

目录

1 实验内容说明	2
1.1 实验题目	2
1.2 实验说明	2
2 实验环境	2
3 程序设计与实现	3
3.1 整体设计	3
3.2 设备打开	3
3.3 获取本机 IP 与 MAC	4
3.4 路由表设计	5
3.5 IP 数据包捕获	6
3.6 IP 数据包转发	6
3.7 ICMP 超时报文	7
3.8 数据包缓存	8
3.9 获取 IP-MAC 映射关系	8
4 实验过程	10
4.1 虚拟机测试	10
4.1.1 路由表的增删改查	10
4.1.2 连通性测试	11
4.1.3 工作日志	11
4.1.4 信息展示	12
4.2 物理机测试	12

1 实验内容说明

1.1 实验题目

编程作业（3）简单路由器程序的设计

1.2 实验说明

- 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。
- 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。
- 需要给出路由表的手工插入、删除方法。
- 需要给出路由器的工作日志，显示数据报获取和转发过程。
- 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。

2 实验环境

本次实验借助 Winpcap 实现，机器配置：Windows7 32 位操作系统。物理机测试的网络拓扑如图1所示。

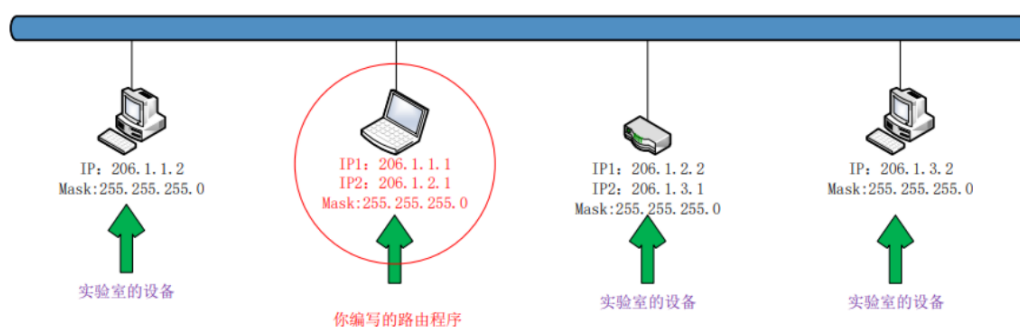


图 1: 实验设备网络拓扑

虚拟机测试环境的操作系统是 Windows Server2003，网络拓扑如图1所示。

3 程序设计与实现

3.1 整体设计

借助 WinpCap, 实现工作在单网络适配器上的路由程序 (单臂路由)。路由程序由 3 大模块组成:

- (a) 路由转发模块
- (b) 路由表管理
- (c) IP-MAC 映射的获取与管理

程序使用两个工作线程, 一个线程负责 ARP 数据包的发送与捕获, 另一个线程负责 IP 数据包的捕获与转发。

3.2 设备打开

Winpcap 提供了获取设备列表的接口, 可以直接获得当前主机的所有网络适配器。

</> CODE 1: 获得设备列表

```
pcap_if_t* find_alldevs()
{
    pcap_if_t *alldevs;
    char errbuf[PCAP_ERRBUF_SIZE];
    if (-1 == pcap_findalldevs_ex(PCAP_SRC_IF_STRING, //获取本机的设备接口
                                NULL,                //无须认证
                                &alldevs,            //指向设备列表首部
                                errbuf)) {            //出错信息保存缓存区

        PANIC("Error in pcap_findalldevs_ex: %s\n", errbuf); //输出错误原因, 并退出
    }
    return alldevs;
}
```

Winpcap 提供的设备列表中, 提供了用于打开设备的设备名称以及 IP 地址等信息, 可以通过 WinpCap 为网卡提供的名称打开设备。

</> CODE 2: 打开设备

```
pcap_t* open_dev(const std::string& name)
{
    char errbuf[PCAP_ERRBUF_SIZE] = {0};
    pcap_t *captureDevice = pcap_open(name.c_str(),
                                      0xFFFF, /*用于捕获ARP包*/
                                      PCAP_OPENFLAG_PROMISCUOUS,
                                      10, /*超时*/
                                      NULL,
                                      errbuf);

    if (NULL == captureDevice) {
        fprintf(stderr, "Can not open device:%s", errbuf);
        exit(EXIT_FAILURE);
    }
}
```

3.3 获取本机 IP 与 MAC

在从 winpcap 获得的 pcap_if_t 列表中, pcap_addr 保存了该设备上分配的 IP 地址等信息。可以直接获得该设备的 IP 地址。

</> CODE 3: 获取本机 IP 地址

```
struct pcap_addr
{
    struct pcap_addr *next;
    struct sockaddr *addr;          /* 网络地址 */
    struct sockaddr *netmask;      /* 子网掩码 */
    struct sockaddr *broadaddr;    /* 广播地址 */
    struct sockaddr *dstaddr;
};
```

而本机的 MAC 地址则需要手动获取, 即通过伪造 ARP 报文来获得本机 MAC 地址。

</> CODE 4: 过滤指定的 IP 所对应 ARP 回复数据包的过滤器

```
void ArpTable::setFilter(pcap_t *dev, uint32_t targetip)
{
    IPv4Addr reqIP(targetip);
    ostringstream filterCondition;
    filterCondition<< "arp and (ether[21]=0x2) and (arp host "<< reqIP<<")";
    string str = filterCondition.str();
    struct bpf_program filter;

    if (pcap_compile(dev, &filter, str.c_str(), 1, 0) < 0) {
        PANIC("Unable to compile the packet filter. Check the syntax.");
    }
    if (pcap_setfilter(dev, &filter) < 0) {
        PANIC("\nError setting the filter.\n");
    }
}
```

</> CODE 5: 获取本机 MAC 地址

```
Mac ArpTable::getMac(pcap_t *dev, uint32_t targetip, uint32_t myip, Mac mymac)
{
    setFilter(dev, targetip); // 设置只接受所需MAC对于ARP回复的过滤器
    ARPFrame reqMac(targetip, myip, mymac);
    if (0 != pcap_sendpacket(dev, (u_char *)&reqMac, sizeof(reqMac))) {
        PANIC("send arp err!");
    }
    struct pcap_pkthdr *pkthdr = NULL;
    const u_char *rawdata = NULL;

    if (1 != pcap_next_ex(dev, &pkthdr, &rawdata))
        continue;
    ARPFrame *arp = (ARPFrame *)rawdata;
    if (arp->frameHeader.frameType == 0x0608 /*htons(0x0806)*/ && /* 必须是arp包 */
        arp->operation == 0x0200 /*htons(0x0002)*/ && /* 必须是arp应答 */
        arp->sendIP.addr == targetip && /* 必须是查询IP的应答 */
        arp->sendIP.addr != myip) /* 不是刚刚发出的包 */
    {
        return arp->sendHA;
    }
    return Mac();
}
```

3.4 路由表设计

路由表需要具备基本的插入、删除、修改、查找的功能。通过三元组列表实现。

</> CODE 6: 路由表数据结构

```
struct RouterTableItem
{
    uint32_t netid;
    uint32_t subnetMask;
    uint32_t nextHop;
};

std::vector<RouterTableItem> table; //维护有序列表, netid主序, subnetMask次之
```

为了保证最长匹配的高效率实现, 需要保证列表插入时保持列表有序。

</> CODE 7: 路由表插入

```
inline void RouterTable::insert(uint32_t netid, uint32_t subnetMask, uint32_t nextHop)
{
    Guard g(mutex); //互斥锁 类似C++11中的lock_guard<mutex>

    RouterTableItem item={netid&subnetMask, subnetMask, nextHop};

    std::vector<RouterTableItem>::iterator curr=table.begin();
    std::vector<RouterTableItem>::iterator end=table.end();
    while(curr!=end && *curr<item) curr++; //遍历至恰好大于本项的位置

    if(curr!=end && *curr==item) { //if 已存在
        *curr=item; //覆盖
    } else {
        table.insert(curr,item); //插入
    }
};
```

查找时从列表末尾向开始遍历, 返回第一个匹配项。由于列表是有序的, 因此可以保证最长匹配。

</> CODE 8: 路由表查找

```
inline uint32_t RouterTable::find(uint32_t destip) const
{
    Guard g(mutex);
    int size=table.size();
    for(int i=size-1; i>=0; i--) {
        if((destip&(table[i].subnetMask))==table[i].netid) {
            if(table[i].nextHop==0) { //if 直接投递
                return destip; //直接返回目的IP
            } else {
                return table[i].nextHop;
            }
        }
    }
    return 0;
}
```

3.5 IP 数据包捕获

IP 数据包捕获时，可以设置过滤器，只捕获目的 MAC 地址为自己的 IP 数据包，然后再做判断于处理。

</> CODE 9: IP 数据包过滤

```
void Router::setFilter()
{
    ostringstream filterCondition;
    filterCondition<< "ip and (ether dst "<<mymac<<")";
    string str = filterCondition.str();
    bpf_program filter;

    if (pcap_compile(dev, &filter, str.c_str(), 1, 0) < 0) {
        PANIC("Unable to compile the packet filter. Check the syntax.");
    }
    if (pcap_setfilter(dev, &filter) < 0) {
        PANIC("\nError setting the filter.\n");
    }
}
```

在接受数据包时，只对完整捕获的数据包进行路由转发，同时对数据包的校验和进行检查。

</> CODE 10: 计算校验和

```
uint16_t checksum(uint16_t *words, size_t len)
{
    uint32_t checksum = 0;
    for (int i = 0; i < len; i++)
        checksum += swapByteOrder(words[i]);

    checksum = (checksum >> 16) + (checksum & 0x0000FFFF);
    checksum = (checksum >> 16) + checksum;
    return swapByteOrder((uint16_t)~checksum);
}
```

3.6 IP 数据包转发

如果目的 IP 不是自己，查看目的 IP 在本地 IP-MAC 映射表中是否存在，存在则查找路由表，并转发。

</> CODE 11: 查表转发

```
if (iphdr->ttl-1>0) { //route pkg
    if (find(ips.begin(), ips.end(), iphdr->des_ip) != ips.end())
        continue; // dst ip is me

    iphdr->ttl--;
    uint32_t nextHop = table.findNextHop(iphdr->des_ip);
    if (nextHop != 0) { //no table item
        deliver(iphdr, pkgsize, nextHop);
    }
}
```

转发时，修改报文的以太网 MAC 和 IP 头 TTL 校验和，并尝试获取 IP 地址对应 MAC 地址，若当前还未获取则对数据包进行缓存。

</> CODE 12: 转发数据包

```
bool Router::deliver(iphdr_s* data, size_t size, uint32_t dstip)
{
    Guard g(mutex);
    Mac dstmac=arps.get(dstip);
    if(!dstmac.isValid()){ //当前没有IP-MAC对应关系
        arps.reponseMac(this, data, size, dstip); //cache pkg
        return false;
    }
    memcpy(data->frmhdr.des_mac, dstmac.addr, 6);
    memcpy(data->frmhdr.src_mac, mymac.addr, 6);

    /* recalculate checksum */
    data->check_sum=0;
    uint16_t *words = (uint16_t *)&(data->ver_hlen);
    size_t len = (sizeof(iphdr_s)-sizeof(frmhdr_s)) / sizeof(uint16_t);
    data->check_sum=checksum(words, len);

    /* send pkg */
    if (0 != pcap_sendpacket(dev, (u_char *)data, size)) {
        PANIC("Send pkg err!");
    }

    return true;
}
```

3.7 ICMP 超时报文

如果 TTL 为 1 则回送 ICMP 超时报文；icmp 数据包定义与构造如下。

</> CODE 13: 构造 ICMP 超时回复

```
typedef struct icmp_pkg {
    iphdr_s iphdr;
    uint8_t type;
    uint8_t code;
    uint16_t checksum;
    uint32_t other;
    uint8_t data[sizeof(iphdr_s)-sizeof(frmhdr_s)+8];
} icmp_s;

void Router::ICMPTimeout(icmp_s* icmp, const iphdr_s* pkg)
{
    /* set icmp data */
    memcpy(icmp->data, &(pkg->ver_hlen), sizeof(icmp->data));
    icmp->type=11;
    icmp->code=0;
    icmp->other=0;
    icmp->checksum=0;

    uint16_t *words1 = (uint16_t *)&(icmp->type);
    size_t len1 = (sizeof(icmp_s)-sizeof(iphdr_s)) / sizeof(uint16_t);
    icmp->checksum=checksum(words1, len1); //计算ICMP校验和

    //IP头与以太网帧的构造，略
}
```

3.8 数据包缓存

暂时未确定 IP 对应 MAC 地址的 IP 数据包，需要暂时被缓存，等到获取到对应数据包时再进行转发。

</> CODE 14: 数据包缓存的数据结构

```
struct CachePkg
{
    iphdr_s* pkg;    //转发的数据包
    size_t size;    //数据包尺寸
    Router* router;  //进行转发的路由器
    time_t time;    //缓存时间
};

multimap<uint32_t,CachePkg> cache; /* key为要投递的目的IP地址 */
```

</> CODE 15: 数据包缓存与超时删除

```
void ArpTable::reponseMac(Router* router,iphdr_s* data,size_t size, uint32_t dstip)
{
    Guard g(mutex2); //互斥锁
    char* mydata=new char[size];
    memcpy(mydata,data,size);
    CachePkg pkg= {(iphdr_s*)mydata,size,router,time(NULL)};
    cache.insert(make_pair(dstip,pkg));

    //delete timeout cache
    multimap<uint32_t,CachePkg>::iterator m,beg= cache.begin(),end= cache.end();
    time_t t=time(NULL);
    for(m = beg; m != end;) {
        const CachePkg &pkg=m->second;
        if(pkg.time+OVER_TIME<t) { //如果超时
            delete[] (char*) (pkg.pkg);
            cache.erase(m++);
        } else {
            m++;
        }
    }
}
```

3.9 获取 IP-MAC 映射关系

设置过滤器，只接受和处理 ARP 回复。

</> CODE 16: ARP 过滤器

```
void ArpTable::setFilter(pcap_t *dev)
{
    const char condition[]="arp and (ether[21]=0x2)";
    struct bpf_program filter;
    if (pcap_compile(dev, &filter, condition, 1, 0) < 0) {
        PANIC("Unable to compile the packet filter. Check the syntax.");
    }
    if (pcap_setfilter(dev, &filter) < 0) {
        PANIC("\nError setting the filter.\n");
    }
}
```


当尝试获取 IP 对应 MAC 地址时，若缓存中无此项，或缓存超时，则发送 ARP 请求重新获取 IP 地址对应 MAC 地址；否则直接返回结果。

</> CODE 17: 发送 ARP 请求

```
Mac ArpTable::get(uint32_t targetip)
{
    Guard g(mutex);
    if(addr2mac.count(targetip)!=0) { //存在缓存
        if(age.count(targetip)==0||age[targetip]+AGED>time(NULL)) { //缓存未超时
            return addr2mac[targetip];
        }
    }

    //发送arp请求
    ARPFrame reqMac(targetip, myip, mymac);
    if (0 != pcap_sendpacket(dev, (u_char *)&reqMac, sizeof(reqMac))) {
        PANIC("send arp err!");
    }
    return Mac();
}
```

此外在另外一个单独的线程，将所有接受到的 ARP 回复存到缓存中，并对相应已缓存的数据包进行发送。

</> CODE 18: 处理 ARP 回复，发送缓存数据包

```
unsigned __stdcall ArpTable::runArp(void* arps)
{
    ArpTable* me=(ArpTable*) arps;
    ArpTable::setFilter(me->dev);

    struct pcap_pkthdr *pkthdr = NULL;
    const u_char *rawdata = NULL;
    while (!me->stop) {
        if (1 != pcap_next_ex(me->dev, &pkthdr, &rawdata)) //if 接受到了数据包
            continue;

        ARPFrame *arp = (ARPFrame *)rawdata;
        if (arp->frameHeader.frameType == 0x0608 /*htons(0x0806)*/ && /* 必须是arp包 */
            arp->operation == 0x0200/*htons(0x0002)*/ && /* 必须是arp应答 */
            arp->sendIP.addr != me->myip) /* 不是刚刚发出的包 */
        {
            Guard g(me->mutex2); //互斥锁
            //update arp table
            uint32_t ip=arp->sendIP.addr;
            me->setArp(ip,Mac(arp->sendHA));

            //flush cache
            multimap<uint32_t,CachePkg>::iterator m,beg,end;
            beg = me->cache.lower_bound(ip);
            end = me->cache.upper_bound(ip);
            for(m = beg; m != end; m++) {
                const CachePkg &pkg=m->second;
                pkg.router->deliver(pkg.pkg,pkg.size,ip);
                delete[] ((char*)pkg.pkg);
            }
            me->cache.erase(ip);
        }
    }
}
```

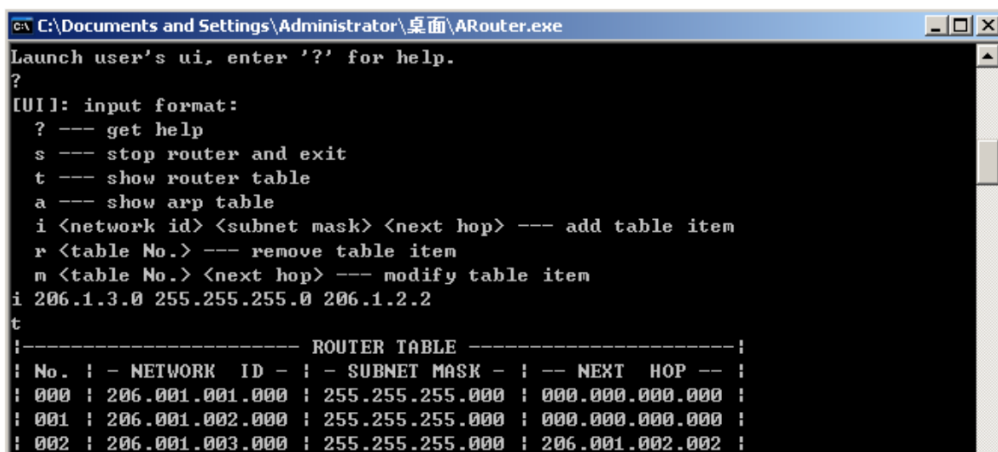
4 实验过程

4.1 虚拟机测试

实验环境在第二节中已经说明。

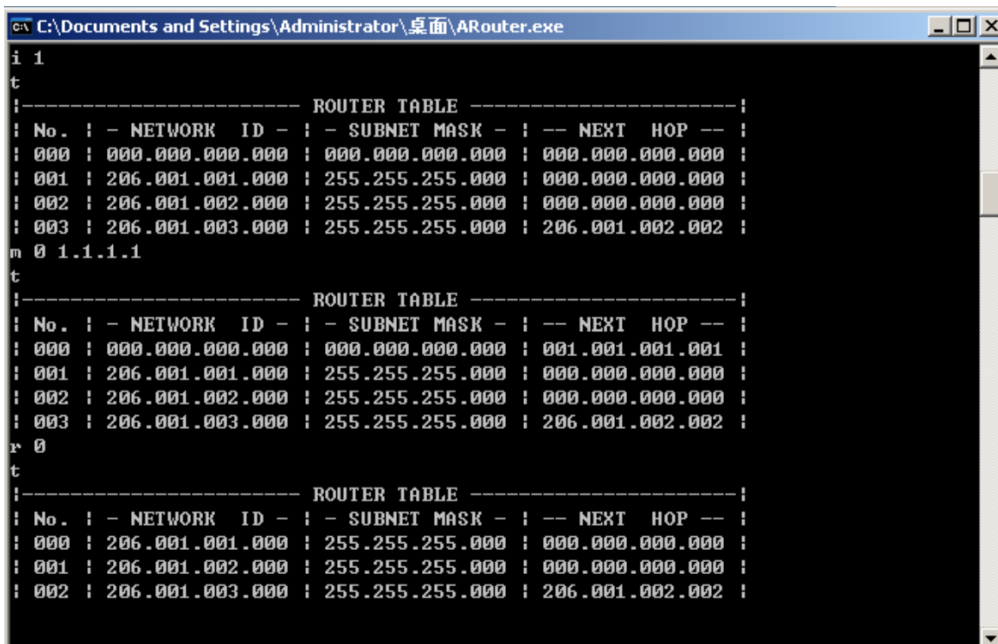
4.1.1 路由表的增删改查

使用规定的格式在命令行对路由表进行插入，结果如图2所示。同时也可以对路由表进行修改和删除，结果如图3所示。



```
C:\Documents and Settings\Administrator\桌面\ARouter.exe
Launch user's ui, enter '?' for help.
?
[UI]: input format:
? --- get help
s --- stop router and exit
t --- show router table
a --- show arp table
i <network id> <subnet mask> <next hop> --- add table item
r <table No.> --- remove table item
m <table No.> <next hop> --- modify table item
i 206.1.3.0 255.255.255.0 206.1.2.2
t
----- ROUTER TABLE -----
No. | - NETWORK ID - | - SUBNET MASK - | -- NEXT HOP -- |
000 | 206.001.001.000 | 255.255.255.000 | 000.000.000.000 |
001 | 206.001.002.000 | 255.255.255.000 | 000.000.000.000 |
002 | 206.001.003.000 | 255.255.255.000 | 206.001.002.002 |
```

图 2: 路由表的插入

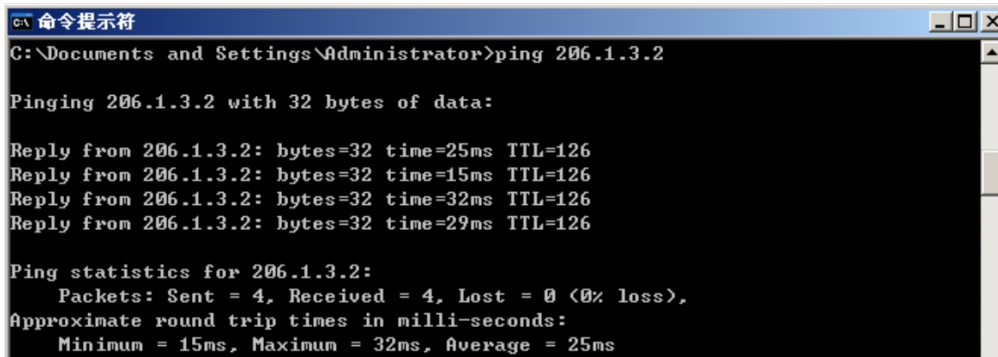


```
C:\Documents and Settings\Administrator\桌面\ARouter.exe
i 1
t
----- ROUTER TABLE -----
No. | - NETWORK ID - | - SUBNET MASK - | -- NEXT HOP -- |
000 | 000.000.000.000 | 000.000.000.000 | 000.000.000.000 |
001 | 206.001.001.000 | 255.255.255.000 | 000.000.000.000 |
002 | 206.001.002.000 | 255.255.255.000 | 000.000.000.000 |
003 | 206.001.003.000 | 255.255.255.000 | 206.001.002.002 |
m 0 1.1.1.1
t
----- ROUTER TABLE -----
No. | - NETWORK ID - | - SUBNET MASK - | -- NEXT HOP -- |
000 | 000.000.000.000 | 000.000.000.000 | 001.001.001.001 |
001 | 206.001.001.000 | 255.255.255.000 | 000.000.000.000 |
002 | 206.001.002.000 | 255.255.255.000 | 000.000.000.000 |
003 | 206.001.003.000 | 255.255.255.000 | 206.001.002.002 |
r 0
t
----- ROUTER TABLE -----
No. | - NETWORK ID - | - SUBNET MASK - | -- NEXT HOP -- |
000 | 206.001.001.000 | 255.255.255.000 | 000.000.000.000 |
001 | 206.001.002.000 | 255.255.255.000 | 000.000.000.000 |
002 | 206.001.003.000 | 255.255.255.000 | 206.001.002.002 |
```

图 3: 路由表的删除与修改

4.1.2 连通性测试

开启路由器后，使用 IP:206.1.1.2 的主机去 ping 另一台 IP:206.1.3.2 的主机，结果如图4所示。
使用 tracert 进行测试，结果如图5所示。



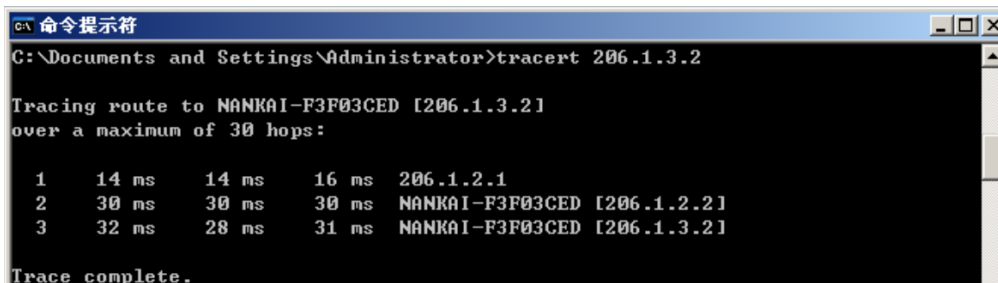
```
CA 命令提示符
C:\Documents and Settings\Administrator>ping 206.1.3.2

Pinging 206.1.3.2 with 32 bytes of data:

Reply from 206.1.3.2: bytes=32 time=25ms TTL=126
Reply from 206.1.3.2: bytes=32 time=15ms TTL=126
Reply from 206.1.3.2: bytes=32 time=32ms TTL=126
Reply from 206.1.3.2: bytes=32 time=29ms 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 = 15ms, Maximum = 32ms, Average = 25ms
```

图 4: ping 从 IP:206.1.1.2 到 IP:206.1.3.2



```
CA 命令提示符
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:

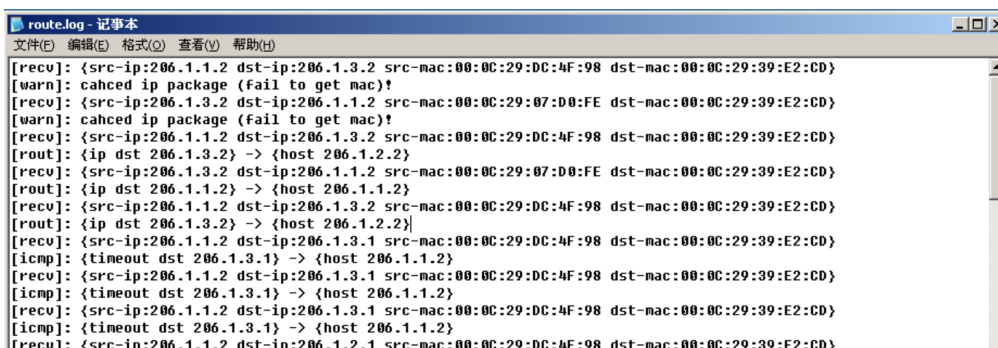
  1    14 ms    14 ms    16 ms    206.1.2.1
  2    30 ms    30 ms    30 ms    NANKAI-F3F03CED [206.1.2.2]
  3    32 ms    28 ms    31 ms    NANKAI-F3F03CED [206.1.3.2]

Trace complete.
```

图 5: tracert 从 IP:206.1.1.2 到 IP:206.1.3.2

4.1.3 工作日志

在日志文件中，保存了捕获与转发的记录，如图6所示。



```
route.log - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

[recu]: {src-ip:206.1.1.2 dst-ip:206.1.3.2 src-mac:00:0C:29:D0:4F:98 dst-mac:00:0C:29:39:E2:CD}
[warn]: cahced ip package (fail to get mac)!
[recu]: {src-ip:206.1.3.2 dst-ip:206.1.1.2 src-mac:00:0C:29:D0:FE dst-mac:00:0C:29:39:E2:CD}
[warn]: cahced ip package (fail to get mac)!
[recu]: {src-ip:206.1.1.2 dst-ip:206.1.3.2 src-mac:00:0C:29:D0:4F:98 dst-mac:00:0C:29:39:E2:CD}
[route]: {ip dst 206.1.3.2} -> {host 206.1.2.2}
[recu]: {src-ip:206.1.3.2 dst-ip:206.1.1.2 src-mac:00:0C:29:D0:FE dst-mac:00:0C:29:39:E2:CD}
[route]: {ip dst 206.1.1.2} -> {host 206.1.1.2}
[recu]: {src-ip:206.1.1.2 dst-ip:206.1.3.2 src-mac:00:0C:29:D0:4F:98 dst-mac:00:0C:29:39:E2:CD}
[route]: {ip dst 206.1.3.2} -> {host 206.1.2.2}
[recu]: {src-ip:206.1.1.2 dst-ip:206.1.3.1 src-mac:00:0C:29:D0:4F:98 dst-mac:00:0C:29:39:E2:CD}
[icmp]: {timeout dst 206.1.3.1} -> {host 206.1.1.2}
[recu]: {src-ip:206.1.1.2 dst-ip:206.1.3.1 src-mac:00:0C:29:D0:4F:98 dst-mac:00:0C:29:39:E2:CD}
[icmp]: {timeout dst 206.1.3.1} -> {host 206.1.1.2}
[recu]: {src-ip:206.1.1.2 dst-ip:206.1.3.1 src-mac:00:0C:29:D0:4F:98 dst-mac:00:0C:29:39:E2:CD}
[icmp]: {timeout dst 206.1.3.1} -> {host 206.1.1.2}
[recu]: {src-ip:206.1.1.2 dst-ip:206.1.2.1 src-mac:00:0C:29:D0:4F:98 dst-mac:00:0C:29:39:E2:CD}
```

图 6: 日志文件

4.1.4 信息展示

在路由器初始化后，能够显示本机 IP 地址及本机 MAC 地址，如图??所示。

```
C:\Documents and Settings\Administrator\桌面\ARouter.exe
Which device do you choose to use for the router?[0,1]>0

After initialization, the router's table is as follows:
!----- ROUTER TABLE -----!
! No. | - NETWORK ID - | - SUBNET MASK - | -- NEXT HOP -- |
! 000 | 206.001.001.000 | 255.255.255.000 | 000.000.000.000 |
! 001 | 206.001.002.000 | 255.255.255.000 | 000.000.000.000 |

initializing arp table...
After initialization, the arp table is as follows:

!----- ARP TABLE -----!
! 206.001.002.001 -> 00:0C:29:39:E2:CD !

Local information:
- device name: rpcap://Device\NPF_{3AC00148-1BFF-47AF-8441-F1D14A619031}
- local mac: 00:0C:29:39:E2:CD
- all ip address: 206.1.2.1 206.1.1.1

initializing router...
initializing log file...
log will be written in 'route.log'.
initializing new thread...
Router launched.
Launch user's ui, enter '?' for help.
```

图 7: mac

在路由器工作一段时间后，展示其 ARP 缓存，如图8所示。

```
C:\Documents and Settings\Administrator\桌面\ARouter.exe
a
!----- ARP TABLE -----!
! 206.001.002.001 -> 00:0C:29:39:E2:CD !
! 206.001.001.002 -> 00:0C:29:DC:4F:98 !
! 206.001.002.002 -> 00:0C:29:07:D0:FE !
```

图 8: arp 缓存

4.2 物理机测试

物理机测试在检查作业时完成，未能截图。