**局域网截包程序设计**

作者：吴玉波-berton

2015 年 12 月 23 日

1. **设计目的**

目前的局域网基本上都采用以广播为技术基础的以太网，任何两个节点之间的通信数据包，不仅为这两个节点的网卡所接收，也同时为处在同一以太网上的任何一个节点的网卡所截取。因此，黑客只要接入以太网上的任一节点进行侦听，就可以捕获发生在这个以太网上的所有数据包，对其进行解包分析，从而窃取关键信息，这就是以太网所固有的安全隐患。网上主要的免费黑客攻击工具如SATAN、ISS、NETCAT等均将以太网侦听作为基本的手段。

本实验实现一个局域网截包程序，目的在于更好地理解网络的工作机制（包括封包机制、协议分析等），该实验是对《计算机网络》课程一个有益的补充。

1. **设计内容**

在一个局域网环境中，用C 语言实现下面的基本功能：

1）截包的方法：

包括RAW 模式SOCKET

sock = socket(PF\_PACKET, SOCK\_RAW, htons(ETH\_P\_ALL));

2）截获以下包的类型并分析：

以太网帧格式（MAC）、IP包、ICMP包、TCP报文段、UDP报文等相关字段进行描述。

3）界面设计

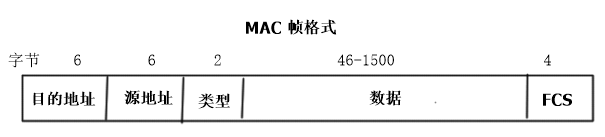
用QT 设计界面，具备交互和显示功能，其交互包括开始、停止、过滤等，显示则为各类型包的分析。

1. **平台与语言**

**平**台： LINUX 语言： C++/C 界面设计： qt

1. **设计方法、步骤及结构测试**
2. **结构体的构造：**

①、 MAC 帧头结构



struct MacHeader

{

unsigned char dest\_adr[6]; //目的地址

unsigned char source\_adr[6]; //源地址

unsigned short type; //类型

};

②、IP数据包头结构



struct IpHeader

{

unsigned int header\_len:4; //首部长度

unsigned int versoin:4; //版本号

unsigned char service; //区分服务

unsigned short tatol\_len; //总长度

unsigned short ident; //标识

unsigned short flag\_frag; //标志与片偏移

unsigned char ttl; //生存时间

unsigned char protocol; //协议

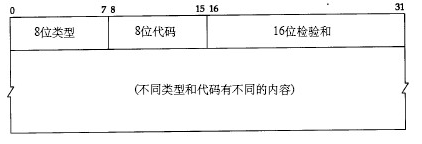
unsigned short check\_sum; //检验和

unsigned char source\_ip[4]; //源地址

unsigned char dest\_ip[4]; //目的地址

};

③、ICMP数据头结构



struct IcmpHeader

{

unsigned char type; //类型

unsigned char code; //代码

unsigned short check\_sum; //检验和

unsigned short id; //标识符

unsigned short seq; //序列号

};

④、UDP数据包头结构



struct UdpHeader

{

unsigned short source\_port; //源端口

unsigned short dest\_port; //目的端口

unsigned short len; //udp总长度

unsigned short check\_sum; //检验和

};

⑤、TCP数据包头结构



struct TcpHeader

{

unsigned short source\_port; //源端口

unsigned short dest\_port; //目的端口

unsigned int send\_num;

unsigned int recv\_num;

unsigned char reserved1:4;

unsigned char offset:4;

unsigned char flag:6;

unsigned char reserved2:2;

unsigned short window;

unsigned short check\_sum;

unsigned short urg\_pointer;

};

**2、设计流程**



监控函数：n = recvfrom(sock,data\_li[i],2048,0,NULL,NULL);

解析数据包：

p = msniff->data\_li[i];

showMac((struct MacHeader \*) p);

struct IpHeader \*ipheader = (struct IpHeader \*)( p + 14);

showIP(ipheader);

switch (ipheader->protocol) {

case ICMP:

showIcmp((struct IcmpHeader \*)(p+14+ipheader->header\_len\*4));

break;

case TCP:

showTcp((struct TcpHeader \*)(p+14+ipheader->header\_len\*4));

break;

case UDP:

showUdp((struct UdpHeader \*)(p+14+ipheader->header\_len\*4));

break;

default:

break;

}

**3、类的设计**

程序有两个类构成，一个是界面类（Qt的MainWindow），一个是截包类（继承Qt的抽象线程类QThread）。

1. 界面类的设计如下：

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

//构造函数

explicit MainWindow(QWidget \*parent = 0);

//析构函数

~MainWindow();

//显示MAC首部格式

void showMac(struct MacHeader \*mheader);

//显示IP首部格式

void showIP(struct IpHeader \*ipheader);

//显示ICMP首部格式

void showIcmp(IcmpHeader \*icmpheader);

//显示TCP首部格式

void showTcp(struct TcpHeader \*tcpheader);

//显示UDP首部格式

void showUdp(struct UdpHeader \*udpheader);

//设置过滤器

int check\_filter(QString qf);

private slots:

//开始按钮

void on\_pushButton\_start\_clicked();

//停止按钮

void on\_pushButton\_stop\_clicked();

//选择窗口列表的选项（截到的数据包）

void on\_listWidget\_list\_doubleClicked(const QModelIndex &index);

private:

//界面设计

Ui::MainWindow \*ui;

//截包线程

Sniff \*msniff;

};

1. 截包类的设计如下：

class Sniff : public QThread

{

Q\_OBJECT

public:

//构造函数

explicit Sniff(QObject \*parent = 0);

//设置显示控件

void setW(QListWidget \*\_list\_lw, QLabel \*\_load\_lb);

//线程运行

void run();

//开始截包

void startsniff(int \_filter);

//停止截包

void stop();

//截包缓存区

char data\_li[MAXDATAGRAM][2048];

//设置过滤

void setFilter(int \_filter);

//获取协议名

QString getProtocol(int protocol);

private:

//原始套接字

int sock;

struct ifreq ethreq;

int n;

struct MacHeader \*mheader;

struct IpHeader \*ipheader;

//截包状态

int state;

//过虑器

int filter;

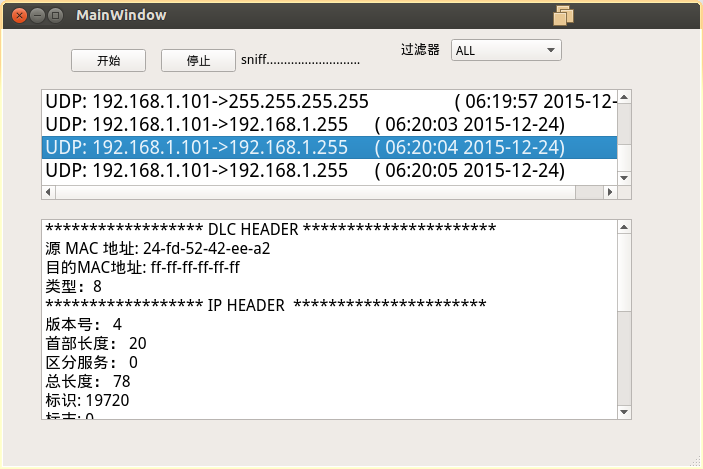
//显示控件

QListWidget \*list\_lw;

QLabel \*load\_lb;

};

**4、界面设计**



开始按键则开始截包、停止为停止截包

Sniff...标签提示正在截包

过滤器，可以过滤ICMP、TCP、UDP的包的截取

窗口列表1是截到的全部包的基本信息

窗口列表2是查看某包的详细信息

A、开始按钮：

void MainWindow::on\_pushButton\_start\_clicked()

{

int filter = ALL;

//设置过滤器

filter = check\_filter(ui->comboBox\_filter->currentText());

//开始截包

msniff->startsniff(filter);

}

//允许抓取

void Sniff::startsniff(int \_filter)

{

setFilter(\_filter);

state = START;

load\_lb->setText("sniff...");

}

B、停止按钮：

void MainWindow::on\_pushButton\_stop\_clicked()

{

msniff->stop();

}

//停止抓取

void Sniff::stop()

{

state = STOP;

load\_lb->setText("stop");

}

C、提示标签：

开始：load\_lb->setText("sniff...");

停止：load\_lb->setText("stop");

D、过滤器：

//设置过滤

int MainWindow::check\_filter(QString qf)

{

if(qf == "ALL") return ALL;

if(qf == "ICMP") return ICMP;

if(qf == "TCP") return TCP;

if(qf == "UDP") return UDP;

return ALL;

}

E、窗口列表1：

//显示源ip和目的ip

path.append(QString("%1: %2.%3.%4.%5->").arg(getProtocol(ipheader->protocol))

.arg(QString::number((int)ipheader->source\_ip[0]))

.arg(QString::number((int)ipheader->source\_ip[1]))

.arg(QString::number((int)ipheader->source\_ip[2]))

.arg(QString::number((int)ipheader->source\_ip[3])));

path.append(QString("%1.%2.%3.%4").arg(QString::number((int)ipheader->dest\_ip[0]))

.arg(QString::number((int)ipheader->dest\_ip[1]))

.arg(QString::number((int)ipheader->dest\_ip[2]))

.arg(QString::number((int)ipheader->dest\_ip[3])));

//获当前时间

QDateTime current\_date\_time = QDateTime::currentDateTime();

QString current\_date = current\_date\_time.toString(" hh:mm:ss yyyy-MM-dd");

//显示数据包类型+抓取时间

path.append(QString("\t (%1)").arg(current\_date));

//显示

list\_lw->addItem(path);

F、窗口列表2：

//显示数据

void MainWindow::on\_listWidget\_list\_doubleClicked(const QModelIndex &index)

{

int i = index.row();

char \*p;

ui->listWidget\_show->clear();

p = msniff->data\_li[i];

showMac((struct MacHeader \*) p);

struct IpHeader \*ipheader = (struct IpHeader \*)( p + 14);

showIP(ipheader);

switch (ipheader->protocol) {

case ICMP:

showIcmp((struct IcmpHeader \*)(p+14+ipheader->header\_len\*4));

break;

case TCP:

showTcp((struct TcpHeader \*)(p+14+ipheader->header\_len\*4));

break;

case UDP:

showUdp((struct UdpHeader \*)(p+14+ipheader->header\_len\*4));

break;

default:

break;

}

}

**5、问题解决**

大小端问题：设计平台是小端的而网络序是大端的

由于一开始忽略了大小端问题，造成解析出来的数据只要是两个字节以上都与预期的不一样。

为了解决这个问题，并且为了更好地理解大小端，做以了以下的解决方案（另外的方法可用函数ntohl(uint32\_t netlong)和ntohs(uint16\_t netshort)进行调整）：

/\*由于接收双字节的顺序网络序的，需要调整过来\*/

ipheader->tatol\_len = (ipheader->tatol\_len>>8) + (ipheader->tatol\_len<<8);

ipheader->ident = (ipheader->ident>>8) + (ipheader->ident<<8);

ipheader->flag\_frag = (ipheader->flag\_frag>>8) + (ipheader->flag\_frag<<8);

ipheader->check\_sum = (ipheader->check\_sum>>8) + (ipheader->check\_sum<<8);

switch(ipheader->protocol)

{

case ICMP:

/\*由于接收双字节的网络序的，需要调整过来\*/

icmpheader->check\_sum = (icmpheader->check\_sum>>8) +

(icmpheader->check\_sum<<8);

icmpheader->id = (icmpheader->id>>8) + (icmpheader->id<<8);

icmpheader->seq = (icmpheader->seq>>8) + (icmpheader->seq<<8);

break;

case TCP:

/\*由于接收双字节的网络序的，需要调整过来\*/

tcpheader->source\_port = (tcpheader->source\_port>>8) +

(tcpheader->source\_port<<8);

tcpheader->dest\_port = (tcpheader->dest\_port>>8) +

(tcpheader->dest\_port<<8);

tcpheader->window = (tcpheader->window>>8) +

(tcpheader->window<<8);

tcpheader->check\_sum = (tcpheader->check\_sum>>8) +

(tcpheader->check\_sum<<8);

tcpheader->send\_num = (tcpheader->send\_num>>24) +

((tcpheader->send\_num>>8)&0x00ff00)

+ ((tcpheader->send\_num<<8)&0x00ff0000) +

(tcpheader->send\_num<<24);

tcpheader->recv\_num = (tcpheader->recv\_num>>24) +

((tcpheader->recv\_num>>8)&0x00ff00)

+ ((tcpheader->recv\_num<<8)&0x00ff0000) +

break;

case UDP:

/\*由于接收双字节的是网络序的，需要调整过来\*/

udpheader->source\_port = (udpheader->source\_port>>8) +

(udpheader->source\_port<<8);

udpheader->dest\_port = (udpheader->dest\_port>>8) +

(udpheader->dest\_port<<8);

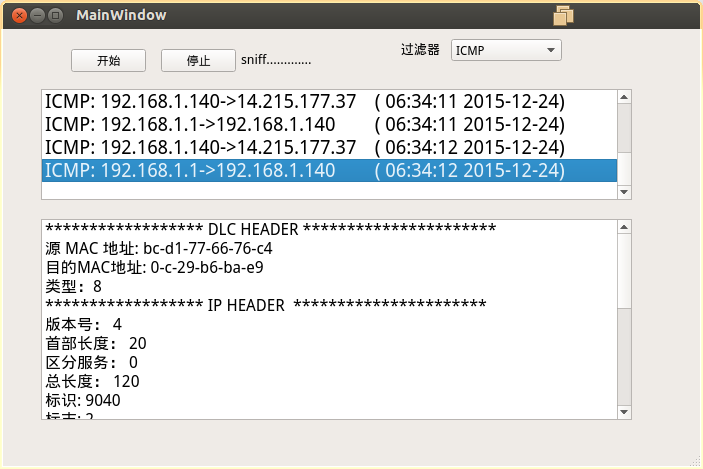
udpheader->len = (udpheader->len>>8) + (udpheader->len<<8);

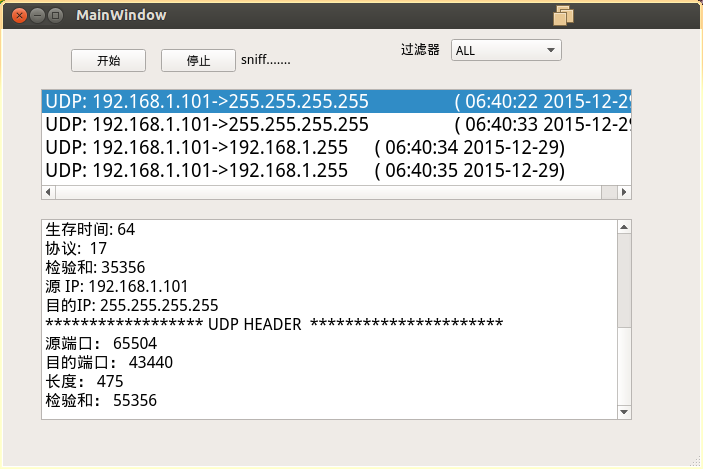
udpheader->check\_sum = (udpheader->check\_sum>>8) +

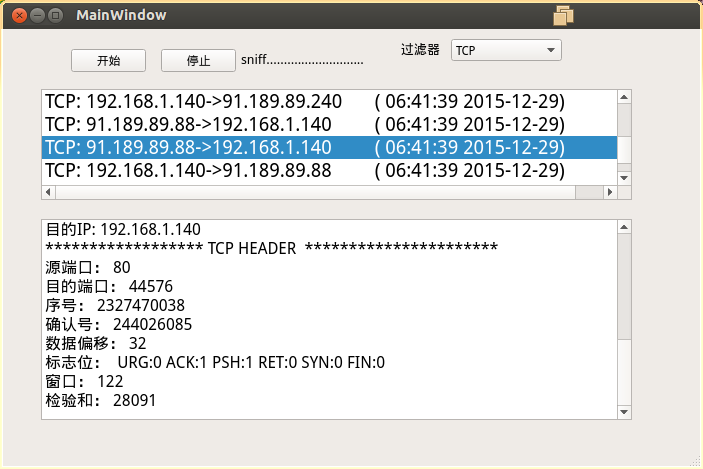
(udpheader->check\_sum<<8);

}

**四．结果测试**







**五．设计心得**

截包程序设计的基础是懂得使用RAW的SOCKET套接字，之前用SOCKET编程一般是用UDP和TCP，对于RAW 没怎么了解决，通过这次设计，对RAW 的套接字又有一种新的认识。

本设计中的难点还是在于数据包的解析，在设计过程中必须要对各类型的首部格式清楚，本次设计当然加深了MAC、IP、TCP、ICMP、UDP的首部的理解。

本设计的一个阻碍为忽略了本地字节序和网络字节序是不同的，造成对包的解析的时候总是得不到正确的数据，这个问题困扰了很久，一直以为是结构体设计不正确，但是在网络上也又找不到问题所在，然后又找各种资料来研究了这几个协议的首部格式。确定格式没理解错后，决定自己把截到的数据转成十六进制打印出来，一个个字节也分析，最终才发现原来是字节序的问题，本地是小端的而网络序是大端的，这才解决了这个问题。

本次设计巩固到的知识有：RAW套接字的使用，数据链路层、网络层和传输层常用的协议的首部格式的理解（MAC、IP、ICMP、TCP、UDP等），大小端的理解，当然还有socket编程和C和C++语言。