实验报告

一、 实验名称

基于 Winpcap 和 Jpcap 的网络嗅探器的实现。

二、 实验目的

① 掌握嗅探器的工作原理

② 熟悉 Winpcap 的使用

③ 掌握基于 Winpcap 的网络嗅探器的开发过程

三、 实验内容

开发出一个 Windows 平台上的网络嗅探器工具，能够显示所捕获的数据包，并能做相应的分析和统计。

主要内容如下：

① 列出所监测主机的所有网卡，选择一个网卡，设置为混杂模式进行监听。

② 捕获所有流经网卡的数据包，并利用 WinPcap 函数库设置过滤规则。

③ 分析捕获到的数据包的包头和数据，按照各种协议的格式进行格式化显示。

④ 将所开发工具的捕获和分析结果与常用的嗅探器，如：Wireshark，进行比较，完善程序代码。

四、 实验过程

（1）开发环境：

Windows 7 专业版+Eclipse Indigo+Java+SWT+Winpcap+Jpcap

（2）基础知识：

由于本嗅探器的开发使用 Java 语言，需要用到 Winpcap 和 Jpcap 两个中间件。

Winpcap 是 Windows 平台下，访问数据链路层的一个工具，目的在于为 Windows 应用程序提供底层网络的访问能力。

Jpcap 是为了弥补 Java 语言在数据链路层访问的弱势而开发出来的，它实际上工作在 Wincap 之上，调用 Winpcap 来实现对数据包的捕获，同样，Jpcap 支持多种操作系统。

Jpcap 更多的信息：http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/index.html

（3）配置环境：

① 安装 Winpcap 程序

本实验使用：Winpcap4.1.2 安装包

② 安装 Jpcap 程序

本实验使用：JpcapSetup0.7 安装包

③ 移动动态链接库

安装完成 Winpcap 和 Jpcap 后，需要找到相应的动态链接库文件（.dll）和相应的包文件（.jar），然后将这两个文件分别放到指定的文件夹。

一般情况下，Jpcap.dll 安装在 C:\Windows\System32 目录下，jpcap.jar 安装在 C:\Windows\Sun\Java\lib\ext 目录下。

将 Jpcap.dll 复制到 jre 路径下的 bin 文件夹下，将 jpcap.jar 复制到 jre 路径下的 lib\ext 文件夹下。

④ 新建 Eclipse SWT 工程

在相应的类中加入：

import jpcap.packet.*

import jpcap.*

其中第一个包中包含了对常见数据包的描述，第二个包中包含了对网络接口以及抓包器的描述。

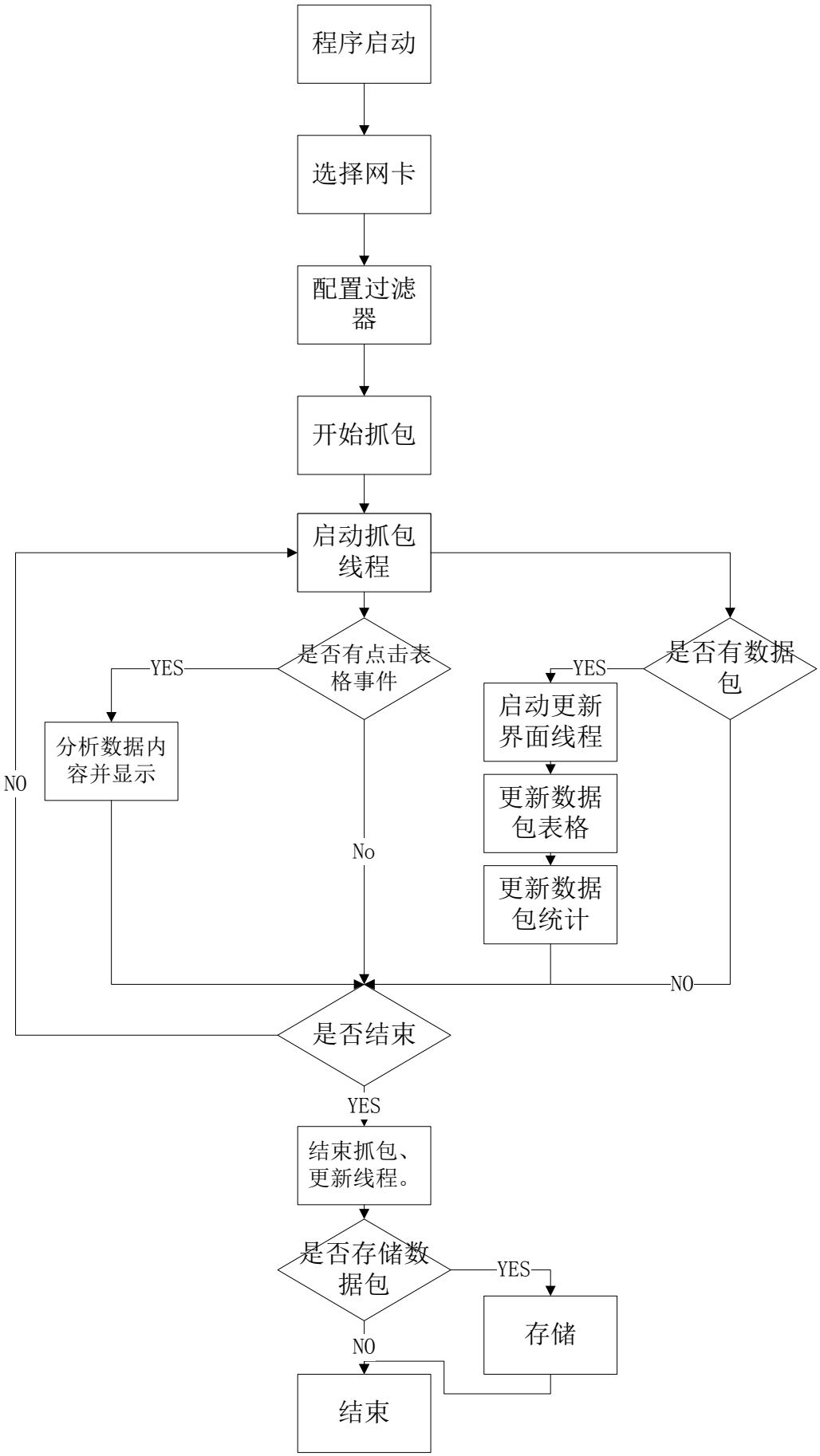
在完成以上四个步骤后，就可以在 eclipse 环境下进行嗅探器的开发了。

Jpcap 的 API 和相应文档

http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/javadoc/index.html

（4）开发过程：

① 程序流程图


程序启动
选择网卡
配置过滤器
开始抓包
启动抓包线程
是否有点击表格事件
YES
分析数据内容并显示
NO
No
是否有数据包
YES
启动更新界面线程
更新数据包表格
更新数据包统计
NO
是否结束
YES
结束抓包、更新线程。
是否存储数据包
YES
存储
NO
结束

② Jpcap 抓包原理

Jpcap 是一个中间件，它提供在 java 环境下调用 Winpcap 进行数据包抓取的 API。下面
分别对 Jpcap 的主要函数进行解释。

首先是获取本机中的网卡列表：

```
NetworkInterface[] devices=JpcapCaptor.getDeviceList();
```

使用上述函数获得本机的网卡列表后，通过 devices 数组就可以访问并获得相应的 device 对象。

然后是打开相应的网卡：

```
JpcapCaptor captor=JpcapCaptor.openDevice(device[index], 65535, false, 20);
```

调用 JpcapCaptor 的 openDevice 函数，该函数参数从左到右分别为 deveice 对象，缓冲区大小，是否设为混杂模式，线程超时时钟。该函数返回一个 JpcapCaptor 对象，使用该对象就可以对数据包进行获取。

然后就是给 captor 设置过滤器：

```
mycaptor.setFilter(Config.filter, true);
```

其中 Config.filter 是我定义的一个 String 类型的值，可以取 ether,fddi, tr, ip, ip6, arp, rarp, decnet, tcp and udp.

之后是实现自己的数据包接收器类，自己定义一个类继承 PacketReceiver 接口，在重写的函数中描述自己对于数据包的处理：

```
class PacketCaptor implements PacketReceiver
{
    @Override
    public void receivePacket(Packet temppacket) {
        // TODO Auto-generated method stub
        packetindex++;
        packet=temppacket;
        packets.add(temppacket);
    }
}
```

然后在调用非阻塞的 processPacket 函数对数据包进行抓取：

```
captor.processPacket(1, captorhandler);
```

其中 1 表示每次抓取一个数据包，captorhandler 为 PacketCaptor 的一个对象。

这样，在 captor 获取到数据包后，将会将数据包交给 captorhandler 进行处理。Captorhandler 会将数据包存入一个全局的 ArrayList 链表，其他类就可以对这个链表中的数据包进行分析和处理了。

③ 程序关键算法

在嗅探器的实现中，使用 SWT 框架技术。嗅探器的 UI 是一个单独的线程，而本嗅探器又要实时的对数据包进行抓取，同时对数据包进行分析。因此，在本实验中必须使用多线程技术。一个线程负责构建和显示 UI，一个线程负责抓取数据包并存入链表，另一个线程负责实时的更新 UI 中的某些控件。因此，本实验的关键在于线程间的同步。

数据包抓取线程的定义如下：

```
class CaptureThread extends Thread
{
    @Override
    public void run() {
        // TODO Auto-generated method stub
        super.run();

        while(!this.isInterrupted())
        {
            captor.startCapture();
            packets=captor.getPackets();
        }
        if(this.isInterrupted())
        {
            captor.stopCapture();
        }
    }
```

```
}
```

显示线程：

```
class DisplayThread extends Thread
{
    @Override
    public void run() {
        // TODO Auto-generated method stub
        super.run();
        while(!this.isInterrupted())
        {
            if(displaypacketindex<packets.size())
            {
                displaypacket();
            }
        }
        if(this.isInterrupted())
        {
        }
    }
}
```

实时更新 UI 线程：

```
Display.getDefault().asyncExec(runnable);
```

除了对 UI 进行处理，嗅探器更关键的一部分在于对数据包进行分析，为此我定义了一些类，这些类的功能各不相同，下面分别进行说明：

MainApplication.java：该类是程序的主类，主要任务是构建程序的框架，生成线程，保存各种变量，以及将数据包交给相应类进行处理并进行显示。

Config.java：该类主要功能是存储嗅探器的配置信息。

FileUtils.java：该类的主要功能是对数据包和抓取报告的存储和读取。

SimpleAnalyzer.java：该类是数据包的简单分析类，由于实时更新时需

要较高的速度，因此，只对数据包的基本信息进行分析并将显示结果显示在 table 中。

DetailAnalyzer.java：该类的主要功能是对数据包进行详细的解析，包括 arp，ip，icmp，igmp，udp，tcp 等基本协议类型进行了分析。

Captor.java：该类主要对数据包进行抓取，并交给主类。

PacketStat.java：该类主要对本次抓取进行统计，包括各种数据包出现的次数。

DnsDialog.java：该类是一个对话框，其中包括了对 DNS 协议的重点分析，并维护相应控件。

ChartAnalyzer.java：该类是一个对话框，用于对数据包统计信息进行作图，绘制相应表格，方便对数据包的整体情况进行浏览。

④ 碰到的问题及解决方法

在本次实验，线程的同步是个比较费时的工作。由于本程序使用了多线程，因此，数据包的抓取，显示和分析由不同的线程来完成，这就产生了一个线程的同步的问题。同时，在 SWT 中，不允许其他线程访问不是由自身所创建的控件。因此，需要使用专用的函数对线程进行修饰。在这一点上花费了不少时间。

在线程的同步中，比较关键的语句是：

`if(displaypacketindex<packets.size())`显示线程和抓取线程的同步量判断

`Display.getDefault().asyncExec(runnable);` 其他线程异步更新控件

`while(!this.isInterrupted())` 停止线程的方法

在使用了上述方法后，线程的同步就可以正确的完成。

⑤ 收获和体会

完成了本次实验，收获很大。

首先，本次实验需要使用界面。以前虽然使用过相关的 UI 构造方法，但是并不熟悉和全面，通过本次实验，我对 SWT 的机制有了更深入的了解，这其中包括 shell 和 display 的关系，多线程更新控件的实现，以及系统维护 UI 的机制。理解这些对我以后的工作有比较大的帮助。

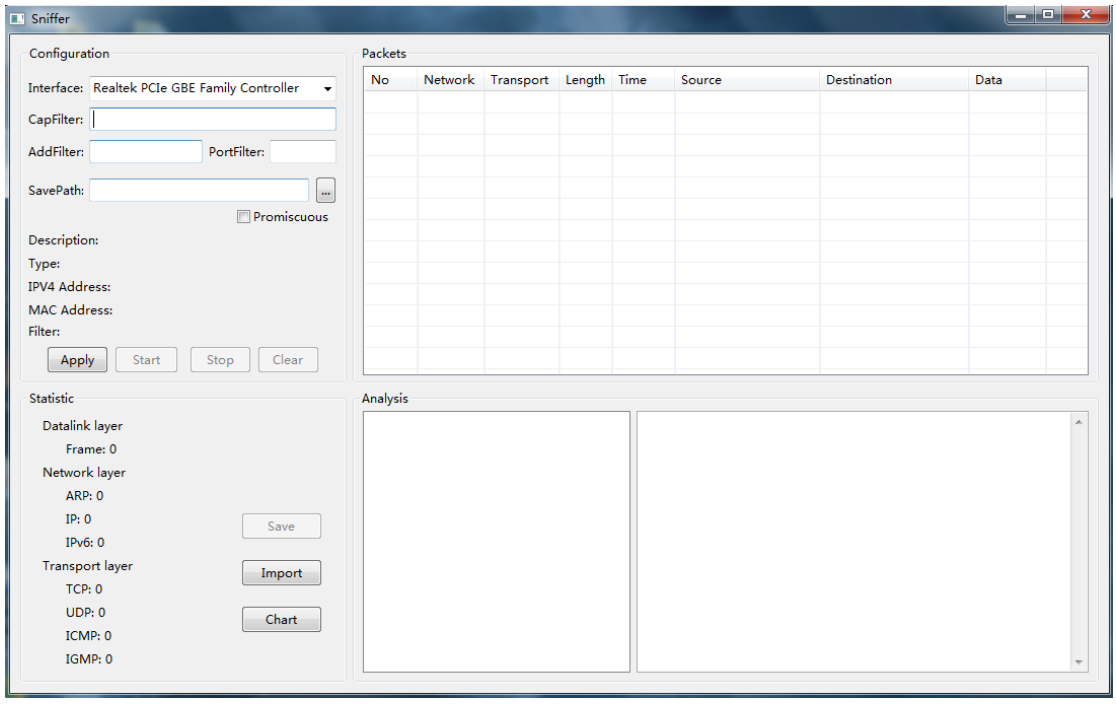
其次，本次实验需要对大量不同的数据包进行分析，这加强了我对数组处理的能力。同时，以前虽然对各个协议的过程有了解，但是并不清楚各个协议的数据包的具体实现过程，并不清楚网络中传输数据包的具体样子，通过这次实验，我对各个协议的具体过程，数据包的具体内容有了更深入的理解，这对于我以后的网络研究非常有帮助。

再次，通过本次实验，我加强了对网络的理解，以前虽然学习过很多网络的相关知识，但是对于实际中网络的样子并没有一个非常清晰的认识，这次实验给了我关于网络的更清晰的轮廓。

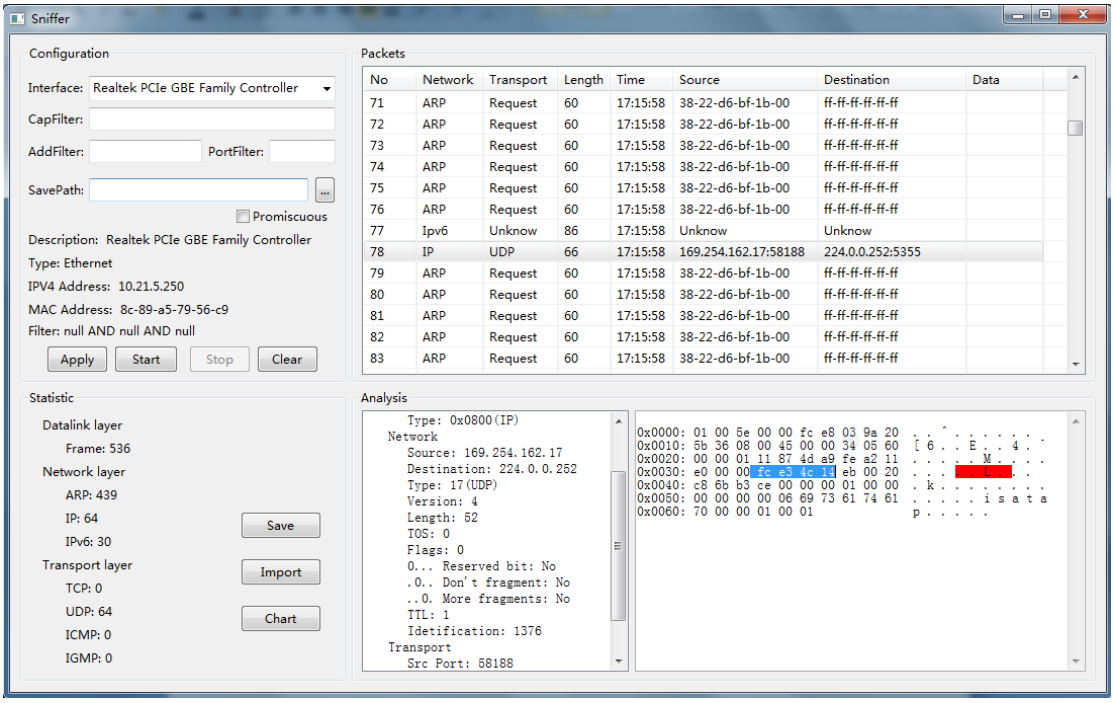
⑥ 程序运行结果和截图

程序的运行环境为：Windows 7 操作系统+Java 运行时环境，由于我所在的计算所网络技术中心使用交换机构建局域网，因此，无论是否将网卡设置为混杂模式，嗅探器都无法嗅探到不是发往自己的数据包。

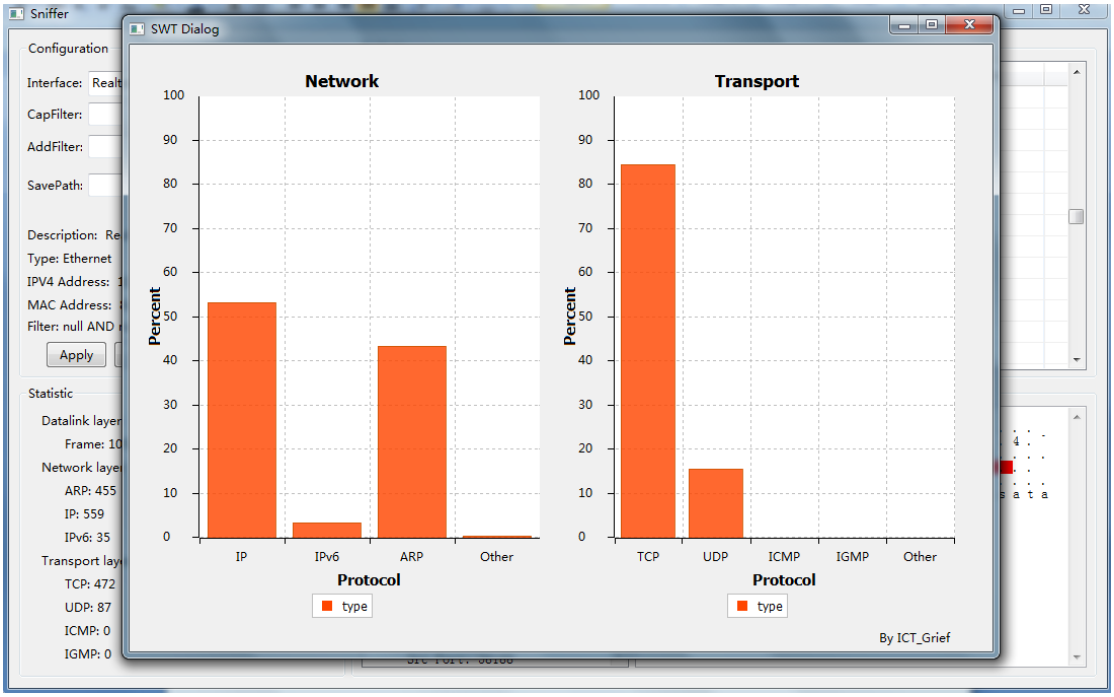
主程序界面：



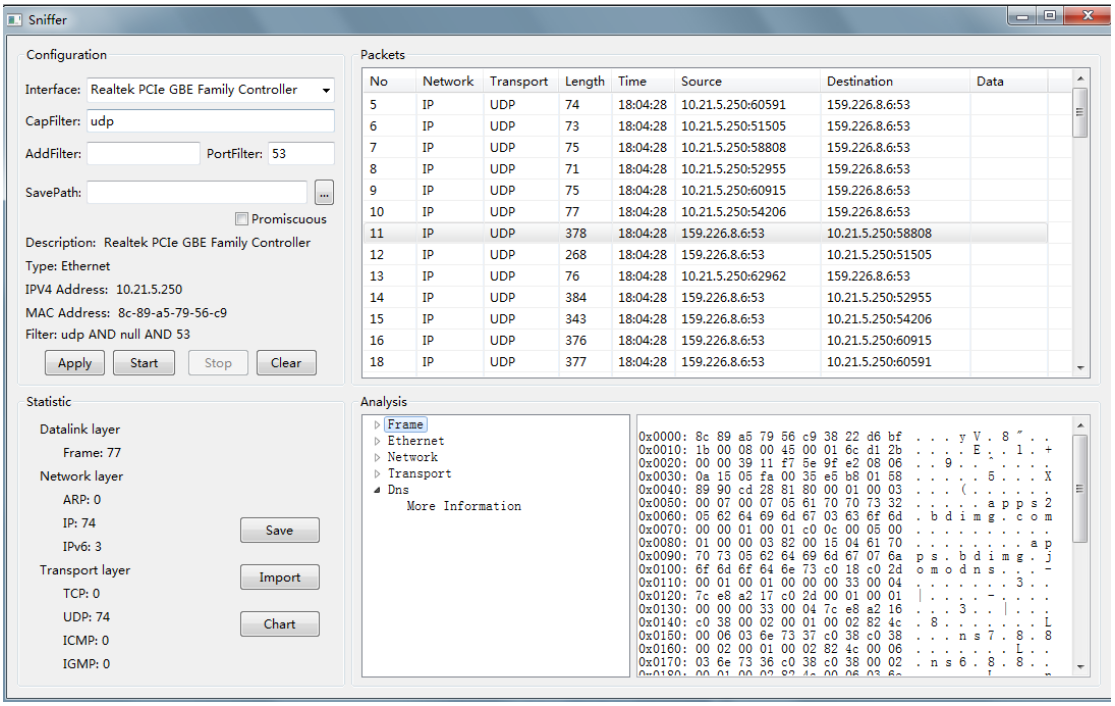
开始抓包，不设置任何过滤器。



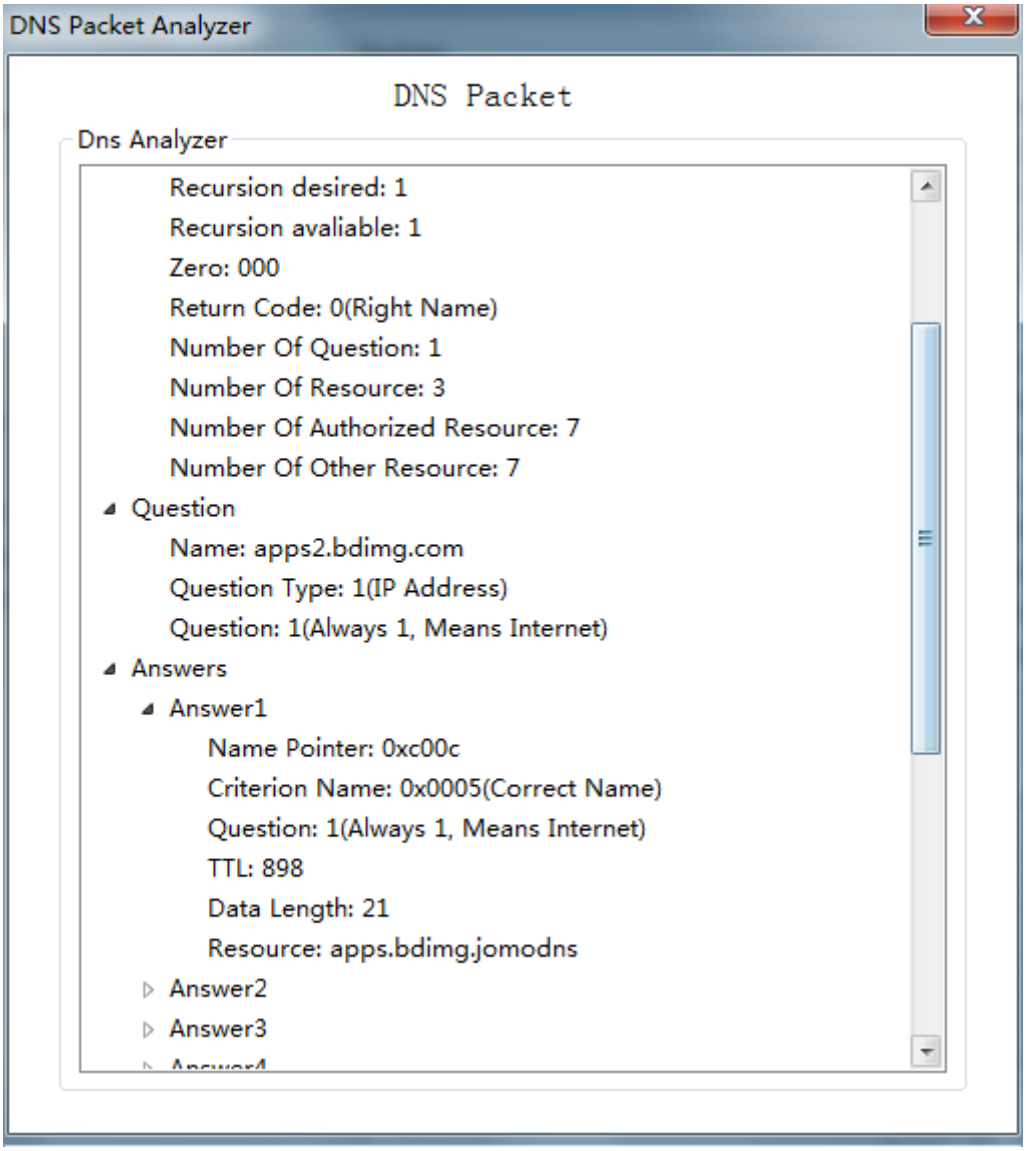
点击 Chart 按钮，显示 Chart 表格。



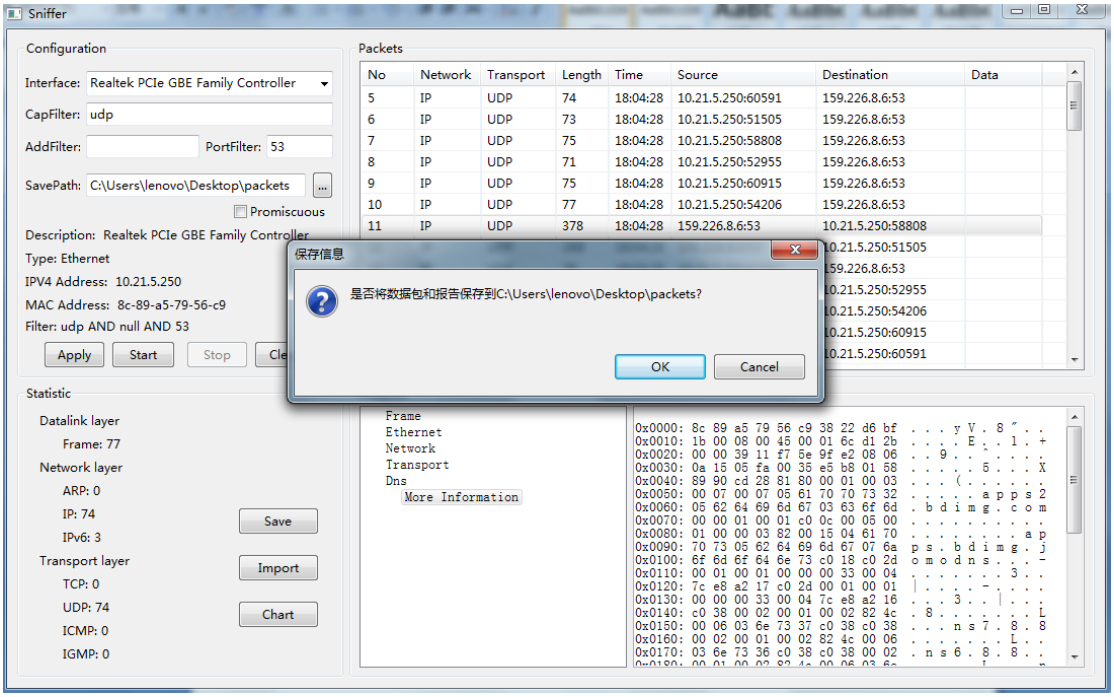
在 53 号端口对 DNS 数据包进行分析。



双击 More information，详细分析 DNS 数据包。


DNS Packet Analyzer
DNS Packet
Dns Analyzer
Recursion desired: 1
Recursion avaliable: 1
Zero: 000
Return Code: 0(Right Name)
Number Of Question: 1
Number Of Resource: 3
Number Of Authorized Resource: 7
Number Of Other Resource: 7
Question
Name: apps2.bdimg.com
Question Type: 1(IP Address)
Question: 1(Always 1, Means Internet)
Answers
Answer1
Name Pointer: 0xc00c
Criterion Name: 0x0005(Correct Name)
Question: 1(Always 1, Means Internet)
TTL: 898
Data Length: 21
Resource: apps.bdimg.jomodns
Answer2
Answer3

存储数据包



Import 导入数据包