# 嗅探器设计与实现

Github地址：[Zhousongjie/sniffer (github.com)](https://github.com/Zhousongjie/sniffer)

## 设计内容

网络嗅探器，Network Packet Sniffer（NPS），是一种专门用来进行网络流量监听的工具。数据包嗅探器是研究网络行为学的基础工具。

软件包含逻辑推导组件。基本的协议过滤需要筛选 HTTP、TCP/UDP、IPv4/v6、ICMP等不同类型、层次的数据包。

使用C++ QT实现图形化界面。

## 设计步骤

### 2.1原理分析

原始套接字是一种不同于 SOCK\_STREAM 和 SOCK\_DGRAM 的套接字，它实现于系统核心。创建方式与TCP或UDP差不多，但是功能与 TCP 或者 UDP 类型套接字的功能有很大的不同：TCP/UDP 类型的套接字只能够访问传输层以及传输层以上的数据，因为当 IP 层把数据传递给传输层时，下层的数据包头已经被丢掉了。而原始套接字却可以访问传输层以下的数据，所以使用 raw 套接字你可以实现上至应用层的数据操作，也可以实现下至链路层的数据操作。

(1)：对于 UDP/TCP 产生的 IP 数据包，内核不将它传递给任何原始套接字，而只是将这些数据交给对应的 UDP/TCP 数据处理句柄(所以，如果你想要通过原始套接字来访问 TCP/UDP 或者其它类型的数据，调用 socket 函数创建原始套接字第三个参数应该指定为 htons(ETH\_P\_IP)，也就是通过直接访问数据链路层来实现。

(2)：对于 ICMP 和 EGP 等使用 IP 数据包承载数据但又在传输层之下的协议类型的 IP 数据包，内核不管是否已经有注册了的句柄来处理这些数据，都会将这些 IP 数据包复制一份传递给协议类型匹配的原始套接字。

(3)：对于不能识别协议类型的数据包，内核进行必要的校验，然后会查看是否有类型匹配的原始套接字负责处理这些数据，如果有的话，就会将这些 IP 数据包复制一份传递给匹配的原始套接字，否则，内核将会丢弃这个 IP 数据包，并返回一个 ICMP 主机不可达的消息给源主机。

(4)：如果原始套接字 bind 绑定了一个地址，核心只将目的地址为本机 IP 地址的数据包传递给原始套接字，如果某个原始套接字没有 bind 地址，核心就会把收到的所有 IP 数据包发给这个原始套接字。

(5)：如果原始套接字调用了 connect 函数，则核心只将源地址为 connect 连接的 IP 地址的 IP 数据包传递给这个原始套接字。

(6)：如果原始套接字没有调用 bind 和 connect 函数，则核心会将所有协议匹配的 IP 数据包传递给这个原始套接字。

### 2.2数据结构

/\*以太网帧头格式结构体 14个字节\*/

typedef struct ether\_header

{

unsigned char ether\_dhost[6];// 目的MAC地址

unsigned char ether\_shost[6];// 源MAC地址

unsigned short ether\_type;// eh\_type的值需要考察上一层的协议，如果为ip则为0×0800

}ETHERHEADER, \*PETHERHEADER;

/\*以ARP字段结构体 28个字节\*/

typedef struct arp\_header

{

unsigned short arp\_hrd;

unsigned short arp\_pro;

unsigned char arp\_hln;

unsigned char arp\_pln;

unsigned short arp\_op;

unsigned char arp\_sourha[6];

unsigned long arp\_sourpa;

unsigned char arp\_destha[6];

unsigned long arp\_destpa;

}ARPHEADER, \*PARPHEADER;

/\*ARP报文结构体 42个字节\*/

typedef struct arp\_packet

{

ETHERHEADER etherHeader;

ARPHEADER arpHeader;

}ARPPACKET, \*PARPPACKET;

// ipv4\_pro字段：

#define PROTOCOL\_ICMP 0x01

#define PROTOCOL\_IGMP 0x02

#define PROTOCOL\_TCP 0x06

#define PROTOCOL\_UDP 0x11

/\*IPv4报头结构体 20个字节\*/

typedef struct ipv4\_header

{

unsigned char ipv4\_ver\_hl;// Version(4 bits) + Internet Header Length(4 bits)长度按4字节对齐

unsigned char ipv4\_stype; // 服务类型

unsigned short ipv4\_plen;// 总长度（包含IP数据头，TCP数据头以及数据）

unsigned short ipv4\_pidentify;// ID定义单独IP

unsigned short ipv4\_flag\_offset;// 标志位偏移量

unsigned char ipv4\_ttl; // 生存时间

unsigned char ipv4\_pro;// 协议类型

unsigned short ipv4\_crc;// 校验和

unsigned long ipv4\_sourpa;// 源IP地址

unsigned long ipv4\_destpa;// 目的IP地址

}IPV4HEADER, \*PIPV4HEADER;

/\*IPv6报头结构体 40个字节\*/

typedef struct ipv6\_header

{

unsigned char ipv6\_ver\_hl;

unsigned char ipv6\_priority;

unsigned short ipv6\_lable;

unsigned short ipv6\_plen;

unsigned char ipv6\_nextheader;

unsigned char ipv6\_limits;

unsigned char ipv6\_sourpa[16];

unsigned char ipv6\_destpa[16];

}IPV6HEADER, \*PIPV6HEADER;

/\*TCP报头结构体 20个字节\*/

typedef struct tcp\_header

{

unsigned short tcp\_sourport;//源端口

unsigned short tcp\_destport;//目的端口

unsigned long tcp\_seqnu;//序列号

unsigned long tcp\_acknu;//确认号

unsigned char tcp\_hlen; //4位首部长度 还有4位保留

unsigned char tcp\_reserved;//前2位保留 后6位标志位

unsigned short tcp\_window;//窗口大小

unsigned short tcp\_chksum;//检验和

unsigned short tcp\_urgpoint;//紧急指针

}TCPHEADER, \*PTCPHEADER;

/\*UDP报头结构体 8个字节\*/

typedef struct udp\_header

{

unsigned short udp\_sourport;// 源端口

unsigned short udp\_destport;// 目的端口

unsigned short udp\_hlen;// 长度

unsigned short udp\_crc;// 校验和

}UDPHEADER, \*PUDPHEADER;

### 2.3数据包解析

/\*

1.分析IP包的协议类型

2.根据协议类型分析首部和数据部分

\*/

void MySniffer::**analyseRecvPacket**(BYTE \*lpBuf)

{

//分析IP协议

//分析IP包的协议类型

struct sockaddr\_in saddr, daddr;

PIPV4HEADER t\_ip = (PIPV4HEADER)lpBuf;

if(t\_ip->ipv4\_pro == IPPROTO\_ICMP)

{

saddr.sin\_addr.s\_addr = t\_ip->ipv4\_sourpa;

daddr.sin\_addr.s\_addr = t\_ip->ipv4\_destpa;

emit this->sendData(QStringList()<<"ICMP"<<inet\_ntoa(saddr.sin\_addr)<<inet\_ntoa(daddr.sin\_addr)<<""<<"");

}

else if(t\_ip->ipv4\_pro == IPPROTO\_IGMP)

{

saddr.sin\_addr.s\_addr = t\_ip->ipv4\_sourpa;

daddr.sin\_addr.s\_addr = t\_ip->ipv4\_destpa;

emit this->sendData(QStringList()<<"IGMP"<<inet\_ntoa(saddr.sin\_addr)<<inet\_ntoa(daddr.sin\_addr)<<""<<"");

}

else if(t\_ip->ipv4\_pro == IPPROTO\_TCP)

{

PTCPHEADER tcp = (PTCPHEADER)(lpBuf + (t\_ip->ipv4\_ver\_hl & 0x0F)\*4);

int hlen = ((t\_ip->ipv4\_ver\_hl & 0x0F) \* 4) + ((((tcp->tcp\_hlen)>>4 & 0x0F) \* 4)& 0x0F);

int dlen = ((quint16)ntohs(t\_ip->ipv4\_plen )) - hlen; //这里要将网络字节序转换为本地字节序

saddr.sin\_addr.s\_addr = t\_ip->ipv4\_sourpa;

daddr.sin\_addr.s\_addr = t\_ip->ipv4\_destpa;

QStringList string\_tcp;

string\_tcp<<"TCP";

string\_tcp <<QString(inet\_ntoa(saddr.sin\_addr))+":"+QString::number(ntohs(tcp->tcp\_sourport));

string\_tcp <<QString(inet\_ntoa(daddr.sin\_addr))+":"+QString::number(ntohs(tcp->tcp\_destport));

string\_tcp<<getDataFromPacket((lpBuf + hlen), dlen, 0);

//MyPrintf("ack:%u syn:%u length=%d\n", tcp->tcp\_acknu, tcp->tcp\_seqnu, dlen);

string\_tcp <<QString("ack:%1 syn:%2 length=%3").arg(tcp->tcp\_acknu).arg(tcp->tcp\_seqnu).arg(dlen);

emit this->sendData(string\_tcp);

}

else if(t\_ip->ipv4\_pro == IPPROTO\_UDP)

{

PUDPHEADER udp = (PUDPHEADER)(lpBuf + (t\_ip->ipv4\_ver\_hl & 0x0F) \* 4);

int hlen = (int)((t\_ip->ipv4\_ver\_hl & 0x0F) \* 4 + sizeof(UDPHEADER));

int dlen = (int)(ntohs(udp->udp\_hlen) - 8);

// int dlen = (int)(udp->udp\_hlen - 8);

saddr.sin\_addr.s\_addr = t\_ip->ipv4\_sourpa;

daddr.sin\_addr.s\_addr = t\_ip->ipv4\_destpa;

QStringList string\_udp;

string\_udp<<"UDP";

string\_udp <<QString(inet\_ntoa(saddr.sin\_addr))+":"+QString::number(ntohs(udp->udp\_sourport));

string\_udp <<QString(inet\_ntoa(daddr.sin\_addr))+":"+QString::number(ntohs(udp->udp\_destport));

string\_udp<<getDataFromPacket((lpBuf + hlen), dlen, 0);

string\_udp <<"";

emit this->sendData(string\_udp);

}

else

{

saddr.sin\_addr.s\_addr = t\_ip->ipv4\_sourpa;

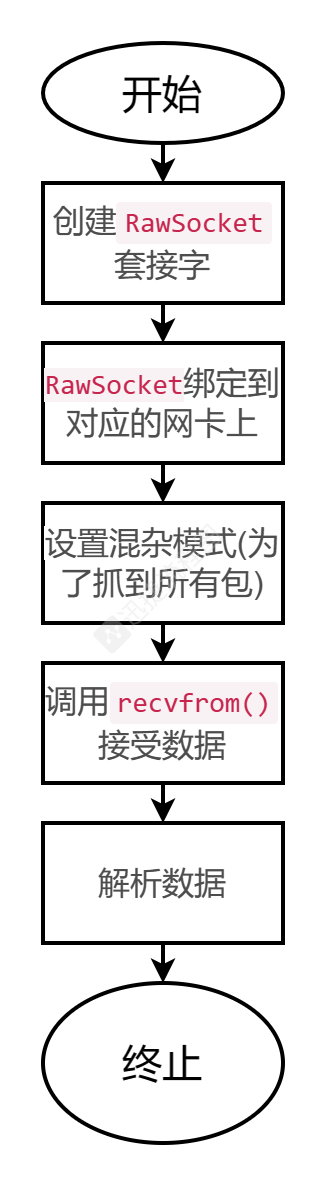
daddr.sin\_addr.s\_addr = t\_ip->ipv4\_destpa;

emit this->sendData(QStringList()<<"OTHER"<<inet\_ntoa(saddr.sin\_addr)<<inet\_ntoa(daddr.sin\_addr)<<""<<"");

}

}

### 2.4流程图

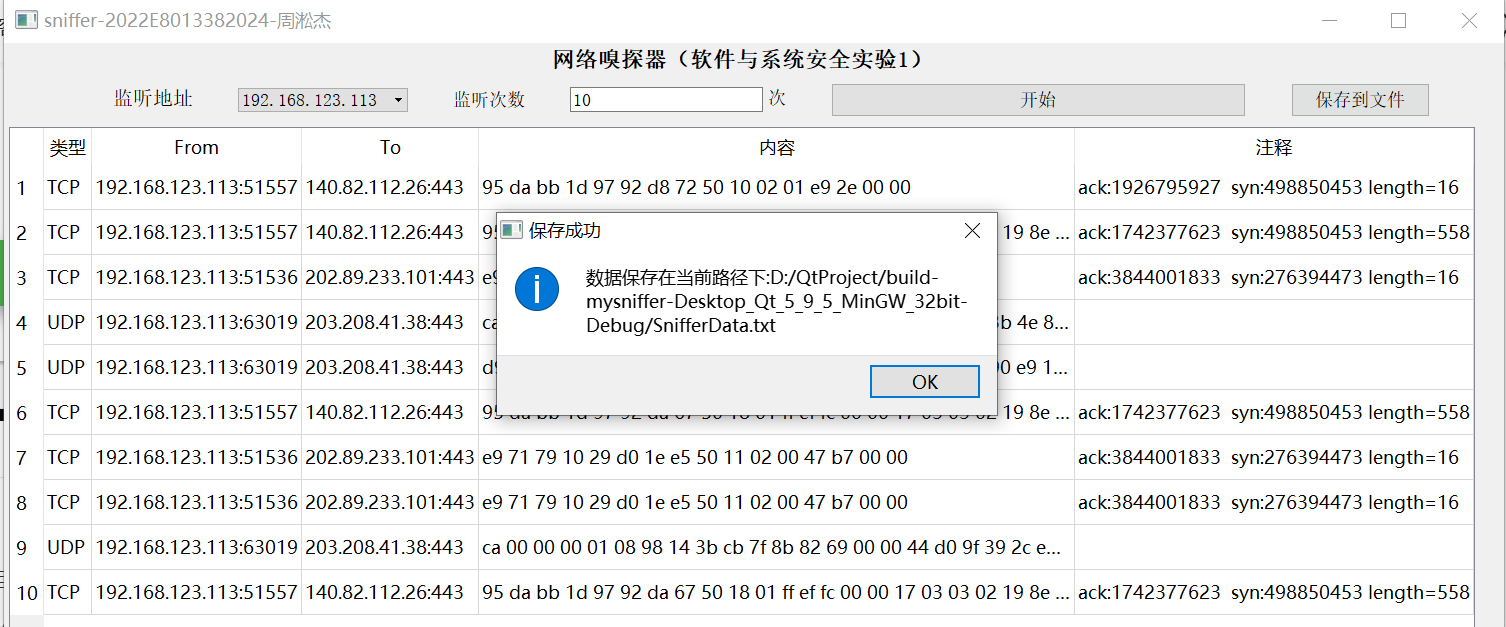


## 实验结果与分析

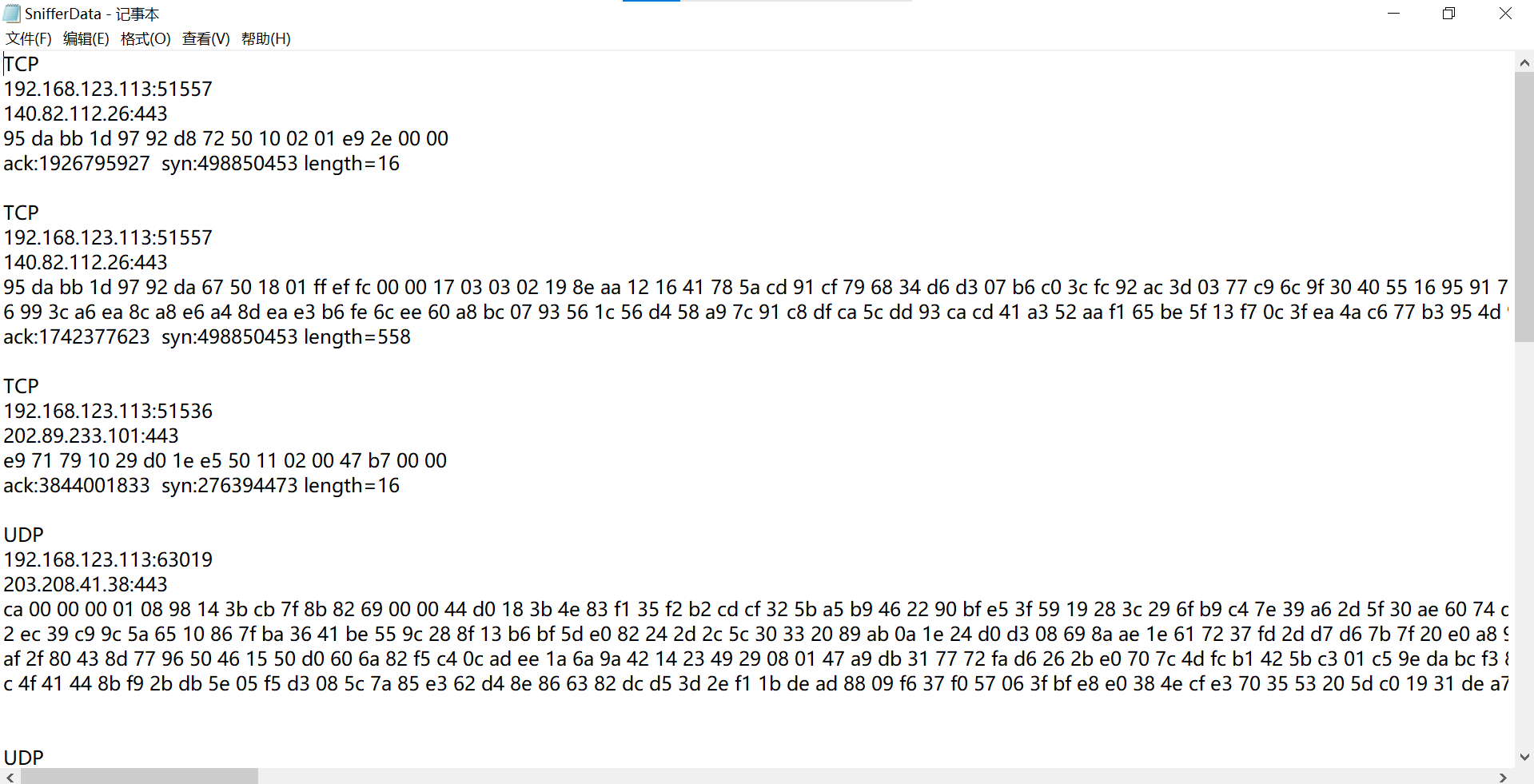
选择监听地址，监听次数，开始监听。



点击保存到文件，将监听数据保存



打开对应路径下的SnifferData.txt文件，可查看对于数据



## 实验总结

本实验使用原始套接字RawSocket实现数据包的嗅探。开启管理员权限后，RawSocket可以抓到IP层及以上的数据包，抓不到MAC层的数据包。

通过本次实验，我对于TCP/UDP的数据格式有了很深的理解，学习到了如何解析数据包数据，同时学习了C++图形化界面QT，从中获益匪浅。