找目的IP对应的下一跳地址。由于默认网关的设置，这里一定能有对应路由表项的下一跳地址。
  - 2.5 在ARP表中查找转发IP对应的mac地址，若存在，则修改数据包以太层的目的mac地址为查找到的mac地址，修改TTL-1，重新计算校验和，进行转发，否则广播ARP请求
- 3. 缓存清理
  - 3.1 若数据包超过最长缓存时间，将其移出缓冲区。在转发遍历时实现
  - 3.2 定时检查ARP缓冲区，若超过了老化时间，则将其删除
- 4. 指令线程——手动维护路由表和ARP缓存表
  - 4.1 路由表的增删改
  - 4.2 ARP表的查询

## 详细代码

### 获取设备信息并打开

程序运行在具有双网卡的虚拟机设备上，在打开网卡后，需要频繁使用打开设备的两个IP地址和对应的子网掩码，以及打开网卡的句柄，因此这里使用Device类管理设备信息，使用DeviceManager类管理设备。主要成员变量及方法如下：

```
1  class Device {
2  private:
3      string name;
4      string description;
5      DWORD ip[2];
6      DWORD subnetMask[2];
7      BYTE mac[6];
8      friend class DeviceManager;
9  };
10 class DeviceManager {
11 private:
12     u_int deviceNum;
13     Device *deviceList;    // 设备列表
14     Device *openDevice;    // 打开设备指针
15     pcap_t *openHandle;    // 打开设备句柄
16 public:
17     void findDevices();    // 查找所有网卡, 获取设备信息
18     void selDevice();      // 选择并打开网卡
19     void setMac(BYTE *mac, Device *device);    // 设置特定设备MAC地址
20     DWORD findItf(DWORD ip); // 根据IP地址, 查看是否与打开设备在同一网段,
    并返回对应接口IP地址
```

```
21 | };
```

获取设备名和描述；遍历pcap\_addr\_t链表，获取设备的IP的子网掩码。

```
1 void DeviceManager::findDevices() {
2     ...
3     deviceList = new Device[deviceNum];
4     for (i = 0, d = alldevs; d != NULL; d = d->next, i++) {
5         deviceList[i].name = string(d->name); // 获取设备名
6         deviceList[i].description = string(d->description); // 获取
设备描述
7         for (j = 0, a = d->addresses; j < 2 && a != NULL; a = a-
>next) { // 获取设备IP地址和子网掩码
8             if (a->addr->sa_family == AF_INET) {
9                 deviceList[i].ip[j] = inet_addr(inet_ntoa(((struct
sockaddr_in *) a->addr)->sin_addr));
10                deviceList[i].subnetMask[j] =
inet_addr(inet_ntoa(((struct sockaddr_in *) a->netmask)-
>sin_addr));
11                j++;
12            }
13        }
14    }
15    ...
16 }
```

输入设备序号，选中并打开设备，同时将打开设备指针指向打开设备的信息。

```
1 void DeviceManager::selDevice() {
2     ...
3     openDevice = &deviceList[i];
4     if ((openHandle = pcap_open(openDevice->name.c_str(), 65536,
PCAP_OPENFLAG_PROMISCUOUS, 1000, NULL, errbuf)) == NULL) ...
5 }
```

该虚拟机仅有一个网卡，获取信息并打印如下：

```
【SUC】 Find Devices Success! Devices:
Device Num: 1
Device 1:
Name: \Device\NPF_{3AC00148-1BFF-47AF-8441-F1D14A619031}
Description: Intel(R) PRO/1000 MT Network Connection
IP Addr1: 206.1.2.1      Subnet Mask: 255.255.255.0
IP Addr2: 206.1.1.1     Subnet Mask: 255.255.255.0
```

## 获取设备mac地址

实验中需要获取本机mac地址和其他主机的mac地址，用以进行包过滤和数据包转发。获取本机mac地址和其他主机mac地址的策略不同：

1. 获取本机mac地址：使用虚构的源IP地址与全0的源mac地址，以本机双IP中任一IP为目的IP，广播ARP请求。设置包过滤条件：捕获包源IP为广播包目的IP，捕获包目的IP为虚构IP、目的mac地址为全0。该数据包的源mac地址即为本机IP对应的mac地址。

2. 获取其他主机的mac地址：首先检查目标主机IP地址是否与打开网卡的双接口在同一网段，若不是，则无法获取。从本机mac地址和与目的IP所处同一网段的接口IP地址广播ARP请求。在数据包捕获线程中，解析ARP响应数据包，将IP与mac地址映射关系加入ARP缓存中。包过滤规则接下来详细介绍。

获取本机网络mac地址运行结果：

```
【SUC】 Get IP-MAC map successfully. Open device info :
Name: \Device\NPF_{3AC00148-1BFF-47AF-8441-F1D14A619031}
Description: Intel(R) PRO/1000 MT Network Connection
IP Addr1: 206.1.2.1      Subnet Mask: 255.255.255.0
IP Addr2: 206.1.1.1      Subnet Mask: 255.255.255.0
MAC Addr: 00-0C-29-30-19-74
```

获取其他主机mac地址运行结果：

```
ARPTable:
IP Address      Mac Address      Time
206.1.2.2       00-0C-29-8F-4C-DF 11:10:53
206.1.1.2       00-0C-29-25-B7-8A 11:10:58
```

## 路由表和ARP缓存数据结构

路由表和ARP缓存使用双向链表进行维护，实现其增删查的功能。以下以路由表为例详细介绍。路由表项和路由表数据结构如下：

```
1  class RoutingEntry {
2  private:
3      DWORD dest;          // 目的地址
4      DWORD netmask;       // 子网掩码
5      DWORD gw;            // 网关地址
6      DWORD itf;           // 出接口
7      RoutingEntry *prev;
8      RoutingEntry *next;
9      friend class RoutingTable;
10 };
11 class RoutingTable {
12 private:
13     Device *openDevice;
14     RoutingEntry *head;
15     RoutingEntry *tail;
16     u_int size;
17 public:
18     RoutingTable(Device *openDevice);
19     void add(DWORD dest, DWORD netmask, DWORD gw);
20     void del(RoutingEntry *routingEntry);
21     RoutingEntry *lookup(DWORD dest);
22     RoutingEntry *lookup(char *dest);
23 };
```

路由表的添加。首先检查路由表项是否存在，避免重复添加。接着检查添加的下一跳地址是否为本地接口可达（即与打开设备的接口在同一网段），以此设定接口地址（接口字段其实没有发挥作用，但仍然表示该接口对网关的可达性），并最终将表项添加到路由表中。

```

1 void RoutingTable::add(DWORD dest, DWORD netmask, DWORD gw) {
2     RoutingEntry *routingEntry;
3     DWORD itf;
4
5     // 避免重复添加
6     if((routingEntry = lookup(dest)) != NULL && (routingEntry->netmask != 0)) {
7         return;
8     }
9     switch(netmask) {
10    case 0:
11        // 子网掩码为0, 添加默认路由, 检查接口到下一跳地址的可达性
12        if ((openDevice->getIP(0) & openDevice->getSubnetMask(0))
13 == (gw & openDevice->getSubnetMask(0))) {
14            itf = openDevice->getIP(0);
15        } else if ((openDevice->getIP(1) & openDevice->getSubnetMask(1))
16 == (gw & openDevice->getSubnetMask(1))) {
17            itf = openDevice->getIP(1);
18        } else {
19            return;
20        }
21        routingEntry = new RoutingEntry(0, 0, gw, itf);
22        break;
23    default:
24        // 检查接口到下一跳地址的可达性
25        if ((openDevice->getIP(0) & openDevice->getSubnetMask(0))
26 == (gw & openDevice->getSubnetMask(0))) {
27            itf = openDevice->getIP(0);
28        } else if ((openDevice->getIP(1) & openDevice->getSubnetMask(1))
29 == (gw & openDevice->getSubnetMask(1))) {
30            itf = openDevice->getIP(1);
31        } else {
32            return;
33        }
34        routingEntry = new RoutingEntry(dest&netmask, netmask, gw,
35 itf);
36    }
37
38    if (head == NULL) {
39        head = tail = routingEntry;
40    } else {
41        tail->next = routingEntry;
42        routingEntry->prev = tail;
43        tail = routingEntry;
44    }
45    size++;
46 }

```

路由表添加的添加演示:

```

output.txt - 记事本
206.1.3.2 00-0C-29-8F-4C-DF 206.1.1.2 00-0C-29-30-
【INF】 Packet Forwarded:
DstIP DstMac TTL
206.1.1.2 00-0C-29-25-B7-8A 126
【INF】 Routing Entry Added: 206.1.3.0 255.255.255.0
【CMD】 Please input command: RoutingTable:
Destination Netmask Gateway Interface
0.0.0.0 0.0.0.0 206.1.2.2 206.1.2.1
206.1.3.0 255.255.255.0 206.1.2.2 206.1.2.1

【CMD】 Please input command: 【CMD】 Please input command: R
Destination Netmask Gateway Interface
0.0.0.0 0.0.0.0 206.1.2.2 206.1.2.1
206.1.3.0 255.255.255.0 206.1.2.2 206.1.2.1

【CMD】 Please input command: 【INF】 Delete Routing Entry: 20
【CMD】 Please input command: RoutingTable:
Destination Netmask Gateway Interface
0.0.0.0 0.0.0.0 206.1.2.2 206.1.2.1

【CMD】 Please input command: RoutingTable:
Destination Netmask Gateway Interface
0.0.0.0 0.0.0.0 206.1.2.2 206.1.2.1

【CMD】 Please input command: 【INF】 Routing Entry Added: 206.1.1.0 255.255.255.0 206.1.1.5 206.1.1.1
【CMD】 Please input command: RoutingTable:
Destination Netmask Gateway Interface
0.0.0.0 0.0.0.0 206.1.2.2 206.1.2.1
206.1.1.0 255.255.255.0 206.1.1.5 206.1.1.1

【CMD】 Please input command: 【INF】 Delete Routing Entry: 206.1.1.0 255.255.255.0 206.1.1.5 206.1.1.1
【CMD】 Please input command: RoutingTable:
Destination Netmask Gateway Interface
0.0.0.0 0.0.0.0 206.1.2.2 206.1.2.1

C:\Documents and Settings\Administrator\桌面\RouterApplication.exe
1
route add 206.1.3.5 mask 255.255.255.0 206.1.2.2
route print
route del 206.1.3.0
route print
route delete 206.1.3.0
route print
route add 206.1.1.0 mask 255.255.255.0 206.1.1.5
route print
route delete 206.1.1.0
route print

```

路由表的删除。删除时需要维护双向链表的结构。默认路由不能删除，在上层输入删除路由表项指令时，检查子网掩码，若子网掩码为0，则不能删除。

```

1 void RoutingTable::del(RoutingEntry *routingEntry) {
2     if (routingEntry->prev == NULL) {
3         head = routingEntry->next;
4     } else {
5         routingEntry->prev->next = routingEntry->next;
6     }
7     if (routingEntry->next == NULL) {
8         tail = routingEntry->prev;
9     } else {
10        routingEntry->next->prev = routingEntry->prev;
11    }
12    delete routingEntry;
13    size--;
14 }

```

路由表的删除演示：

```

output.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
206.1.3.2 00-0C-29-8F-4C-DF 206.1.1.2 00-0C-29-30-
【INF】 Packet Forwarded:
DstIP DstMac TTL
206.1.1.2 00-0C-29-25-B7-8A 126
【INF】 Routing Entry Added: 206.1.3.0 255.255.255.0
【CMD】 Please input command: RoutingTable:
Destination Netmask Gateway Interface
0.0.0.0 0.0.0.0 206.1.2.2 206.1.2.1
206.1.3.0 255.255.255.0 206.1.2.2 206.1.2.1

【CMD】 Please input command: 【CMD】 Please input command: R
Destination Netmask Gateway Interface
0.0.0.0 0.0.0.0 206.1.2.2 206.1.2.1
206.1.3.0 255.255.255.0 206.1.2.2 206.1.2.1

【CMD】 Please input command: 【INF】 Delete Routing Entry: 20
【CMD】 Please input command: RoutingTable:
Destination Netmask Gateway Interface
0.0.0.0 0.0.0.0 206.1.2.2 206.1.2.1

【CMD】 Please input command: RoutingTable:
Destination Netmask Gateway Interface
0.0.0.0 0.0.0.0 206.1.2.2 206.1.2.1

【CMD】 Please input command: 【INF】 Routing Entry Added: 206.1.1.0 255.255.255.0 206.1.1.5 206.1.1.1
【CMD】 Please input command: RoutingTable:
Destination Netmask Gateway Interface
0.0.0.0 0.0.0.0 206.1.2.2 206.1.2.1
206.1.1.0 255.255.255.0 206.1.1.5 206.1.1.1

【CMD】 Please input command: 【INF】 Delete Routing Entry: 206.1.1.0 255.255.255.0 206.1.1.5 206.1.1.1
【CMD】 Please input command: RoutingTable:
Destination Netmask Gateway Interface
0.0.0.0 0.0.0.0 206.1.2.2 206.1.2.1

```

路由表的查找。从头指针依次向后查找，维护一个始终指向匹配目的地址的最长子网掩码。将要查找的地址与表项的下一跳地址进行与运算，比较结果与目的网络，若相等，则进一步比较子网掩码长度，始终维护最长子网掩码的表项作为待返回的候选者。遍历完后，返回指向最长表项的候选者。

```

1 RoutingEntry *RoutingTable::lookup(DWORD dest) {
2     RoutingEntry *routingEntry;
3     RoutingEntry *candidate;
4     DWORD maxPrefixNetmask;
5
6     routingEntry = head;
7     if (routingEntry == NULL) {
8         cout << "【ERR】 Look up Routing Table Error: Routing table
is empty." << endl;
9         return NULL;
10    }
11    candidate = NULL;
12    maxPrefixNetmask = head->netmask;
13    while (routingEntry != NULL) {
14        if ((routingEntry->dest & routingEntry->netmask) == (dest &
routingEntry->netmask)) {
15            if (ntohl(routingEntry->netmask) >
ntohl(maxPrefixNetmask)) { // little endian in network
16                maxPrefixNetmask = routingEntry->netmask;
17                candidate = routingEntry;
18            }
19            candidate = routingEntry;
20        }
21        routingEntry = routingEntry->next;
22    }
23    if (candidate == NULL) {
24        cout << "【ERR】 Look up Routing Table Error: Routing entry
not found." << endl;
25    }
26    return candidate;

```



路由表的输出。依次显示目的网路地址、子网掩码、下一跳地址、接口地址

```

1  string RoutingEntry::toStr(bool showAttr) {
2      string str = "";
3      string temp;
4      if (showAttr) {
5          str += "Destination      Netmask      Gateway
Interface\n";
6      }          //255.255.255.255 255.255.255.255 255.255.255.255
255.255.255.255
7      temp = b2s(this->dest); temp.resize(16, ' '); str += temp;
8      temp = b2s(this->netmask); temp.resize(16, ' '); str += temp;
9      temp = b2s(this->gw); temp.resize(16, ' '); str += temp;
10     temp = b2s(this->itf); str += temp;
11     return str;
12 }

```

## 数据包捕获线程——过滤数据包

本实验暂时仅考虑ICMP echo request和reply类型数据包的获取并转发，因此对ARP数据包和ICMP数据包设置如下的包过滤条件和后续操作：

1. 过滤本机发送的数据包。若源mac地址为本机mac地址，则不用处理
2. 对于ICMP数据包。过滤目的IP地址为本机IP地址的数据包，过滤mac地址不为本机mac地址的数据包（忽略组播和广播），过滤校验和字段出错的数据包。将满足条件的数据包加入缓冲区等待转发线程进行转发
3. 对于ARP响应数据包，若本地可达，则将源IP地址和源MAC地址加入ARP缓存

```

1  DWORD WINAPI Router::rcvThrd(LPVOID lpParam) {
2      ...
3      while ((res = pcap_next_ex(router->getDeviceManager()-
>getOpenHandle(), &header, &pktData)) >= 0) {
4          if (res == 0) continue;
5          if (macCmp(router->getDeviceManager()->getOpenDevice()-
>getMac(), ((EthHeader*)pktData)->srcMac)) // 如果是本机发出的数据包则
        丢弃
6              continue;
7          switch (ntohs(((EthHeader *)pktData)->type)) {
8              case 0x0806:
9                  if ((ntohs(((ARPPkt *)pktData)->ad.op) == 0x0001) // 如果
        是ARP请求
10                     || router->getDeviceManager()->findItf(((ARPPkt
        *)pktData)->ad.srcIP) == 0) // 或者不与本机接口在同一网段，即不可达的情况，
        则丢弃
11                     continue;
12                     router->getARPTable()->add(((ARPPkt *)pktData)-
        >ad.srcIP, ((ARPPkt *)pktData)->ad.srcMac); // 添加ARP表项
13                     break;
14                     case 0x0800:

```

```

15         if ((IPpkt *)pktData)->ih.dstIP == router-
>getManager()->getOpenDevice()->getIP(0)           // 如果目的IP为
本机IP
16         || ((IPpkt *)pktData)->ih.dstIP == router-
>getManager()->getOpenDevice()->getIP(1)
17         || !macCmp(router->getManager()-
>getOpenDevice()->getMac(), ((EthHeader*)pktData)->dstMac) // 或目的
MAC不为本机
18         || isICMPCorrupted((u_short*)pktData,
sizeof(ICMPPingPkt)))           // 或ICMP校验和错误
19         continue;
// 丢弃
20         EnterCriticalSection(&router->getCS());
21         router->getPktBuf()->addBefore((ICMPPingPkt *)pktData);
22         cout << recvLog(((ICMPPingPkt *)pktData)->ih.srcIP,
((ICMPPingPkt *)pktData)->eh.srcMac, ((ICMPPingPkt *)pktData)-
>ih.dstIP, ((ICMPPingPkt *)pktData)->eh.dstMac, (int)((ICMPPingPkt
*)pktData)->ih.ttl) << endl;
23         LeaveCriticalSection(&router->getCS());
24         break;    }
25     }
26     ...
27 }

```

## 缓冲区维护线程——处理ICMP数据包

缓冲区维护即转发线程，是实验中最关键的部分。其基本思路为：遍历接收缓存，将所有需要移除的数据包移除，对剩余数据包进行尝试转发操作，数据包的移除状态也在此进行设定。接收线程和转发线程对缓冲区数据一致性的维护详见实验问题部分。

```

1  DWORD WINAPI Router::fwdThrd(LPVOID lpParam)
2  {
3      ...
4      while(true) {
5          EnterCriticalSection(&router->getCS());
6          pkt = router->getPktBuf()->getHead();
7          while(pkt != NULL) {
8              if(pkt->shouldDiscard()) {
9                  pkt = router->getPktBuf()->del(pkt);
10             }
11             else {
12                 pkt = pkt->getNext();
13             }
14         }
15         pkt = router->getPktBuf()->getHead();
16         if(pkt == NULL) {
17             LeaveCriticalSection(&router->getCS());
18             continue;
19         }
20         router->tryToFwd(router->getPktBuf()->getHead());
21         pkt = pkt->getNext();
22         LeaveCriticalSection(&router->getCS());
23         while(pkt != NULL) {
24             router->tryToFwd(pkt);

```

```

25         pkt = pkt->getNext();
26     }
27 }
28 }

```

尝试转发过程即对数据包的状态进行一系列检查，确认其是否可以转发，并获取下一跳mac地址或目的mac地址，进行转发。若数据包的生存期为0，设置其状态为可移除，直接返回；若在缓冲区中的时间超过最长允许时间，设置其状态为可移除，直接返回；若数据包的目的IP地址是打开设备本地可达的，则在ARP缓存中查找目的IP对应的mac地址。若找到则需要设置TTL-1，重新计算校验和，进行转发，否则广播对该IP地址的ARP请求。若数据包不是本地可达的，则查找路由表。在本实验中，由于除了路由器A，仅有一个路由器B，因此将默认路由设置为B，这样查找路由一定可以有表项返回。检查表项对应下一跳地址是否存在mac映射。若找到则需要设置TTL-1，重新计算校验和，进行转发，否则广播对该IP地址的ARP请求。

```

1 void Router::tryToFwd(Packet* pkt) {
2     if (pkt->getICMPpingPkt()->ih.ttl == 0) {
3         pkt->setDiscardState(true);
4         return;
5     }
6     if (time(NULL) - pkt->getTime() > pktLifetime) {
7         pkt->setDiscardState(true);
8         return;
9     }
10    if (deviceManager->findItf(pkt->getICMPpingPkt()->ih.dstIP) !=
11    0) {
12        if ((arpEntry = arpTable->lookup(pkt->getICMPpingPkt()-
13    >ih.dstIP)) == NULL) {
14            bcstARPReq(pkt->getICMPpingPkt()->ih.dstIP);
15            return;
16        }
17        dstMac = arpEntry->getMac();
18        forward(pkt->getICMPpingPkt(), dstMac);
19        cout << fwrLog(pkt->getICMPpingPkt()->ih.dstIP, dstMac,
20    (int) (pkt->getICMPpingPkt()->ih.ttl), false) << endl;
21        pkt->setDiscardState(true);
22        return;
23    }
24    if ((routingEntry = routingTable->lookup(pkt->getICMPpingPkt()-
25    >ih.dstIP)) == NULL) {
26        pkt->setDiscardState(true);
27        // TODO : send ICMP net unreachable
28        return;
29    }
30    if ((arpEntry = arpTable->lookup(routingEntry->getGw())) ==
31    NULL) {
32        bcstARPReq(routingEntry->getGw());
33        return;
34    }
35    dstMac = arpEntry->getMac();
36    forward(pkt->getICMPpingPkt(), dstMac);
37    cout << fwrLog(routingEntry->getGw(), dstMac, (int) (pkt-
38    >getICMPpingPkt()->ih.ttl)) << endl;

```

```

33 |     pkt->setDiscardState(true);
34 |     return;
35 | }

```

转发过程演示：

```

NextHop      DstMac      TTL
206.1.2.2    00-0C-29-8F-4C-DF 127
【INF】 Packet Received:
SrcIP      SrcMac      DstIP      DstMac      TTL
206.1.3.2    00-0C-29-8F-4C-DF 206.1.1.2    00-0C-29-30-19-74 127
【ERR】 ARP cache miss. IP: 206.1.1.2 不存在ARP缓存时，广播ARP请求
【ERR】 ARP cache miss. IP: 206.1.1.2
【INF】 Add ARP Entry: 206.1.1.2    00-0C-29-25-B7-8A 11:34:27 收到ARP响应，建立映射
【ERR】 tryToFwd Error: Packet TTL is 0 超时提示，设置为可移除
【INF】 Packet Received:
SrcIP      SrcMac      DstIP      DstMac      TTL
206.1.1.2    00-0C-29-25-B7-8A 206.1.3.2    00-0C-29-30-19-74 128
【INF】 Packet Forwarded:
NextHop  206.1.2.2  00-0C-29-8F-4C-DF 127  主机A->主机B
          发送至  需要经路由器A
          206.1.2.2  至路由器B
          由器B
【INF】 Packet Received:
SrcIP      SrcMac      DstIP      DstMac      TTL
206.1.3.2    00-0C-29-8F-4C-DF 206.1.1.2    00-0C-29-30-19-74 127
【INF】 Packet Forwarded:
DstIP      DstMac      TTL
206.1.1.2    00-0C-29-25-B7-8A 126  TTL-1
【INF】 Packet Received:
SrcIP      SrcMac      DstIP      DstMac      TTL
206.1.1.2    00-0C-29-25-B7-8A 206.1.3.2    00-0C-29-30-19-74 128
【INF】 Packet Forwarded:
NextHop  206.1.2.2  00-0C-29-8F-4C-DF 127
【INF】 Packet Received:
SrcIP      SrcMac      DstIP      DstMac      TTL
206.1.3.2    00-0C-29-8F-4C-DF 206.1.1.2    00-0C-29-30-19-74 127
【INF】 Packet Forwarded:
DstIP      DstMac      TTL
206.1.1.2    00-0C-29-25-B7-8A 126

```

最终主机A和主机B能相互ping通

```

C:\Documents and Settings\Administrator>ping 206.1.3.2

Pinging 206.1.3.2 with 32 bytes of data: 主机A ping 主机B

Request timed out.
Request timed out.
Reply from 206.1.3.2: bytes=32 time=8ms TTL=126
Reply from 206.1.3.2: bytes=32 time=120ms TTL=126

```

```

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=4ms TTL=126 主机B ping 主机A
Reply from 206.1.1.2: bytes=32 time=4ms TTL=126
Reply from 206.1.1.2: bytes=32 time=104ms TTL=126
Reply from 206.1.1.2: bytes=32 time=159ms 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 = 4ms, Maximum = 159ms, Average = 67ms

```

指令输入线程

指令输入线程完成对地址表的手动维护和显示功能——对路由表的增删查和对ARP表的打印。通过使用strtok函数将输入指令按空格拆分，放入向量中，再对向量各元素进行检查，即可完成指令的解析。

```
1 void Router::cmdThrd() {
2     ...
3     char cmd[50];
4     while (true) {
5         cin.getline(cmd, 50);
6         parseCmd(cmd);
7     }
8 }
9
10 void Router::parseCmd(char* cmd) {
11     char* p;
12     vector<string> cmdVec;
13     if(string(cmd) == "") {
14         cout << "【CMD】 Command empty!" << endl;
15         return;
16     }
17     p = strtok(cmd, " ");
18     do {
19         cmdVec.push_back(string(p));
20     } while ((p = strtok(NULL, " ")) != NULL);
21     if(cmdVec[0] == "route") {
22         if(cmdVec[1] == "add") {
23             routingTable->add(cmdVec[2].c_str(), cmdVec[4].c_str(),
24 cmdVec[5].c_str());
25         }
26         if(cmdVec[1] == "delete") {
27             if(cmdVec[2] == "0.0.0.0") {
28                 cout << "【ERR】 Cannot delete default route!" <<
29 endl;
30                 return;
31             }
32             routingTable->del(routingTable->
33 >lookup(inet_addr(cmdVec[2].c_str())));
34         }
35         if(cmdVec[1] == "change") {
36             routingTable->del(routingTable->
37 >lookup(inet_addr(cmdVec[2].c_str())));
38             routingTable->add(cmdVec[2].c_str(), cmdVec[4].c_str(),
39 cmdVec[5].c_str());
40         }
41         if(cmdVec[1] == "print") {
42             cout << routingTable->toStr() << endl;
43         }
44     }
45     if(cmdVec[0] == "arp") {
46         if(cmdVec[1] == "-a") {
47             cout << arpTable->toStr() << endl;
48         }
49     }
50 }
```

# 特殊现象分析

## 处理组播/广播数据包——直接丢弃

使用wireshark捕获的数据包中，包含组播和广播的数据包。对于广播数据包，其以太网帧首部中目的mac地址为全1；对于组播数据包，该目的mac地址第8位为1。如虚拟机使用的简单服务发现协议（SSDP）中的目的mac地址便为组播地址。

该实验中为了处理方便，仅对基于IP协议的单播数据包进行可能的转发操作，因此不考虑对基于组播/广播数据包的上层协议进行相应转发，直接丢弃，通过捕获数据包时限制目的mac地址为打开设备的mac地址进行过滤。

689	1633.890	206.1.1.1	206.1.1.2	ICMP	74 Echo (ping) reply	id=0x0200, seq=2304/9, ttl=128 (request in 688)
690	1634.883	206.1.1.2	206.1.1.1	ICMP	74 Echo (ping) request	id=0x0200, seq=2560/10, ttl=128 (reply in 691)
691	1634.883	206.1.1.1	206.1.1.2	ICMP	74 Echo (ping) reply	id=0x0200, seq=2560/10, ttl=128 (request in 690)
694	1635.883	206.1.1.2	206.1.1.1	ICMP	74 Echo (ping) request	id=0x0200, seq=2816/11, ttl=128 (no response fou
695	1635.883	206.1.1.1	206.1.1.2	ICMP	74 Echo (ping) reply	id=0x0200, seq=2816/11, ttl=128 (request in 694)
698	1636.883	206.1.1.2	206.1.1.1	ICMP	74 Echo (ping) request	id=0x0200, seq=3072/12, ttl=128 (reply in 699)
699	1636.883	206.1.1.1	206.1.1.2	ICMP	74 Echo (ping) reply	id=0x0200, seq=3072/12, ttl=128 (request in 698)
708	1688.079	10.130.69.135	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1	
709	1689.090	10.130.69.135	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1	
710	1690.106	10.130.69.135	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1	
711	1691.114	10.130.69.135	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1	
722	1707.981	206.1.1.1	206.1.1.255	广播	243 Local Master Announcement NANKAI-F3F03CED, Workstation, Server, NT Wc	
767	1780.685	10.130.69.135	224.0.0.251	组播	MDNS	171 Standard query response 0x0000 A, cache flush 10.130.69.135
770	1781.709	10.130.69.135	224.0.0.251	组播	MDNS	171 Standard query response 0x0000 A, cache flush 10.130.69.135
773	1782.701	10.130.69.135	224.0.0.251	组播	MDNS	186 Standard query response 0x0000 AAAA, cache flush 2001:250:401:6561:3
776	1783.687	10.130.69.135	224.0.0.251	组播	MDNS	186 Standard query response 0x0000 AAAA, cache flush 2001:250:401:6561:3
778	1784.705	10.130.69.135	224.0.0.251	组播	MDNS	171 Standard query response 0x0000 A, cache flush 0.0.0.0
780	1785.685	10.130.69.135	224.0.0.251	组播	MDNS	171 Standard query response 0x0000 A, cache flush 0.0.0.0
782	1786.695	10.130.69.135	224.0.0.251	组播	MDNS	186 Standard query response 0x0000 AAAA, cache flush ::
785	1787.698	10.130.69.135	224.0.0.251	组播	MDNS	186 Standard query response 0x0000 AAAA, cache flush ::

878	1928.098601000	10.130.69.135	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1	
Frame 878: 217 bytes on wire (1736 bits), 217 bytes captured (1736 bits) on interface 0						
Ethernet II, Src: 28:39:26:e2:7d:6b (28:39:26:e2:7d:6b), Dst: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)						
Destination: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)						
Address: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)						
.....0..... = LG bit: Globally unique address (factory default)						
.....1..... = IG bit: Group address (multicast/broadcast) mac地址第8位为1的组播地址						
Source: 28:39:26:e2:7d:6b (28:39:26:e2:7d:6b)						
Type: IP (0x0800)						
Internet Protocol Version 4, Src: 10.130.69.135 (10.130.69.135), Dst: 239.255.255.250 (239.255.255.250)						
User Datagram Protocol, Src Port: 62322 (62322), Dst Port: 1900 (1900)						
Hypertext Transfer Protocol						

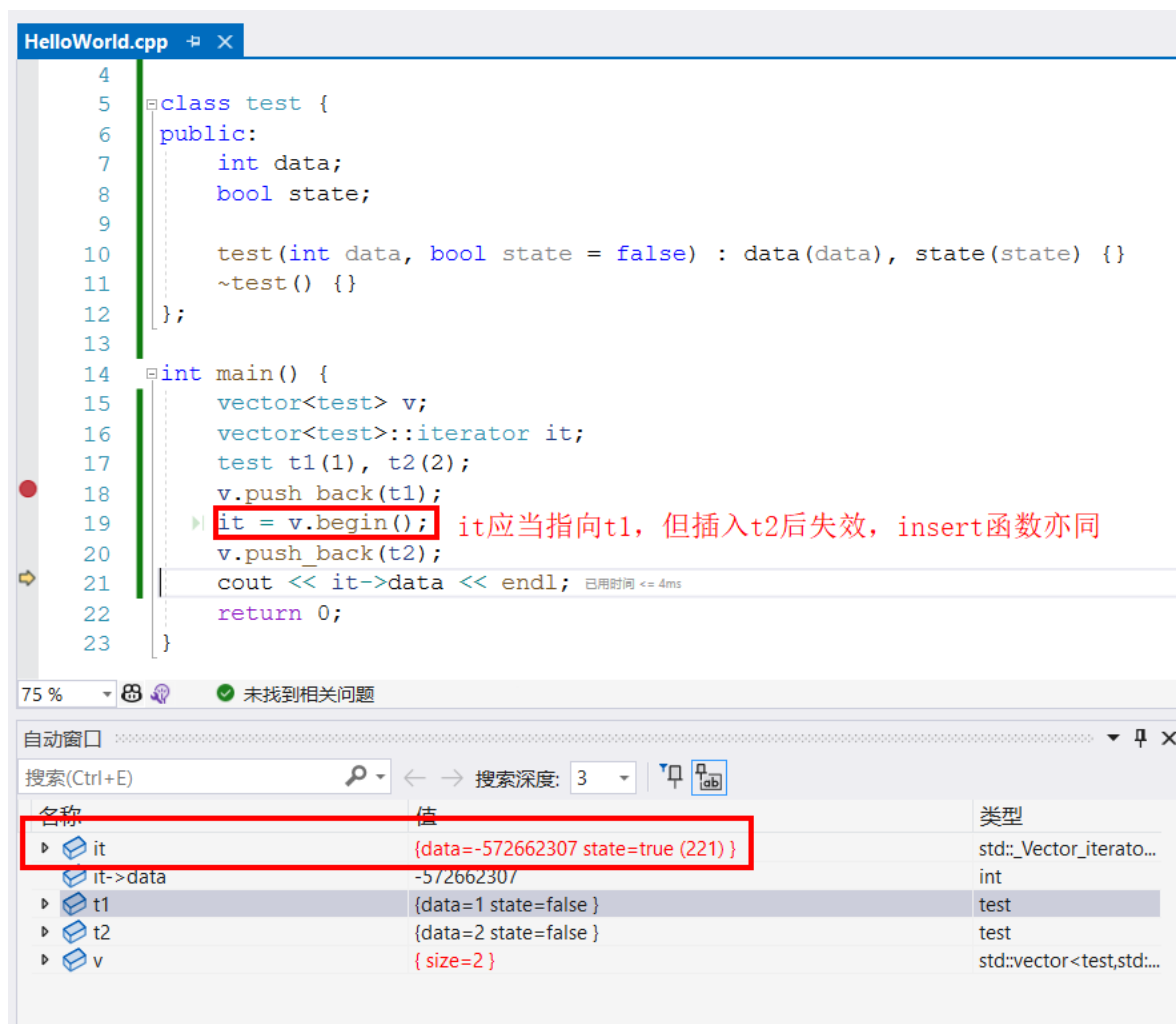
## 转发缓存的一致性问题的

实验中需要对数据包缓存进行维护。对于接收线程，将过滤后的IP数据包加入缓冲区中；对于转发线程，转发IP数据包，并将应当弹出的数据包移出缓冲区。这样，就需要对缓冲区的一致性进行维护。设计如下：

1. 使用临界区（同一进程下，相较于互斥锁更轻量级）进行共享变量访问控制
2. 对缓冲区的起始位置进行加锁。转发线程将数据包加入起始位置之前，转发线程从起始位置之后遍历，这样共享变量只有缓冲区的其实位置，提高了并行度，减少了同步开销

问题为：使用vector进行维护时，任一线程对缓冲区的修改，将导致另一线程中维护的迭代器失效

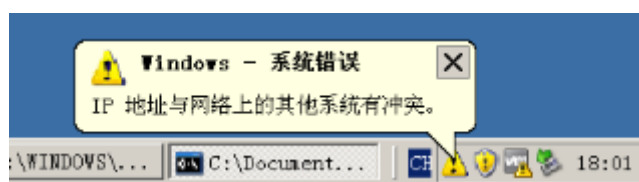
原因为：vector类型为连续地址管理，无论在begin之前插入，还是使用push\_back函数，都将使已确定的迭代器失效，在VS2022下进行调试，得到验证：



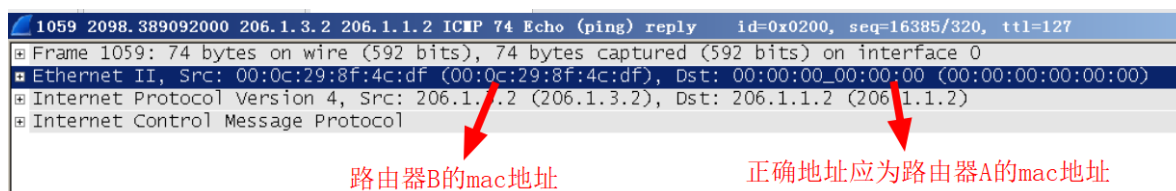
解决：使用自定义链表数据结构进行维护。

## IP地址与网络上的其他系统有冲突

如图所示，当主机A ping 主机B时，在路由器A处提示IP地址重复。且此时不能正确ping通。



通过Wireshark抓包分析，注意到由主机B向主机A的响应报文，在从路由器B发往路由器A的过程中出现异常，目的mac地址应当为主机A的mac地址，而此时却为全0。因此想到为获取路由器B端口mac地址时，广播数据包中源IP地址为路由器A的IP地址，而源mac地址为全0。因此在路由器B接收到广播后，将源IP地址和源mac地址形成映射填入ARP缓存，而该映射中的mac地址为全0。造成了IP地址与mac地址映射的冲突



查看路由器B的ARP缓存表，验证了这一点



```
C:\Documents and Settings\Administrator>arp -a

Interface: 206.1.2.2 --- 0x10003
    Internet Address      Physical Address      Type
    206.1.2.1             00-00-00-00-00-00     dynamic
```

解决：在获取非本机mac地址时，将广播数据包中的源mac地址设置为本机mac地址，路由程序正常工作

## 参考资料

1. [Internet Control Message Protocol - Wikipedia](#)