项目文档：NetScope 网络嗅探器

许润昕　516021910387

张子瑄　516021910509

2019 年 1 月 4 日

摘要

NetScope 可以对特定网卡流过的数据进行抓取和智能过滤分析，无论是在网络安全还是在黑客攻击方面均扮演了很重要的角色。在本项目中，我们对网络嗅探器的功能和用户界面进行开发，并对每个功能模块分别进行了测试。

我们的网络嗅探器实现了**ARP**、**IPv6**、**IPv4**、**TCP**、**UDP**、**ICMP**、**IGMP**、**DHCP**、**DNS协议数据包**的数据抓取与分析显示，并在此基础上，实现了数据包的**过滤**、**搜索**、**筛选**、**保存到本地**和**从本地打开文件**等功能。除此之外，我们还成功完成了**IP分段报文的重组**，以及**FTP文件传输下的文件嗅探和文件恢复重组**，并实现了友好的用户界面。

我们的实验报告主要由项目介绍、功能实现方法与实例、遇到的问题与解决方法、总结与体会四个部分构成，详细阐述了我们的项目作品的功能示例和实现方法。

1　项目介绍

1.1　概况

本项目NetScope实现了网络嗅探器（Sniffer）的功能与界面实现，同时针对各个功能进行了测试与分析。项目基于Python语言，使用了Scapy库来进行网络数据包的侦听获取，使用了PyQt4库来进行用户界面的实现。应用的界面如图1所示，下面对其各个功能进行简单介绍。

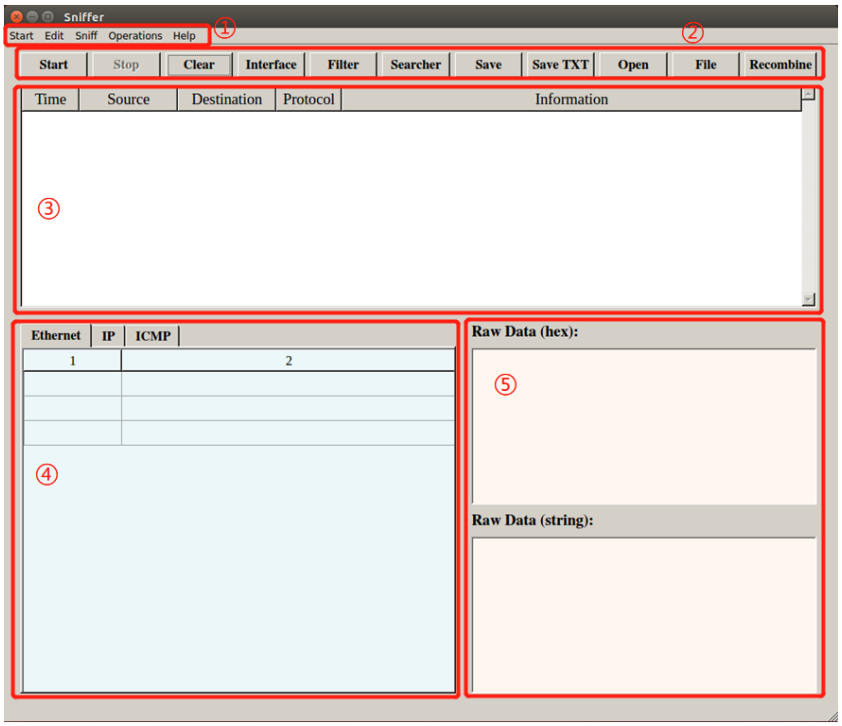


图 1: 功能说明

- 第一部分：菜单栏。所有功能都集成在了菜单栏中供用户选择，同时也有快捷键的实现。
- 第二部分：快捷工具栏。其从左到右对应的功能是：
  - 开始嗅探。
  - 停止嗅探。
  - 清空各个显示窗口信息。
  - 选择嗅探的网络端口。
  - 数据包过滤规则设置。
  - 数据包查询规则设置。
  - 保存选中数据包为pcap格式文件。
  - 保存选中数据包为txt格式文件。
  - 打开pcap格式文件加载数据包。
  - 文件重组。
  - IP报文重组。

- 第三部分：显示主窗口。被嗅探到的数据包主要信息显示在该窗口中。
- 第四部分：数据包详细信息窗口。点击显示主窗口中的数据包，该数据包对应的详细信息，包括数据链路层、IP层、传输层等等的各个字段的信息都会显示在该窗口中。
- 第五部分：原始数据信息窗口。点击显示主窗口中的数据包，该数据包对应的首部之外的数据信息以十六进制的形式与字符串的形式分别显示在这部分窗口中。

1.2 环境配置

该项目中所用到的第三方库与版本参数如表1所示，而测试环境的配置参数如表2所示：

表 1: 第三方库环境配置

第三方库	参数
Scapy	Scapy 2.4.0 (Python2.7) (http://scapy.readthedocs.io)
PyQt4	PyQt4 v4.11 (http://http://pyqt.sourceforge.net/Docs/PyQt4/)

表 2: 测试环境配置

项目	参数
操作系统	Linux ubuntu 14.04
CPU	Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
内存	4G
IP地址(eth0)	10.162.169.130
IP地址(FTP Client:ens33)	192.168.42.130
IP地址(FTP Server)	192.168.42.128

1.3 程序列表

项目的程序列表如图2所示，其中各个文件功能如表3所示。

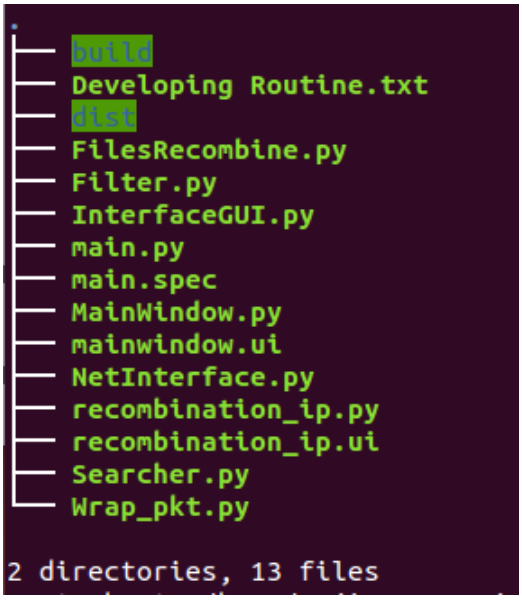


图 2: 项目程序列表

表 3: 项目各文件功能

文件	功能
main.py	主程序文件
NetInterface.py	网卡选择功能的实现
InterfaceGUI.py	网卡选择界面的实现
Wrap_pkt.py	Wrap_pkt类的定义，实现了数据包的头部信息提取
Filter.py	数据包过滤功能的实现
Searcher.py	数据包搜索及筛选功能的实现
recombination_ip.ui	IP分段报文重组GUI的qtdesigner设计文件
recombination_ip.py	IP分段报文重组功能的实现
FilesRecombine.py	FTP文件重组功能的实现
MainWindow.ui	主体用户界面的qtdesigner设计文件
MainWindow.py	MainWindow.ui生成的python代码文件

1.4 主要数据结构

如图3所示为项目用到的主要数据结构，其中myapp为最主要的类，也是项目运行的入口，开始、停

止嗅探，保存数据包成pcap或者txt格式，打开pcap文件加载数据包等等功能的实现也都在这个主要类中。而项目主窗口GUI模块为Ui_MainWindow类，该类被myapp类所继承。myapp类通过调用InterfaceGUI来实现网卡的选择功能及网卡信息更新，而InterfaceGUI类又借助了NetInterface类。同时，myapp类通过调用MyFilterGUI类来实现数据包的过滤功能，通过MySearcherGUI类来实现数据包查询功能，通过调用RecombinationIPGUI来实现IP报文重组功能，通过调用FilesRecombineGUI来是实现文件重组功能。而对于抓取到的数据包，通过Wrap_pkt类进行包装存储，其包含的主要信息如图4所示。

其中，id为嗅探过程中按时间顺序给数据包的一个唯一标识号，便于进行数据处理；time是嗅探到包的时间戳；src是该包的源地址；dst是该包的目的地址；proto是该包的协议类型；info存储的是原始包的summary()函数返回的主要描述信息；packet是数据包的原始数据类型；Ethernet、IP、ARP、IPv6分别存储对应的类型信息；underIP和underUDP存储在特定类型协议下专有的数据字段。

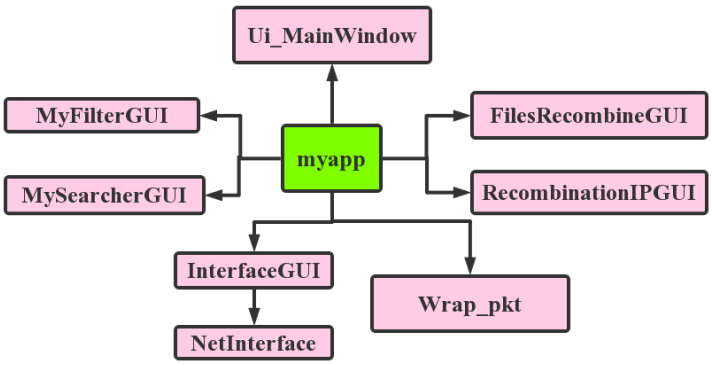


图 3: 项目主要数据结构

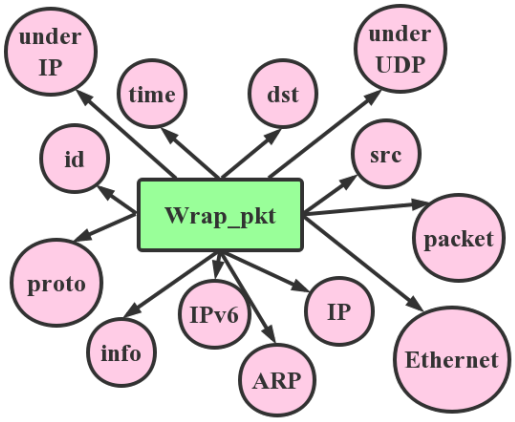


图 4: Wrap_pkt类主要信息

2 功能实现方法与示例

2.1 网卡选择

一个主机上可能有多个网卡，所以我们设计了网卡选择功能，用户可以自由选择嗅探的网卡。我们使用了“/proc/net/dev”文件来获取主机的网卡信息，包括网卡的名字、网卡端口IP地址以及收发的通信量（以MB为单位），并在主函数中调用相关类来获取用户选择网卡的信息（如图5所示。网卡选择界面如图6所示，其中从左到右依次为网卡名字、网卡IP地址、收包通信量大小和发包通信量大小。

```
def do_interface(self):

    if not self.STOP:

        QtGui.QMessageBox.critical(self, "Error", 'You have to stop the sniffer first!')
        return
    else:
        self.iface = InterfaceGUI(self.interface_name)
        self.iface.exec_()  # GUI has to be executed!
        self.interface_name = self.iface.cur_interface
```

图 5: 主函数中关于网卡选择的代码

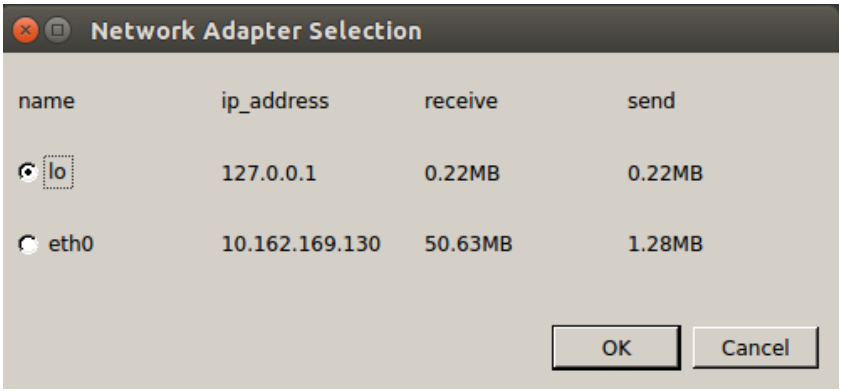


图 6: 网卡选择界面

2.2 数据包嗅探

2.2.1 sniff函数的应用

在数据包的嗅探和解析过程中，我们使用了scapy库提供的功能函数sniff (iface=”iface”, filter=”filter”, count=”count”, prn=callback) 。该函数输入的参数包括字符串格式的网卡名称iface，嗅探数据包的限制条件filter，以及限制嗅探数据包的个数count，返回值是一个数据包对象的列表。另外，也可以在输入的参数中指定回调函数callback(packet),这个函数以嗅探到的数据包作为唯一参数，sniff函数将会在每嗅探

到一个数据包时，将这个数据包送入callback函数进行处理。例如，当我们想要从eth0网卡嗅探10个TCP数据包，并将每个数据包通过function(packet)函数进行处理，那么，对应的函数调用实例为sniff(iface="eth0", count=10, filter="tcp", prn=function)。

2.2.2 Wrap_pkt类提取数据包头部字段

在应用了sniff函数进行数据包抓取之后，我们在回调函数中对每一个数据包创建了一个Wrap_pkt类的实例，用于提取数据包的头部字段，并将每一个字段以字符串的形式储存，便于后续的数据包解析。具体的Wrap_pkt类的定义详见源代码中的Wrap_pkt.py文件。

2.3 数据包报文解析及显示

在我们的NetScope中，实现了IPv4报文、TCP报文、UDP报文（包括DNS、DHCP）、ICMP/IGMP报文、ARP报文、IPv6报文的全面解析和分层显示，每一层之间的关系如图7所示。我们将不同类型的数据报大体上分为如下四层。

- 第一层为Ethernet底层传输信息，并用myapp类的成员函数show_ethernet_info(self,Wrap_pkt)进行数据层解析与显示。
- 第二层为ARP、IPv4、IPv6层的传输信息，我们通过myapp类的成员函数show_second_layer_info(self, Wrap_pkt)进行解析与显示。
- 第三层为传输层数据报信息，包括IP协议下的TCP、ICMP、IGMP、UDP(包括DNS和DHCP)协议，我们通过成员函数show_third_layer_info(self,Wrap_pkt)进行数据包解析与显示。
- 第四层为数据层(Raw Data)，通过成员函数show_raw_content(self,Wrap_pkt)进行显示。

本节内容中，我们将详细介绍每一种协议的数据包的具体解析方式及各字段内容。

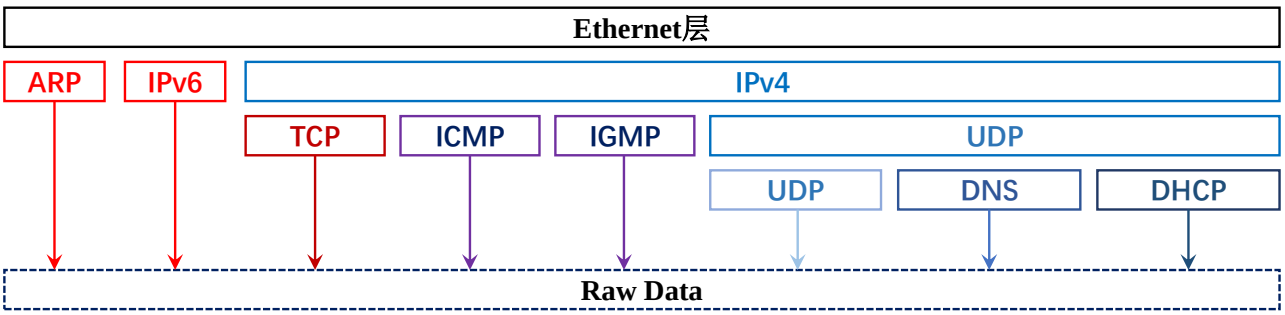


图 7: 数据包的内部分层图示

2.3.1 Ethernet层报文解析

在物理层和数据链路层传输的数据帧头部主要有如下三个字段。

源物理地址(Source Physical Address)： 长度为6字节，代表数据帧源物理地址。

目的物理地址(Destination Physical Address)： 长度为6字节，代表数据帧目的物理地址。

数据帧类型(Protocol Type)： 长度为2字节，代表数据帧的类型。在sniff函数的嗅探结果中，协议类型用一个long类型的整数变量type表示。以太网中常用的协议类型有ARP：0x0806，IPv4:0x0800，IPv6:0x86dd，PPPoE:0x8864，MPLS Label:0x8847等等。在解析包的过程中，我们可以根据type变量的数值判断数据帧的类型。本字段的值也可以作为后面对更高层数据的解析过程中对协议种类的一个判断。

如图8所示为一个TCP数据包的Ethernet层数据的解析的实例。

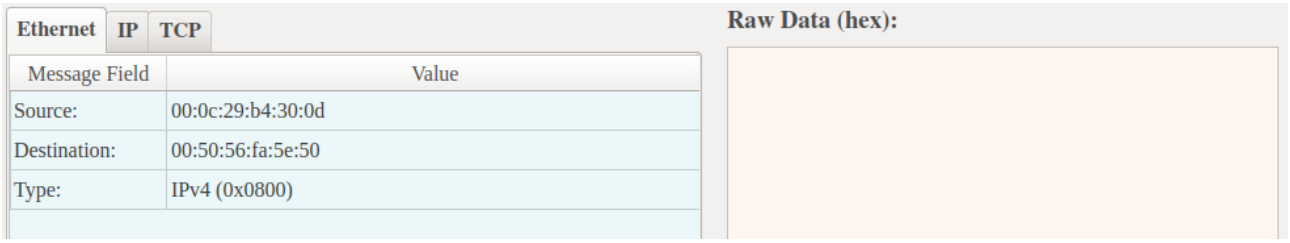


图 8: Ethernet层数据解析实例

2.3.2 ARP报文解析

ARP协议的报文头部主要含有以下9个字段。

硬件类型(Hardware Type)： 长度为2字节，代表硬件地址的类型，值为1代表以太网MAC地址。

协议类型(Protocol Type)： 长度为2字节，代表需要映射的协议地址类型，值为0x800代表IPv4地址。

硬件地址长度(Hardware Address Length)： 长度为1字节，代表硬件地址的长度，如果硬件地址类型为以太网MAC地址，以字节为单位，那么长度应当为6字节。

协议地址长度(Protocol Address Length)： 长度为1字节，代表需要映射的协议地址的长度，如果协议地址类型为IPv4地址，那么长度应当为4字节。

操作类型(Operation Type)： 长度为2字节，1代表ARP请求，2代表ARP应答。

源物理地址(Source Physical Address)： 代表ARP请求映射的源物理地址。

目的物理地址(Destination Physical Address)： 代表ARP请求的目的物理地址。

源协议地址(Source Protocol Address)： 代表ARP请求的源协议地址。

目的协议地址(Destination Protocol Address)： 代表ARP请求的目的协议地址。

在sniff函数的嗅探结果中，前五个字段均用整数变量来表示，后面四个字段用字符串变量表示。如图9所示为一个ARP数据包解析实例。

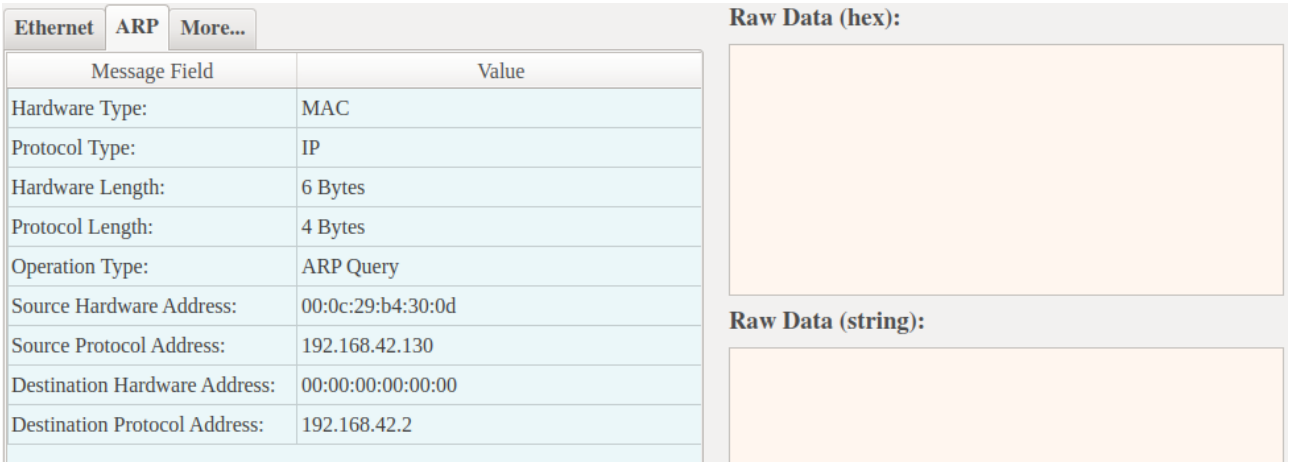


图 9: ARP数据包解析实例

2.3.3 IPv6报文解析

IPv6报文的头部共有8个主要的报文字段可供我们解析，下面我们将详细叙述这些字段的含义和在scapy库中的具体表示形式和解析方法。

版本号(Version)： 长度为4比特，用于标识IP报文的版本，IPv4为0100，IPv6为0100。在sniff函数的嗅探结果中，这一字段用一个int类型的变量version表示，当version=4时，代表IPv4报文，当version=6时，代表IPv6报文。

流量类型/优先级(Traffic Class)： 长度为8比特，与IPv4报文中的tos字段类似，4-7位代表服务类型。在sniff函数的嗅探结果中，这一字段用一个long类型的变量tc表示。

流标记(Flow Label)： 长度为20比特，用来标记报文的数据流类型，以便在网络层区分不同的报文。在sniff函数的嗅探结果中，这一字段用一个long类型的变量fl表示。

有效载荷长度(Payload Length)： 长度为20比特，代表IPv6报文除头部外其他部分的数据总长度，单位为字节。在sniff函数的嗅探结果中，这一字段用一个long类型的变量pl表示。

下一包头(Next Header)： 长度为8比特，用来标识当前报头（或者扩展报头）的下一个头部类型。在sniff函数的嗅探结果中，这一字段用一个long类型的变量nh表示。在这一部分，我们实现了以下几种常用的Next Header的解析。

- $nh = 0$: IPv6 Hop-by-Hop Option
- $nh = 1$: ICMP
- $nh = 2$: IGMP
- $nh = 4$: IPv4 Encapsulation
- $nh = 6$: TCP
- $nh = 17$: UDP
- $nh = 41$: IPv6 Encapsulation
- $nh = 43$: IPv6 Routing Header
- $nh = 44$: IPv6 Fragment Header
- $nh = 50$: Encap Security Payload
- $nh = 51$: Authentication Header
- $nh = 58$: IPv6 ICMP

跳数限制(Hop Limit)： 长度为8比特，类似于IPv4协议中的生存周期，用于限定数据包的最大转发次数，防止出现因为环路的存在而被无限转发的情况。在sniff函数的嗅探结果中，这一字段用一个long类型的变量hlim表示。

源地址(Source)： 字符串形式的变量，用于表示数据包的源IPv6地址。

目的地址(Destination)： 字符串形式的变量，用于表示数据包的目的IPv6地址。

如图10所示为一个IPv6数据包的解析实例。

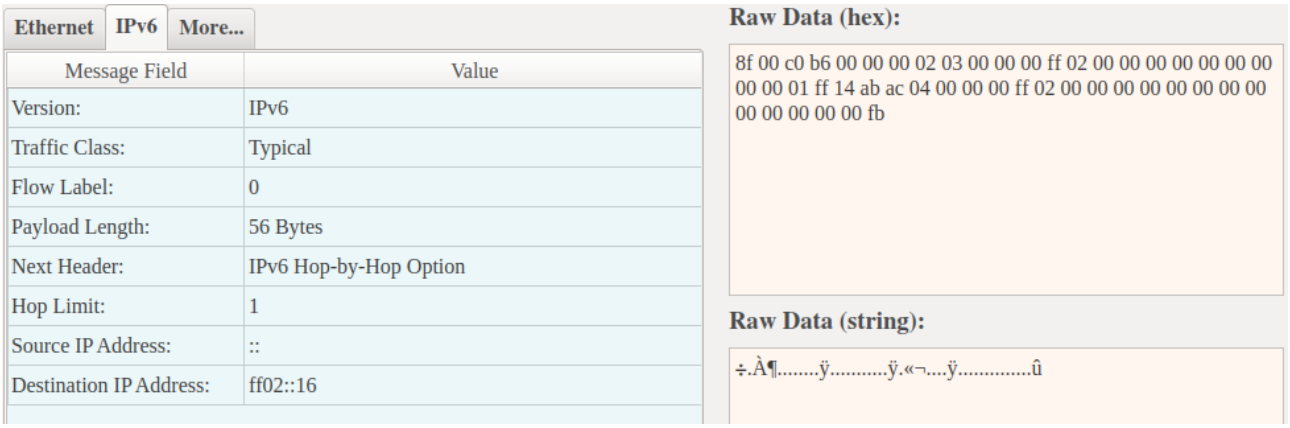


图 10: IPv6数据包解析实例

2.3.4 IPv4报文解析

IPv4报文的头部共有14个报文字段可供我们解析，下面我们将详细叙述这些字段的含义和在scapy库中的具体表示形式和解析方法。

版本号(Version)： 长度为4比特，用于标识IP报文的版本，IPv4为0100，IPv6为0100。在sniff函数的嗅探结果中，这一字段用一个int类型的变量version表示，当version=4时，代表IPv4报文，当version=6时，代表IPv6报文。

IP报头长度(Internet Header Length)： 长度为4比特，用于表示IP报文头部的长度。值得注意的是，这个字段的每一个比特代表4个字节的IP报头长度。因此，头部的最长长度为4Byte×15=60Byte。在sniff函数嗅探结果中，这一字段用一个int类型的变量ihl表示，将ihl变量值乘4即得到头长度的字节数。

服务类型(Type of Service)： 长度为8比特，用于定义IP包传输过程中的优先级类型和服务类型。根据课本上的描述，这8个比特的前三位表示传输的优先级类型，但现阶段已停止使用。第4至7位代表传输需求和服务类型，具体来说，第4位置1代表最小延时传输，第5位置1代表最大流量传输，第6位置1代表最大可靠性传输，第7位置1代表最小费用传输。第8位为保留位0。在sniff函数中，这一部分被用一个long类型的整数变量tos表示。我们先用python语句tos_str=':08b'.format(packet.IP['tos']):08b'将这个整数转化为一个8位固定长度的二进制字符串tos_str，再提取4-7位进行分析即可。

数据包总长度(Length)： 长度为16比特，每一个比特代表一个字节，因此IPv4报文的最大长度为65535字节。在sniff函数的嗅探结果中，这个字段同样也被一个long类型的整数变量len表示。

IP包序号(ID)： 长度为16比特，用于表示在IP报文分段之后，哪些数据包属于同一个IP报文。在sniff函数的嗅探结果中，这个字段也被用一个long类型的整数变量id表示。

DF标志位(DF Flag)： 1比特标志位，含义为Don't Fragment，当此标志位设置为1时，代表此数据报不允许被分段。

MF标志位(MF Flag)： 1比特标志位，含义为More Fragment，当此标志位设置为1时，代表此数据包之后还有数据包和自己属于同一个报文。在DF和MF标志位之前，有一位保留位0。在sniff函数的嗅探结果中，这三个比特被整体视为一个int类型的整数变量flags。因此当flags=0时，DF=0，MF=0；当flags=1时，DF=0，MF=1；当flags=2时，DF=1，MF=0。

分段偏移量(Fragment)： 长度为13比特，用于在数据报分片时，代表这些分片数据包的相对位置。，接收端靠此来组装IP报文。由于该字段长度只有13比特，为了能够完整的表示最长65535字节的IP数据报，需要要求分片IP报文长度必须为8字节的整数倍。在sniff函数的嗅探结果中，分片偏移量用long类型的整数来表示。

生存时间(Time to Live)： 长度为8比特，用于设置路由器对此IP包的最大转发次数，防止因为环

路的存在导致数据包无限被转发。在sniff函数的嗅探结果中，这部分用一个long类型的整数变量ttl表示。

协议类型(Protocol)： 长度为8比特，代表了IP报文上层承载的协议类型。在sniff函数的嗅探结果中，这部分用一个long类型的整数变量proto表示。常见的协议类型有ICMP:1,IGMP:2,TCP:6,UDP:17等等。

头部校验和(Checksum)： 长度为16比特，用于检验收到的报文是否出现错误。在sniff的嗅探结果中，这部分用一个long类型的整数变量chksum表示。

源IP地址(Source IP)： 数据报的源IP地址，在sniff的嗅探结果中，用一个字符串变量src表示。

目的IP地址(Destination IP)： 数据报的目的IP地址，在sniff的嗅探结果中，用一个字符串变量dst表示。

可选项(Options)： 长度为32比特，不足时用0补充，在可选项中，常见的字段有选项表结束、无操作、安全选项、松原地址和记录路由、紧源地址和记录路由、记录路由、流标记、时间戳、路由器警告等等。在sniff函数中，这些字段被用一个列表类型的变量options表示，列表中的每一个元素为一个二元组，分别代表选项的名称和对应的值，在解析的过程中，我们只需要将每一个元组提取并显示即可。

如图11所示为一个TCP数据包的IP层数据解析实例。

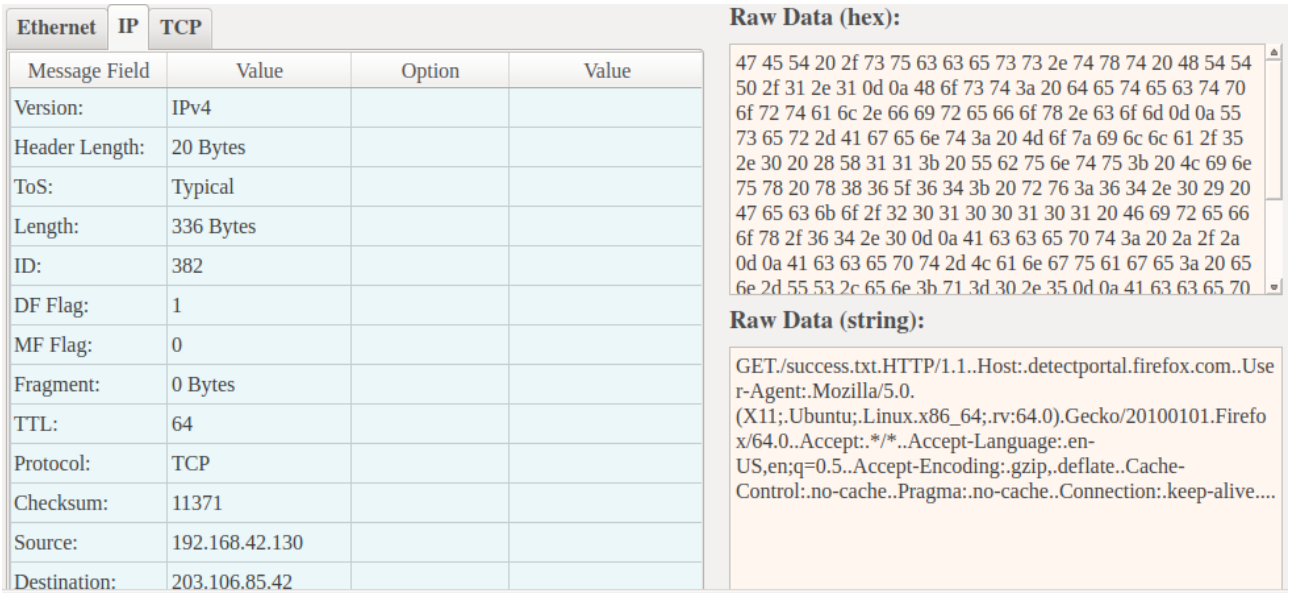


图 11: IPv4数据包解析实例

2.3.5 TCP报文解析

TCP报文的头部共有9个主要报文字段可供解析，下面我们将详细叙述这些字段的含义和在scapy库中的具体表示形式和解析方法。

源端口(Source Port)： TCP数据包的源端口，长度为16比特。在sniff函数的嗅探结果中，用一个long类型的整数变量sport表示。

目的端口(Destination Port)： TCP数据包的目的端口，长度为16比特。在sniff函数的嗅探结果中，用一个long类型的整数变量dport表示。

发送序号(seq)： 代表TCP数据包的发送序号，长度为32比特，在sniff函数的嗅探结果中，用一个long类型的整数变量seq表示。

确认序号(ack)： 代表TCP数据包的确认序号，长度为32比特，在sniff函数的嗅探结果中，用一个long类型的整数变量ack表示。

首部长度(dataofs)： 代表TCP数据包的首部长度，长度为4比特，在sniff函数的嗅探结果中，用一个long类型的整数变量dataofs表示。

标志位(flags)： 每一个TCP数据包都会有6个固定的标志位，用于TCP连接的建立与释放。这些标志位分别为URG、ACK、PSH、RST、SYN和FIN。在sniff函数的嗅探结果中，这6个比特被解释成了一个整数变量flags，因此，我们只需要使用python语句':06b'.format(packet.underIP['flags'])将变量flags转化为一个6位的二进制字符串，之后分别提取字符串中的每一位，这样就可以的到6个标志位的值。

窗口大小(Window Size)： 代表TCP传输协议的窗口大小，长度为16比特，最大窗口大小位65535，在sniff函数的嗅探结果中，用一个long类型的整数变量window表示。

首部校验和(Checksum)： 代表TCP数据包的首部校验和，长度为16比特，在sniff函数的嗅探结果中，用一个long类型的整数变量chksum表示。

可选项(Options)： 在TCP报文的可选项中，常见的选项有选项表结束、无操作、最大报文段长度(MSS)、窗口扩大因子、SACKOK选项、SACK选项、时间戳等等。在sniff函数的嗅探结果中，可选项用一个list类型的变量options表示，每一个列表的元素是一个二元组，二元组的两个分量分别代表选项的名称和选项的值。另外，在选项中的Padding字段负责填充0值以使得包头长度为32的整数倍。

如图12所示为一个TCP数据包的TCP层数据解析实例。

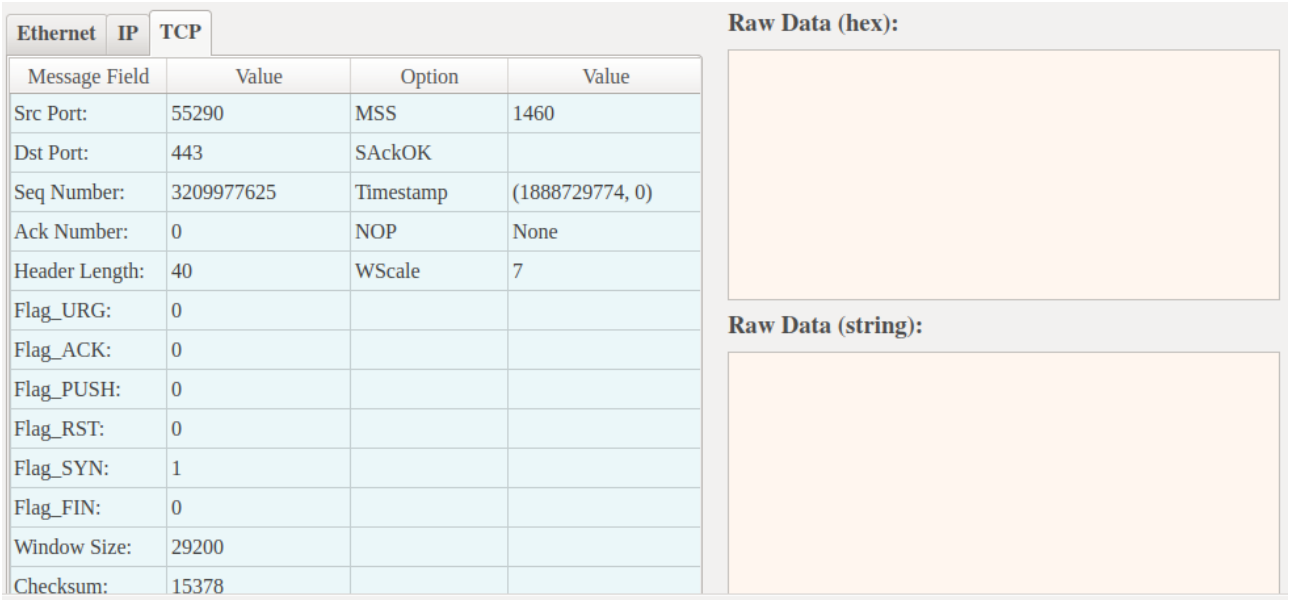


图 12: TCP数据包解析实例

2.3.6 ICMP报文解析

ICMP报文的头部共有5个主要报文字段可供解析，下面我们将详细叙述这些字段的含义和在scapy库中的具体表示形式和解析方法。

类型(Type)： 代表ICMP数据包的操作类型，长度为1字节，在sniff函数的嗅探结果中，用一个long类型的整数变量type表示。常见的ICMP操作类型有Echo-Request 'ping'(Type = 0)，Echo-Reply (Type = 0)，Unreachable (Type = 3)，Overtime(Type = 11), Timestamp Request(Type = 13)，Timestamp Reply(Type = 14)等等。

代码(Code)： 代表ICMP数据包的操作码或确认码，长度为1字节，在sniff函数的嗅探结果中，用一个long类型的整数变量code表示。code一般的含义是对type的一个更加具体的解释。例如当type=3时，代表不可到达，对应不同的code有以下几种情况：Network Unreachable (Code = 0)，Host Unreachable (Code = 1)，Protocol Unreachable (Code = 2)，Port Unreachable (Code = 3)等等。当type=11时，代表请求超时，对应不同的code有：Transmission Overtime (Code = 0)，Fragment Overtime (Code = 1)。

校验和(Checksum)： 代表ICMP数据包的头部校验和，长度为16比特，在sniff函数的嗅探结果中，用一个long类型的整数变量chksum表示。

标识符(ID)： 代表ICMP数据包的标识符，长度为16比特，用于标识本ICMP进程，但仅适用于回显请求和应答ICMP报文，对于目标不可达ICMP报文和超时ICMP报文等，该字段的值为0在sniff函数的嗅探结果中，用一个long类型的整数变量id表示。

序列号(Sequence Number)： 代表TCP数据包的确认序号，长度为32比特，在sniff函数的嗅探结果中，用一个long类型的整数变量ack表示。

如图13所示为一个ICMP数据包的数据解析实例。

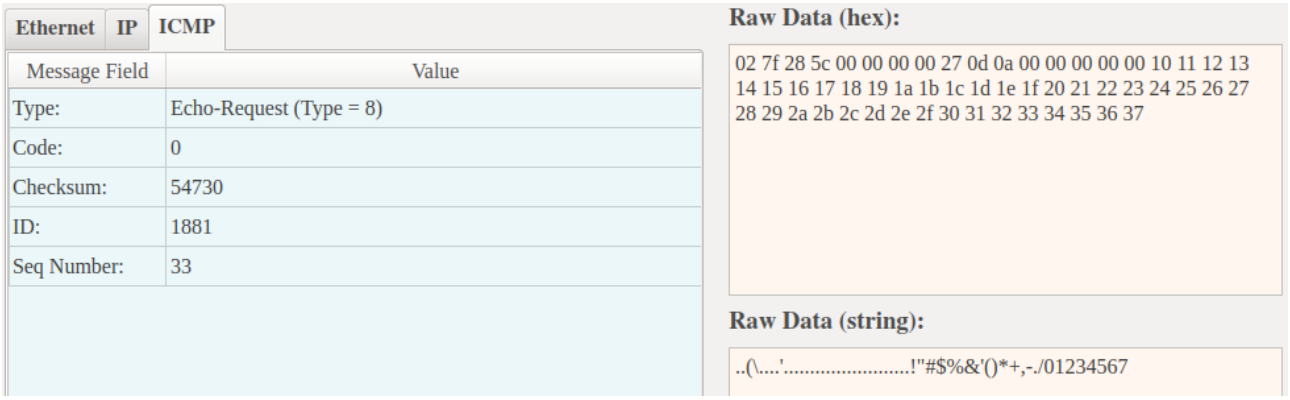


图 13: ICMP数据包解析实例

2.3.7 IGMP报文解析

在scapy库中，没有提供对IGMP组播协议的报头解析，因此，我们借助IP层的Raw Data自行解析了IGMP数据包。在IGMP协议中，我们选取了4个主要的报头字段进行解析。具体的解析过程见Wrap_pkt类的get_IGMP_information成员函数，在该成员函数中，我们将IGMP数据包的IP层Raw Data解析为了4个主要的报文字段type,mrt,checksum,group_addr。

类型(Type)： 此字段代表了IGMP数据包的类型，长度为8比特，具体的可取值及其对应的含义如表4所示。

表 4: IGMP类型字段解析

type	含义	英文含义
0x11	成员关系查询	Membership Query
0x12	IGMP v1 成员关系报告	Version 1 Membership Report
0x16	IGMP v2 成员关系报告	Version 2 Membership Report
0x17	离开组	Leave Group
0x22	IGMP v3 成员关系报告	Version 3 Membership Report

最大响应时延(MRT)： 长度为8比特，只在成员关系查询类型的报文中有效，代表的含义是主机必

须在最大响应时间到达之前发出成员关系报告的报文。

头部校验和(Checksum)： 长度为16比特，为IGMP数据报的头部校验和。

组地址(Group Address)： 长度为32比特，用于存放将要查询的组播组的地址。

如图14所示为一个IGMP数据包的数据解析实例。

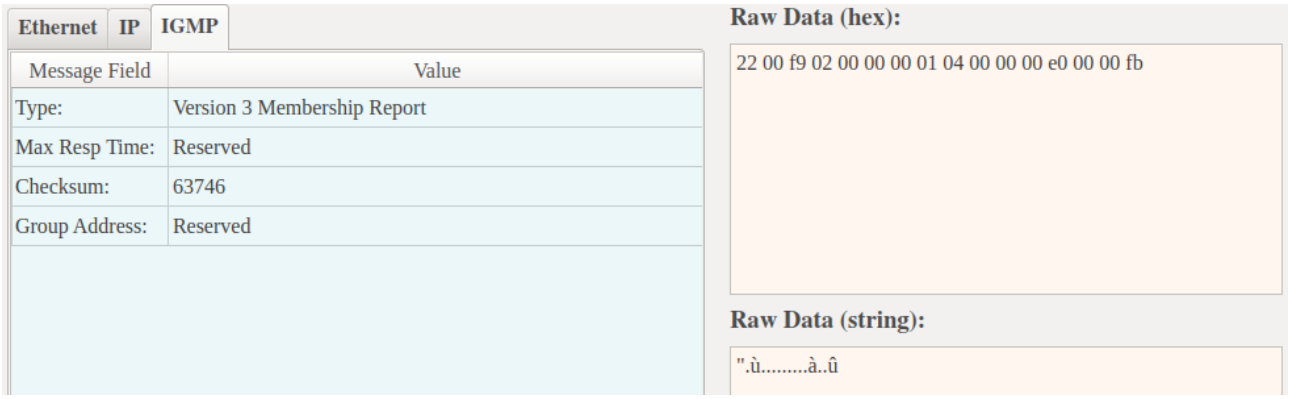


图 14: IGMP数据包解析实例

2.3.8 UDP报文解析

根据图7所示的结构，DNS和DHCP协议均属于UDP协议框架下的内容。在这一小节中，我们先讨论普通UDP数据包的抓取和解析。 UDP报文的头部共有4个主要报文字段可供解析，下面我们将详细叙述这些字段的含义和在scapy库中的具体表示形式和解析方法。

源端口(Source Port)： 用于表示UDP数据传输的源端口号，长度为2字节。在sniff函数的嗅探结果中表示为一个long类型的整数变量sport。

目的端口(Destination Port)： 用于表示UDP数据传输的目的端口号，长度为2字节。在sniff函数的嗅探结果中表示为一个long类型的整数变量dport。

长度(Length)： UDP报文的总长度，单位为字节，长度为2字节（16比特），因此，UDP报文最长长度为65535字节。在sniff函数的嗅探结果中表示为一个long类型的整数变量len。

校验和(Checksum)： UDP报文的头部校验和，长度为2字节，可用来检查数据传输是否出错。在sniff函数的嗅探结果中表示为一个long类型的整数变量chksum。

一个典型的UDP数据包解析的例子如图15所示。

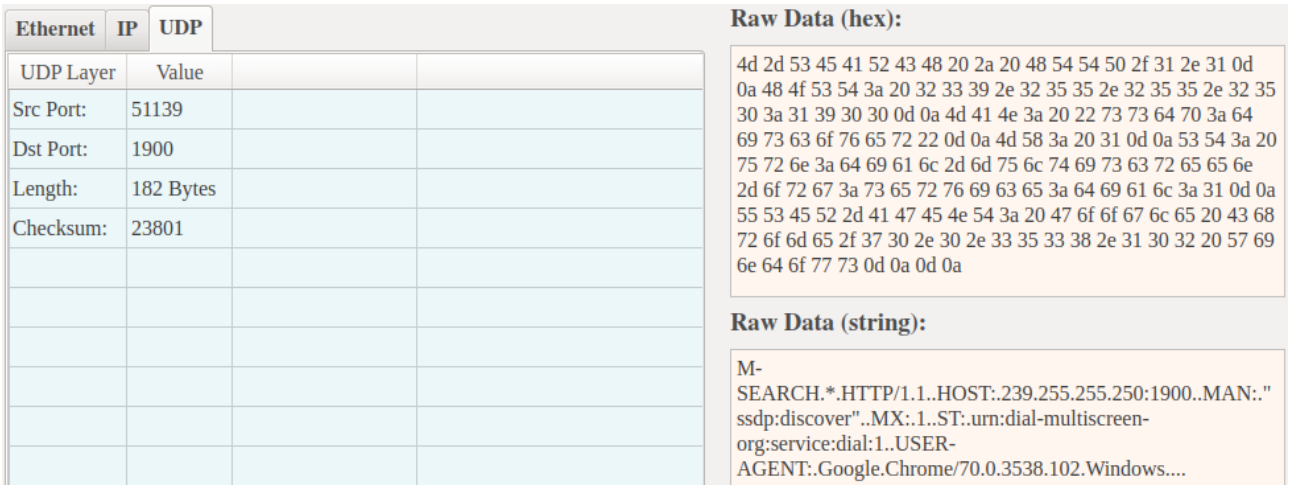


图 15: UDP数据包解析实例

2.3.9 DNS报文解析

DNS协议常用于网站域名和URL的相互查询，是UDP协议的一种。除了必需的UDP协议的4个字段之外，DNS报文的头部还有8个主要报文字段可供解析，下面我们将详细叙述这些字段的含义和在scapy库中的具体表示形式和解析方法。

序号(ID)： 表示DNS请求的序列号，用于区别不同的DNS请求和响应，长度为2字节。

种类(type)： 长度为1比特，用于表示DNS数据报的类型，0表示请求，1表示响应，对应DNS协议flags里面的qr字段。

操作码(Operation Code)： 一个整数用于表示请求查询的类型，长度为4比特，实现解析的类型有DNS Standard Query(opcode=0)、DNS Reverse Query(opcode=1)、Server Status Request(opcode=2)。

响应码(Return Code)： 一个整数用于表示响应的结果，长度为4比特，实现解析的类型有No Error(rcode=0)、Server Failure(rcode=2)、Name Error(opcode=3)。

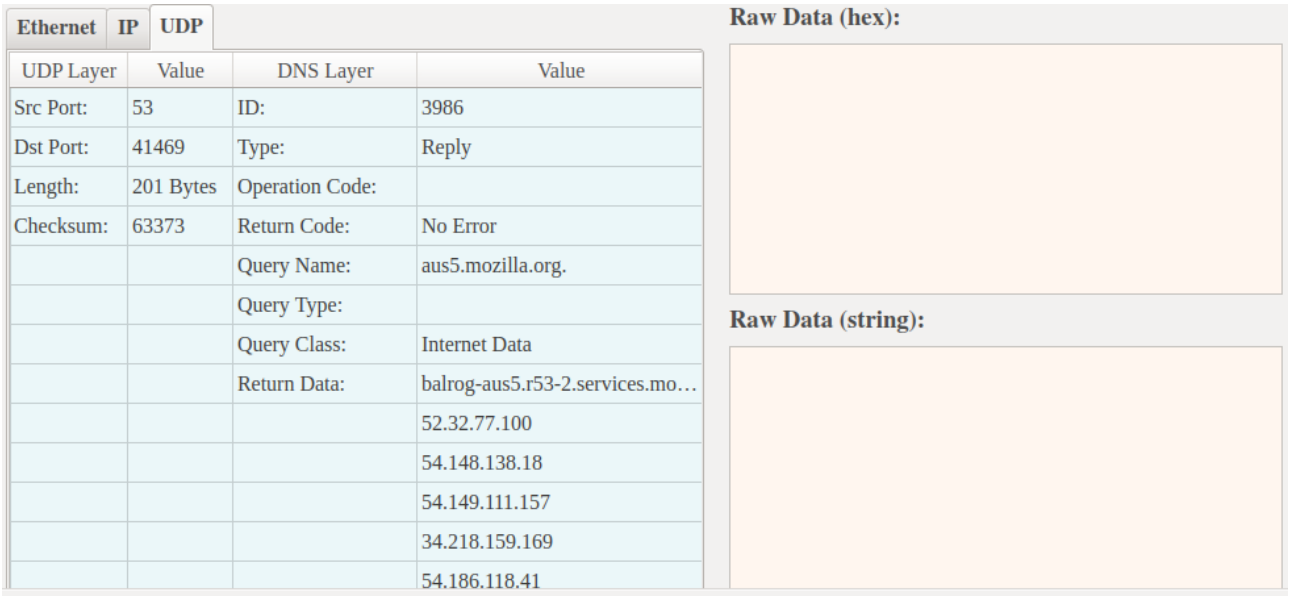
查询名(Query Name)： 用于表示查询的输入。

查询目标(Query Type)： 用于表示查询的目标，我们的NetScope实现的解析包括IPv4查询(qtype=1)、DNS服务器查询(qtype=2)、IPv6查询(qtype=28)、主机信息查询(qtype=13)、域名查询(qtype=12)。

查询类别(Query Class)： 用于表示查询的类别，即查询是要在哪一种网络上，最常用的就是互联网Internet(qclass=1) 。

查询结果(Return Data)： 用于表示查询的结果。

一个典型的DNS数据包解析的例子如图16所示。



UDP Layer	Value	DNS Layer	Value
Src Port:	53	ID:	3986
Dst Port:	41469	Type:	Reply
Length:	201 Bytes	Operation Code:	
Checksum:	63373	Return Code:	No Error
		Query Name:	aus5.mozilla.org.
		Query Type:	
		Query Class:	Internet Data
		Return Data:	balrog-aus5.r53-2.services.mo…
			52.32.77.100
			54.148.138.18
			54.149.111.157
			34.218.159.169
			54.186.118.41

图 16: DNS数据包解析实例

2.3.10 DHCP报文解析

DHCP协议可以帮助用户自动获取IP地址，不再需要用户进行手动设置。除了UDP协议的4个字段之外，我们在DHCP数据报中另外解析出了3个主要字段如下。

数据报类型(Message Type)： 用于表示数据报的类型，实现解析的类型有DHCP Discover(type=1)、DHCP Offer(type=2)、DHCP Request(type=3)、DHCP Decline(type=4)、DHCP ACK(type=5)、DHCP NAK(type=6)、DHCP Release(type=7)、DHCP Inform(type=8)。

获取的地址(Requested Address)： 用于表示DHCP请求得到的地址。

主机名称(Host Name)： 用于表示得到IP地址的主机名称，以便于进行标识。

一个典型的DHCP数据包解析的例子如图17所示。

2.4 数据包过滤

在调用的抓包函数sniff中，有filter参数，所以我们通过一个filter界面来获取用户所输入的过滤规则，

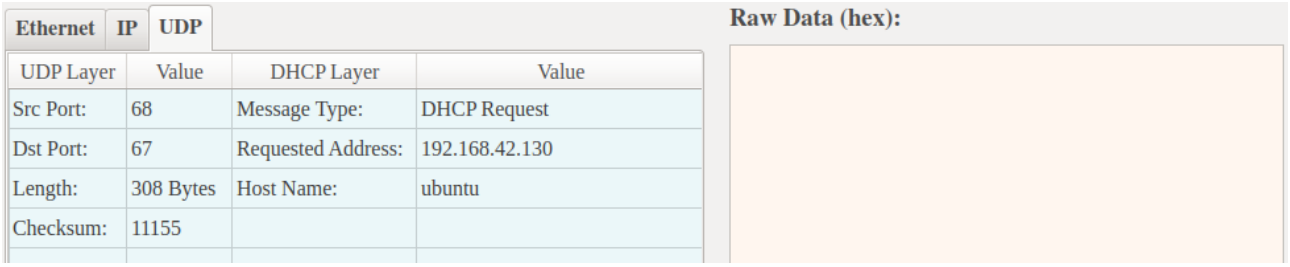


图 17: DHCP数据包解析实例

之后传给sniff函数，从而达到过滤包的目的。Filter的信息包括数据报协议、源端IP地址、目的端IP地址、源端口号、目的端口号，可以填写一个字段，也可以同时填写多个字段，如果所有的字段都不填写则相当于不设置过滤条件。Filter类的主要处理代码如图18所示，首先通过循环检测五个相关字段，并通过**正则表达式或isdigit()函数**来检测输入的合法性。对于非法输入，弹出Error窗口提醒用户重新输入；对于合法的输入，将字符串拼接在一起作为最后返回的过滤规则。

```
        if key == 'Protocol':
            if self.checkproto(tmp):
                new_value[key] = tmp.lower()
            else:
                new_value[key] = ''
                if tmp != '':
                    QtGui.QMessageBox.critical(self, "Error", 'Please put in the right protocol!')
                    return

        elif key == 'Source':
            if self.checkip(tmp):
                new_value[key] = 'src host ' + tmp
            else:
                new_value[key] = ''
                if tmp != '':
                    QtGui.QMessageBox.critical(self, "Error", 'Please put in the right source!')
                    return
        elif key == 'Destination':...
        elif key == 'Src Port':
            if not tmp.isdigit():
                new_value[key] = ''
                if tmp != '':
                    QtGui.QMessageBox.critical(self, "Error", 'Please put in the right source port!')
                    return
            else:
                new_value[key] = 'src port ' + tmp
        elif key == 'Dst Port':...

    for key in new_value:
        if new_rule == '':  # '' add directly . no problem
            new_rule = new_rule + new_value[key]
        elif new_value[key] != '':
            new_rule = new_rule + ' and ' + new_value[key]
```

图 18: Filter类主要代码

以下是一个使用filter的例子。首先其filter界面如图19，我们在其中protocol字段部分输入icmp，表示过滤掉所有不是icmp协议的数据包。通过命令行窗口中执行”ping www.baidu.com”命令，得到其抓包结果如图20所示。可以看到，所嗅探到的数据包都是ICMP协议的。更具体的例子可以参见文件夹中的“filter-example.gif”。

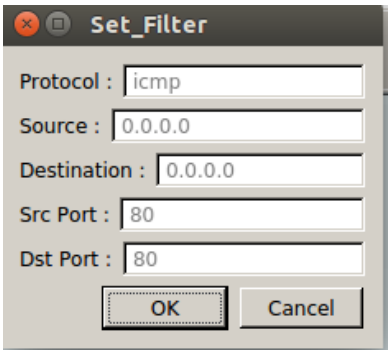


图 19: filter界面

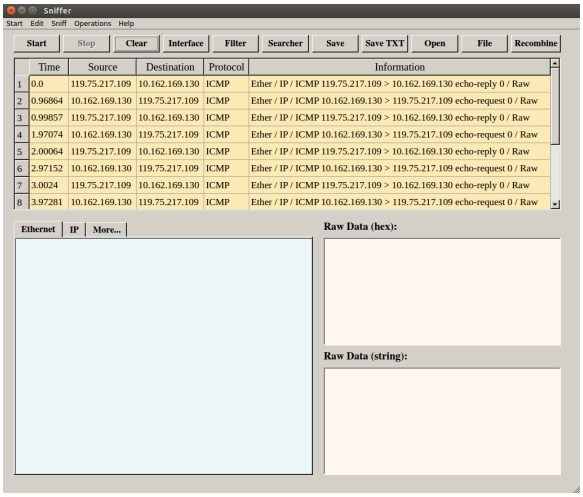


图 20: 过滤非ICMP数据包

2.5 数据包查询

数据包查询与数据包过滤的实现类似，但过滤是在嗅探的过程中，只抓取符合过滤条件的数据包；而查询是在抓包完成之后，查询符合相关条件的包，且可以进行多次不同查询，每次都会显示相应的查询内容，便于用户在众多抓取的数据包中显示出想要的数据包。数据包的查询同样可以根据数据包协议、源端IP地址、目的端IP地址，还可以根据数据包的ID大小来查询某个ID区间的数据包。该ID并非数据包内容中的ID，而是抓包过程中，给每个包自动按抓取的先后顺序自动加上的一个唯一的ID标识符。

通过弹出窗口获取用户输入的search条件之后，在程序内部维护的抓包数据库字典(self.w_pkts)中通过遍历搜索，将所有符合条件的包的ID放到结果ID列表(search_result_id)中，同时通过调用self.clear_text()函数将显示窗口清空，之后将符合search条件的包重新显示在主窗口中，达到搜索的目的。用户可以多次改变search条件，由于抓取的包始终保存在数据库字典中，所以多次搜索都可以成功。具体主要实现代码如图21所示。

以下是一个一个具体例子。在图6中已经看到，我们一共有两个网卡，我们首先选择eth0网卡，其IP地址为10.162.169.130，启动抓包之后其结果如图22所示，可以看到其抓到的包都与10.162.169.130相关。之后我们选择lo网卡，其IP地址为127.0.0.1，是本机的环回接口，启动抓包之后其结果如图23所示，可以看到其抓到的包都与127.0.0.1相关。更具体的例子可以参见文件夹中的”interface-example.gif”演示。

以下用一个例子来示范。首先我们开启嗅探，抓取到一定数量报文之后，发现里面包括dns、TCP等报文。如图24我们使用搜索条件：协议为dns，源地址IP为202.112.26.40，来进行搜索符合条件的报文。点击确定按钮之后，如图25所示，主窗口中显示的数据包都为符合条件的报文，而原来的TCP报文等等已经没有显示出来了，达到了搜索的要求。更具体的例子可以参考文件夹中的”searcher-example.gif”。

```
        search_result_id = []

        for wpkt_id in self.w_pkts:
            w_pkt = self.w_pkts[wpkt_id]
            accept_flag = True
            for key in self.searcher_rule:
                if self.searcher_rule[key] == '':
                    continue

                if (key == 'Protocol' and w_pkt.proto != self.searcher_rule[key]) \
                        or (key == 'Source' and w_pkt.src != self.searcher_rule[key]) \
                        or (key == 'Destination' and w_pkt.dst != self.searcher_rule[key]):

                    accept_flag = False
                    break
                elif key == 'Begin_id' and w_pkt.id < int(self.searcher_rule[key]):
                    accept_flag = False
                    break
                elif key == 'End_id' and w_pkt.id > int(self.searcher_rule[key]):
                    accept_flag = False
                    break

            if accept_flag:
                search_result_id.append(wpkt_id)

        self.clear_text()
        self.id_list = []
        for id in search_result_id:
            self.show_packets(self.w_pkts[id])
            self.id_list.append(self.w_pkts[id].id)
```

图 21: 搜索功能主要代码

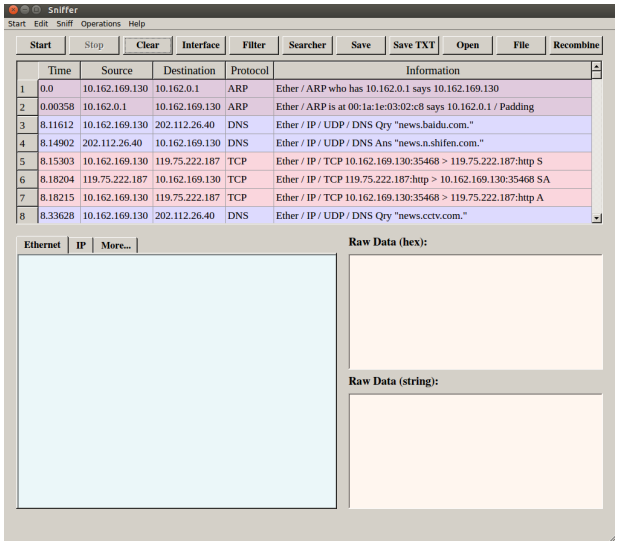


图 22: 嗅探eth0网卡

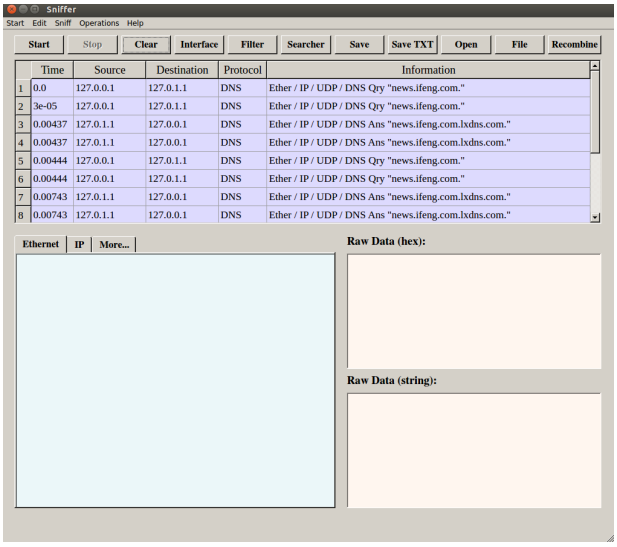


图 23: 嗅探lo网卡

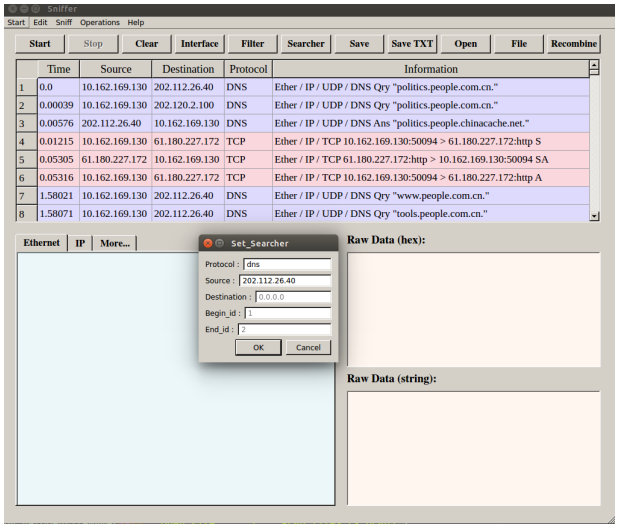


图 24: 设置搜索条件

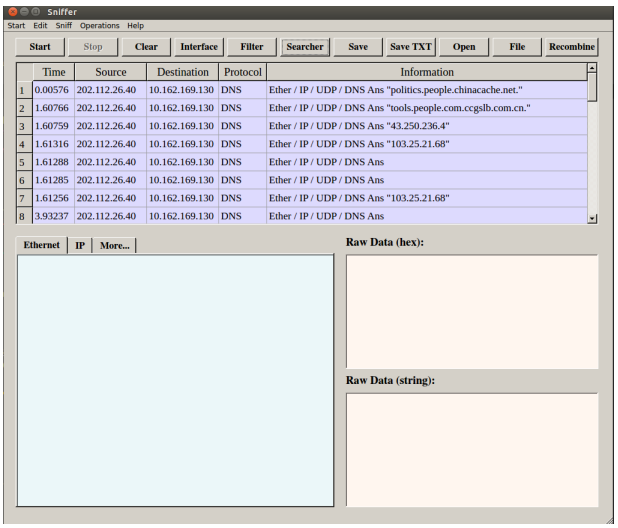


图 25: 搜索结果

2.6 数据包保存与加载

首先，对于数据包的保存，需要允许用户点击选择将要保存的数据包。由于我们的显示窗口使用的是QtGui.QTableWidget空间，因此我们可以通过QtGui.QTableWidget.selectedIndexes()来获取用户选中的行数，并转换成具体数据包的ID，从而在程序维护的数据包字典中对应地取出相应的数据包来进行保存处理。

程序支持两种保存格式：pcap和txt。pcap格式的文件，可以通过scapy中的PcapWriter类来进行实现，将选中的数据包保存为一个pcap文件；而对于txt格式文件的保存，则通过python普通的文件输出流即可实现，文件内容即为显示在主窗口中的数据包的内容。pcap文件便于嗅探器程序加载之前的数据包进行处理，但不可读；而txt文件则将数据包内容进行保存，便于用户查看得知数据包内具体信息而无需加载进程序。

嗅探器同时允许加载之前保存的pcap格式的文件，将文件中的数据包加载进程序中，同时显示相关信息在主窗口里面，就像刚刚抓取的数据包一样。该功能通过rdpcap()函数进行实现。

下面通过一个具体例子来进行测试。首先开启嗅探器，并选中几个数据包保存成pcap格式”hehe.pcap”文件，如图26所示，同时再保存成txt格式，如图27所示。打开保存的txt文件，如图28所示，可以看到，选中的数据包信息已经保存在txt文件里面。之后打开前面保存的“hehe.pcap”文件，如图29与30所示，之前保存的数据包被重新加载。更具体的例子可以查看文件夹中的”open-example.gif”，“savepcap-example.gif”，”savetxt-example.gif”。

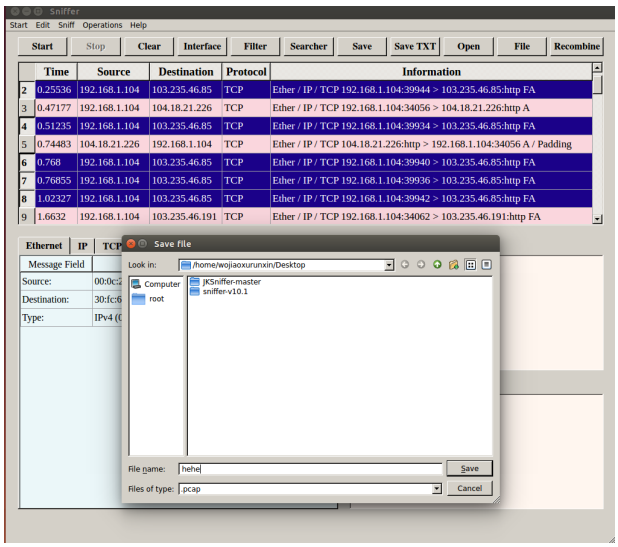


图 26: 保存为pcap格式文件

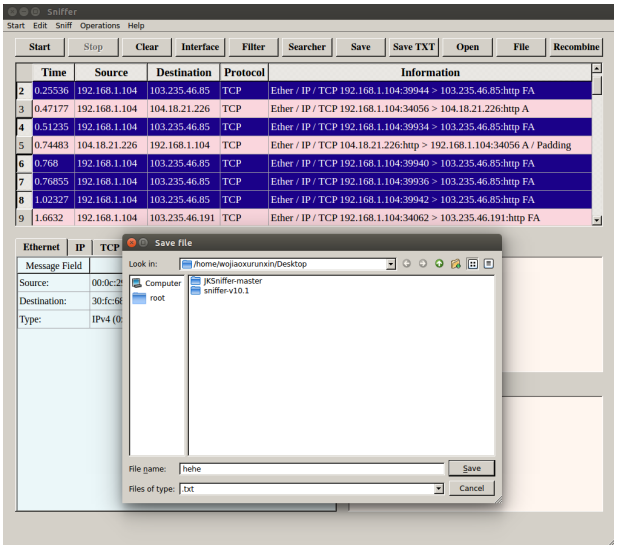


图 27: 保存为txt格式文件

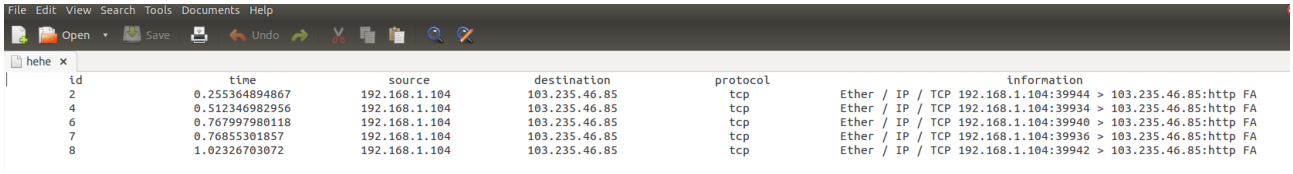


图 28: txt文件内容

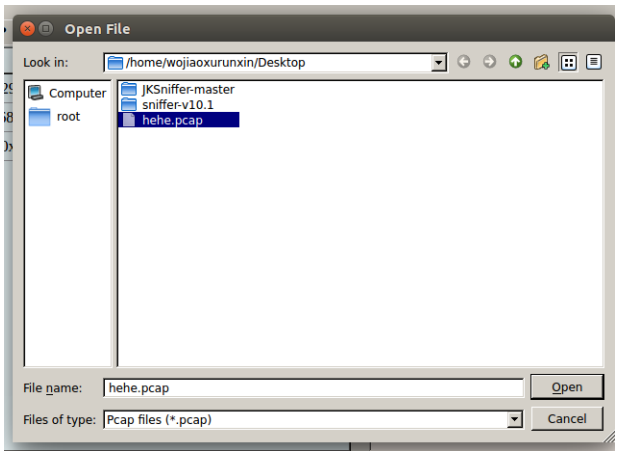


图 29: 打开之前保存的pcap文件

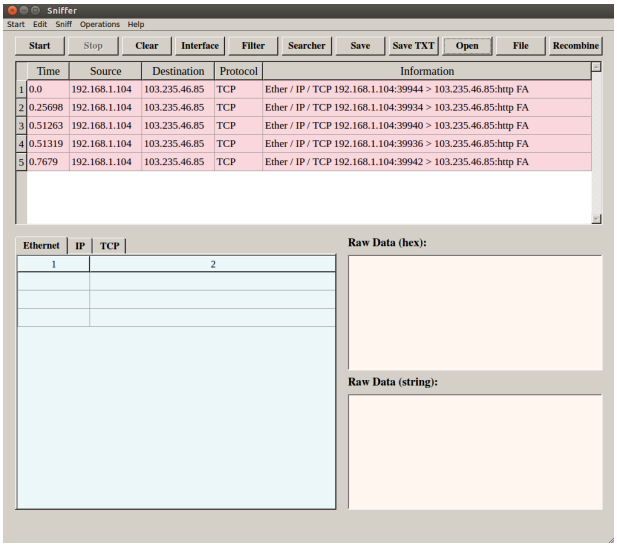


图 30: 之前保存的pcap文件内容被加载

2.7 IP报文重组

如果IP报文过长有时候会进行分段，而我们的嗅探器应用可以抓取到通过网卡的被分段的IP报文并组装成原始的完整的报文。识别是否一个来自相同原始IP报文的标志可以用一个三元组来实现，即（源IP地址，目的IP地址，IP报文标志id），相同原始IP报文分段之后，三元组中的元素都不会改变，且依据它们的偏移量字段可以按顺序将分段报文合并起来。

以下通过一个具体例子来进行测试。我们使用命令“ping -c 1 -s 4000 192.168.1.1” 来向192.168.1.1发送一个4000字节的ICMP数据包，从而达到分段的目的。通过抓包可以看到，前三个ICMP数据包分别如图31，32和33所示，可以看到它们的IP报文标志id都为9797，而偏移量分别是0，1480，2960Bytes，且源IP与目的IP地址都一样，说明三个数据包来自同一个原始IP报文。之后点击Recombine实现重组功能，结果如图34所示，可以看到第一项即为刚才的三个数据包重组之后的结果，且其IP报文标识id确实为9797，与前面相符。更具体的例子可以查看文件夹中的”recombination.gif”。

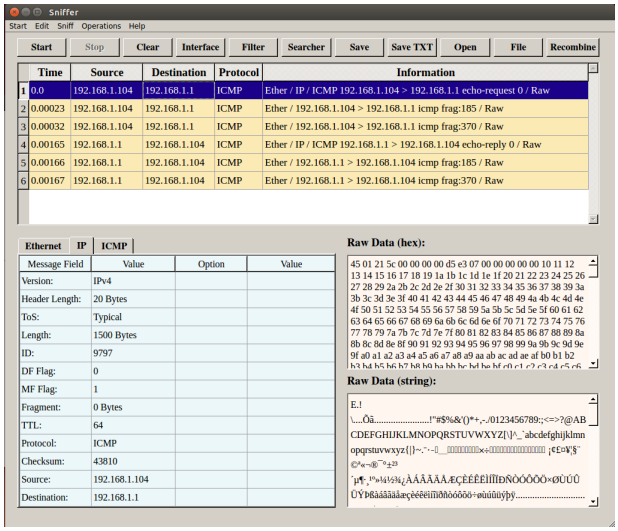


图 31: 第一个ICMP数据包信息

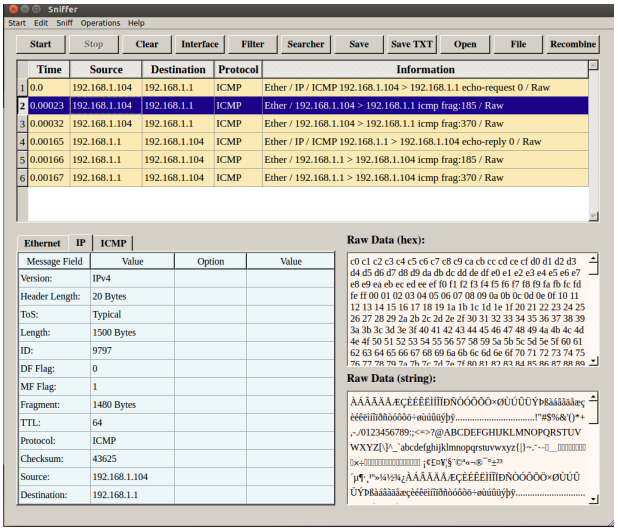


图 32: 第二个ICMP数据包信息

2.8 文件重组

在我们的网络嗅探器NetScope中，实现了对FTP被动模式下协议文件传输的重组功能。在FTP的文件传输过程中，一些过大的文件会被分为若干个TCP数据包进行传送，我们的嗅探器可以实现从已抓取的数据包中自动检测其中含有的所有文件，并将每个文件按顺序进行重组成原始文件后，进行保存。在抓取一定量数据包后，点击File按钮，即可打开所有嗅探到的文件列表（如图35），并选择其中的文件保存到自己想要的路径（如图36所示）。文件重组功能实现的大致思路如下:

(i) 首先从所有嗅探到的TCP数据包中，找到目的端口为21的数据包，这些数据包与FTP文件传输有

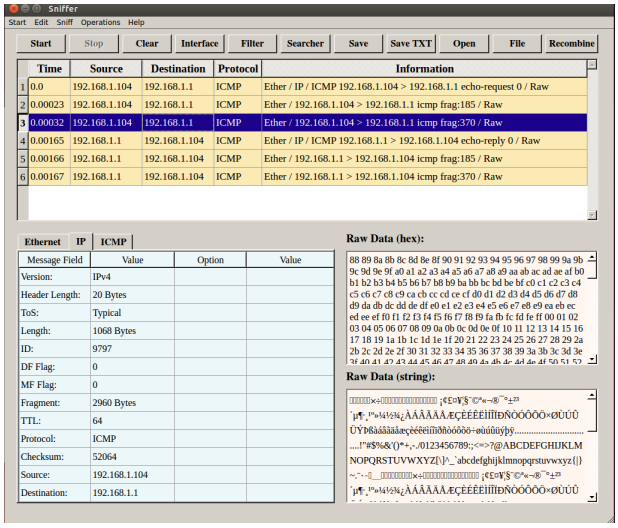


图 33: 第三个ICMP数据包信息

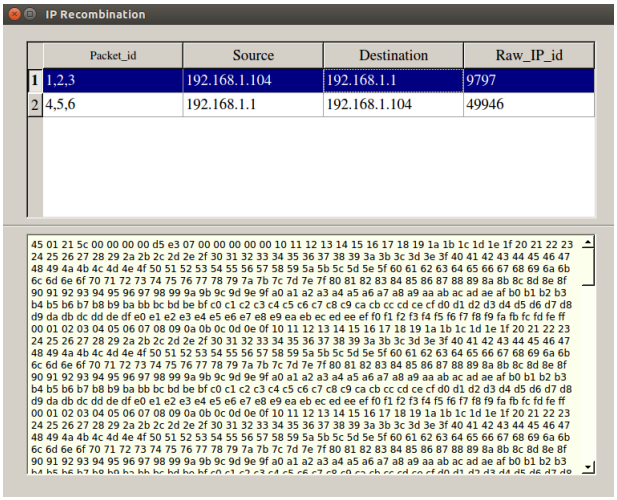


图 34: 合并ICMP数据包信息

关。

(ii) 从这些数据包的数据报文中搜索字符串'RETR'(FTP的文件传输命令)，并在RETR命令后面找到对应的请求传输的文件名filename存入文件名列表filelist。

(iii) 对filelist中的每一个filename，在其'RETR'数据包对应的应答数据包的数据报文中，搜索到被动模式对应的高位端口号。

(iv) 在所有的TCP数据包中，以这个高位端口号再次进行搜索，搜索到的数据包即包含了该文件的所有数据内容。

(v) 对搜索到的数据包，按TCP报头的seq_number进行排序，将排序好的数据包序列进行内容重组，即得到完整的重构文件。

下面我们对文件重组功能进行实验验证。首先我们在虚拟机FTP Server中配置vsftpd软件，并通过该软件对外开放FTP服务，采用被动模式登录。在这之后，我们从另外一台虚拟机FTP Client上访问FTP Server的FTP服务，并发起FTP文件传输请求，我们在虚拟机FTP Client上开启NetScope进行抓包，并将抓到的FTP数据包重组为文件，并检验得到的文件是否和源文件相同。

本实验中，我们传输的文件有两个：png格式的图片文件test_img.png和文本文件test_txt。首先，我们在FTP Client的NetScope上开启数据包抓取，之后通过FileZilla软件登录FTP Server进行文件传输，文件传输结束后停止抓包。在NetScope抓到的数据包中，点击“File”按钮进行传输文件嗅探，嗅探得到的文件列表如图35所示。我们选取图片文件“test_img.png”保存到FTP Client虚拟机的根目录下，保存名称设置为“img_recombine”，如图36所示。最后我们对比我们重组的png文件和通过FTP传输得到的文件，发现他们完全相同，如图37所示。文件重组功能成功实现。

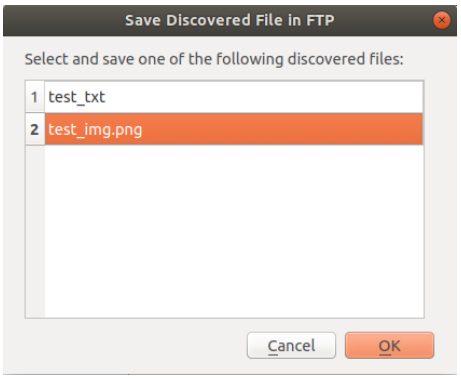


图 35: 嗅探得到的文件列表

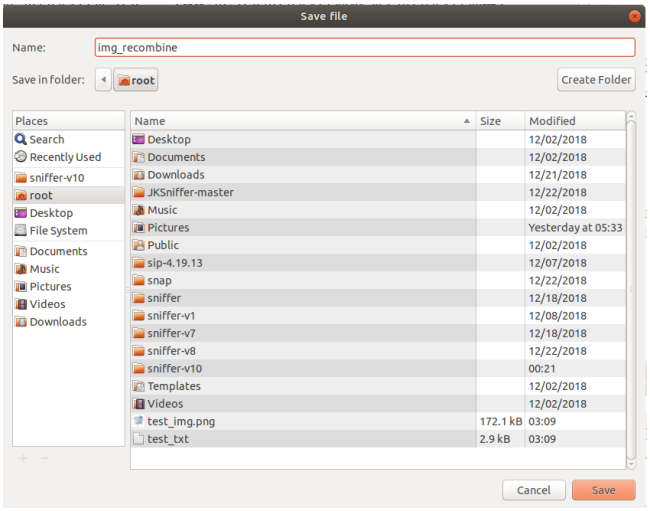


图 36: 文件以img_recombine命名保存到root

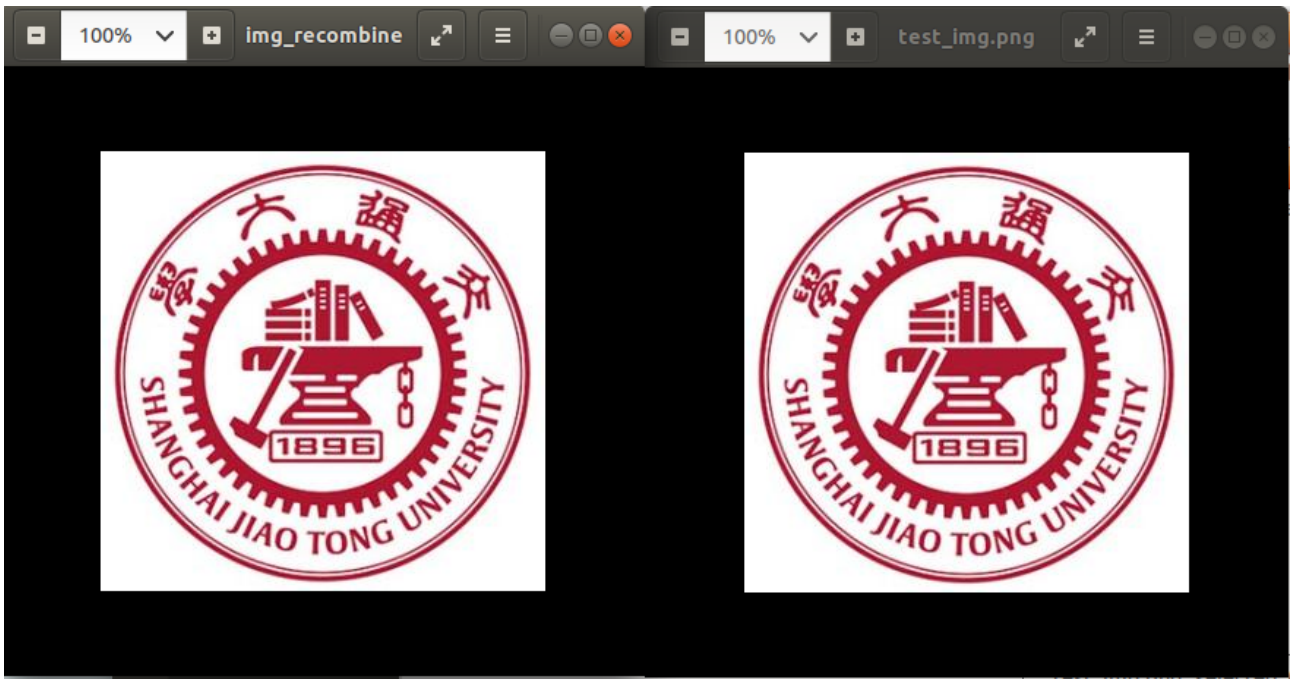


图 37: 文件重组结果：重组成功

更详细具体的实验过程请见程序运行demo：FileRecombine-example.gif，该图像详细演示了完整的实验过程。

3 遇到的问题及解决办法

3.1 数据包抓取的GUI多线程问题

由于在PyQt4中，GUI窗口在开始运行后，会等待用户进行操作，并根据用户操作对应的信号执行相应的函数。因为python2.7的解释器是单线程的，这样就涉及到一个线程冲突问题，也就是当我们点击Start按钮开始抓包时，python解释器的线程会跳到Start按钮对应的函数start_sniffing，由于该函数是一个持续的过程，所以在抓包的过程中我们不能干任何事，甚至不能够停止抓包，这显然是不符合设计要求的。

我们的解决方法是利用了sniff函数中的参数stop_filter，它可以作为停止sniff行为的一个过滤参数，当满足传给该参数的参数条件时，sniff函数就会停止执行。因此，我们利用了threading库中的事件类，即定义了e=threading.Event()，并在sniff函数中的stop_filter参数中用lambda表达式来检测时间e是否被设置，即sniff(iface = self.interface_name, prn = self.sniffing_callback, filter = self.filter_rule, stop_filter = lambda p: self.e.is_set())，如果事件e没有被设置，则会持续嗅探；如果事件e被设置了，那么在下一次抓包过程中就会检测到，并退出sniff线程。所以当开始进行嗅探时，我们将事件e清除，即self.e.clear()；当用户点击停止，即调用了self.stop_sniffing(self)函数时，我们在函数内部将事件e设置，即self.e.set()。通过以上技巧，我们实现了根据用户需求执行、停止sniff线程的功能。

3.2 scapy库未提供IGMP协议的解析

scapy库未提供IGMP协议的解析，我们只好自己通过IGMP数据包的上层（IP层）的数据进行解析。经过查阅资料，我们了解到了每一个字段的长度、作用、可取值及其对应的含义。由于scapy库给出的Raw Data是字符串格式，将每8个比特视为一个ASCII字符。因此，我们通过提取字符串对应位数的取值，并进行分析和翻译，实现了IGMP数据报的解析。

3.3 string格式的Raw Data无法显示的问题

在我们进行报的内容显示时，一些字节对应的ASCII码不是可打印字符，会导致报文内容整体无法显示。我们的解决方法是，将ASCII码的不可打印字符(0-31,127)替换为句号‘.’，这样就可以成功的显示string格式的Raw Data。

3.4 FTP文件存在加密

经过试验，我们发现学校的FTP：public.sjtu.edu.cn采用了加密传输，无法进行实验，甚至无法实现FTP的文件重组，因为加密的信息无法判断一个文件的开始和结束，也无法判断被动FTP下的服务器开放的高位端口。所以，我们只好自行搭建实验环境进行实验。我们通过在虚拟机上安装vsftpd软件开放FTP服务，并在另一台虚拟机上登录传输文件，来验证文件重组功能的正确性。这也造成了我们无法对加密传输的FTP进行监听并重组文件。除此之外，由于现在的Web服务器均使用了https加密，造成我们也无法通过sniffer来监听http的文件传输。

3.5 IP报文重组功能测试的问题

在一开始进行测试时，点击重组按钮之后，并没有出现重组之后的报文。一开始怀疑是自己的程序出现了问题，后来经过仔细排查后，发现是嗅探到的数据包中没有是分片的。这也给测试带来了不少麻烦。后来经过查找资料之后，发现可以通过ping命令的参数来控制ICMP报文属性实现分片，所以我们通过了命令”ping -c 1 -s 4000 192.168.1.1” 来发ICMP报文并进行嗅探，从而对IP报文重组功能顺利进行了测试，验证了程序的正确性。

3.6 PyQt4库的使用问题

我们在程序开发的初期，对PyQt4的类继承关系并不熟练，网络上的资料又不是很可靠，官方的英文文档也写得很简略，导致我们走了很多弯路，浪费了大量的时间去调试各种类的继承关系。到程序开发的后期，对各种GUI的子类关系熟练后，效率就提高了很多。

4 总结与体会

在这次完成大作业的过程中，我们首先针对大作业的相关要求做了分析，将项目的完成基本上划分为三部分：a）各个功能逻辑代码实现；b）界面的设计与实现；c）界面与功能的衔接与结合。在此基础上，我们通过查找资料，确定采用Scapy库来完成数据包的嗅探等逻辑功能，而通过PyQt4库来进行界面的设计与实现，基于Ubuntu14.04来进行整个项目的开发设计。

在功能实现过程中，我们采用了先易后难的做法，先实现一些简单易实现的功能，之后再不断地完善比较复杂的功能。在实现选择侦听网卡的过程中，我们学习了Linux系统的一些基本命令与其文件系统的主要结构，懂得了哪些系统文件存储哪些系统使用的参数；在嗅探并解析数据包的功能实现中，通过对不同报文的字段解析，将老师在课堂上讲过的知识融会贯通，同时也更加细致深刻地了解到了网络通信协议的相关知识；在实现数据包过滤和查询功能的过程中，我们学习到了可以通过哪些字段来唯一确定一个数据包，各个字段对于网络通信中唯一标识一个数据包的作用和重要性；在数据包的保存与加载功能

实现中，了解到了Scapy库使用的pcap格式文件的基本知识；在实现IP报文重组和文件重组功能中，又深刻认识到了标识、分段标志、偏移量等等字段的重要性，在网络通信中就是根据这些字段来进行报文与文件的分段与重组，从而使信息流能够适应不同带宽的信道。

在界面的设计和实现过程中，由于我们对PyQt4不太熟悉，所以查阅了大量的资料进行学习，了解到它不仅可以通过代码命令来实现，还有更加友好的工具qt-designer，可以让我们以GUI的形式来设计我们所需要的界面，并自动生成相关的代码，这极大的节省了我们的时间，提高了开发的效率。对于PyQt4中的各个控件，我们也相当好奇，对不同的控件和相关的方法都进行了尝试，最终找到了最适合项目的控件，在这个过程中对于界面设计也有了更深刻的了解。而在设计过程中，我们也了解到不仅仅要从程序员的角度思考问题，还需要从用户的角度来思考，从而让界面呈现出让用户更为喜欢以及方便使用的效果。

关于本作品的评价和未来的展望，我们认为我们的优点在于相比于市面上许多其他的sniffer软件，我们的作品可以对数据包内的各种字段进行更加详细和易懂的解析。因为在数据报文的内部，数据包的性质和各个字段的含义是隐含在比特串中的，我们尽可能地将这些含义进行更加详细的解析，包括各个字段的值具体对应什么内容，而不是像很多网络上开源的sniffer那样，将这些字段的值仅仅用一个类似show()函数打印屏幕上，在进行深度解析的过程中也极大增加了作品的代码量，但也极大增加了数据包字段的可读性。

当然，我们的作品也有很多不足之处。例如，我们没有实现在HTTP协议下的文件传输嗅探与重组。现在由于HTTPS加密协议的应用比较广泛，我们无法找到合适的实验平台进行功能的实现和代码的调试，再加上临近期末时间紧迫，导致我们没有对HTTP协议进行解析。但我们实现了对加密应用的相对不是特别广泛的FTP传输协议进行了文件重组功能的实现，由于FTP和HTTP协议均为基于字节流进行传输，所以两者的设计思想应该可以相互借鉴。在今后软件的开发完善中，对HTTP协议的解析将会成为我们努力的方向。

总而言之，这次大作业让我们收获很多，将近1个月的开发的过程让我们学到了很多东西，比如如何找到并快速学习应用一些库，GUI开发的一些基本思想，还有一些有关linux操作系统内核原理等等。作为计算机相关专业的学生，这些将来都是我们的必备技能。同时我们也非常感谢老师和助教的热心帮助，在我们遇到问题时，老师很耐心的帮我们进行了解答，极大的提高了我们的工作效率。在本学期结束后，我们也将会保持对本项目的持续开发和维护，请大家期待。