

# 嗅探器

## 一、捕获数据包

### 1.1 定义各种协议头部格式结构体

包含：以太网帧(Ethernet), IPV4(Internet Protocol Version 4), TCP(Transmission Control Protocol), UDP(User Data Protocol), Http(Hypertext ), DNS()

①macAddress 结构体

```
// MAC address structure
struct MACAddress {
    u_char b1, b2, b3, b4, b5, b6;
    MACAddress(){}
    MACAddress(const u_char B1, const u_char B2, const u_char B3, const u_char B4, const u_char B5, const u_char B6){
        b1 = B1;
        b2 = B2;
        b3 = B3;
        b4 = B4;
        b5 = B5;
        b6 = B6;
    }
};
```

②IP 结构体

```
// IP address structure
struct IPAddress {
    u_char b1, b2, b3, b4;
};
```

③以太网帧结构体

```
// Ethernet header structure
struct EthernetHeader {
    MACAddress destAddress;
    MACAddress srcAddress;
    u_short type;
};
```

④IPV4 头部结构体

```
// IP header structure
struct IPHeader {
    u_char ver_ihl, tos;
    u_short len, id, flags_offset;
    u_char ttl, proto;
    u_short checksum;
    IPAddress srcAddress, destAddress;
};
```

⑤UDP 头部结构体

```
// UDP header structure
struct UDPHeader {
    u_short srcPort, destPort, len, checksum;
};
```

⑥TCP 头部结构体

```
// TCP header structure
struct TCPHeader {
    u_short srcPort, destPort;
    u_int sequenceNum, ackNum;
    u_short flags, window, checksum, urgentPointer;
};
```

## ⑦DNS 头部结构体

```
// DNS header structure
struct DNSHeader {
    u_short id, flags, questCount, ansCount, authRecordCount, addRecordCount;
};
```

## 1.2 通过 winpcap 查看本地网络接口

```
//获取设备接口列表
pcap_if_t *alldevs;
char errbuf[PCAP_ERRBUF_SIZE];
if (pcap_findalldevs(&alldevs, errbuf) == -1) {
    return;
}
int pos = 0;
for (pcap_if_t *dev = alldevs; dev != nullptr; dev = dev->next) {
    QString device = QString::number(++pos) + ". " + QString::fromStdString(dev->name)
        + "(" + QString::fromStdString(dev->description) + ")";
    QAction *action = menuInterfaces->addAction(device);
    action->setCheckable(true);
    action->setData(QVariant::fromValue(MenuItemDevice(dev->name, dev->description)));
    actionGroup->addAction(action);
}
//设置触发器
connect(actionGroup, &QActionGroup::triggered, this, &MainWindow::onInterfaceSelected);
connect(stopMenu, &QMenu::aboutToShow, this, &MainWindow::onClickedStopMenu);
//释放接口权柄
pcap_freealldevs(alldevs);|
```

## 1.3 通过 winpcap 捕获本地某个网络接口的数据包

```
void PacketCaptureThread::run()
{
    /* deviceName是接口名称 */
    char errbuf[PCAP_ERRBUF_SIZE];
    pcap_t *handle=pcap_open(deviceName.toStdString().c_str(),
                             65536,
                             PCAP_OPENFLAG_PROMISCUOUS,
                             1000,
                             NULL,
                             errbuf);

    if (handle == nullptr) return;
    if(pcap_datalink(handle) != DLT_EN10MB) return;
    struct bpf_program fp; // The compiled filter
    char filter_exp[] = "ip and (tcp or udp) and (port 53 or port 80 or port 443)";

    // Compile and set the filter
    if (pcap_compile(handle, &fp, filter_exp, 0, 0) == -1) return;
    if (pcap_setfilter(handle, &fp) == -1) return;

    //共享变量
    handleShare = handle;
    (StopCapture::timer).start();

    pcap_loop(handle, 0, packetHandler, reinterpret_cast<u_char*>(this)); // 0 means loop forever
    pcap_close(handle);
}
```

## 二、 解析数据包

### 2.1 解析以太网帧

通过调用 processEthernet(EthernetHeader\*)函数来解析以太网帧，其中

```
EthernetHeader *ethernetHeader = (EthernetHeader*) (packet)
```

```
void MainWindow::processEthernet(const EthernetHeader *ethernetHeader){
    //目的MAC地址
    QString destMac = MacToQStr(ethernetHeader->destAddress);
    //源MAC地址
    QString srcMac = MacToQStr(ethernetHeader->srcAddress);
    //帧类型
    unsigned short macType = ntohs(ethernetHeader->type);
}
```

### 2.2 解析 IP 数据包

调用 processIP(IPHeader\*)来解析 IP 数据包，其中

```
IPHeader *ipHeader = (IPHeader*)((u_char*)ethernetHeader + ETHER_HEADER_SIZE);
```

ETHER\_HEADER\_SIZE 为以太网帧首部长度(14)(偏移量)

```

void MainWindow::processIp(const IPHeader *ipHeader){
    /* 处理IP数据包 */
    u_int ipHeaderLen = (ipHeader->ver_ihl & 0xf) * 4;          //ip头部长度
    u_char ipVersion = (ipHeader->ver_ihl & 0xf0)>>4;          //ip版本号
    u_char ipServiceType = ipHeader->tos;                        //ip服务类型
    unsigned short ipLength = ntohs(ipHeader->len);              //ip数据包总长度
    u_short ipIdentify = ntohs(ipHeader->id);                    //标识
    u_short ipFragments = (ntohs(ipHeader->flags_offset))>>13;  //切片标志位
    u_short ipOffset = (ntohs(ipHeader->flags_offset) & 0x1FFF); //片偏移
    u_char TTL = ipHeader->ttl;                                   //生存时间
    QString Protocol = protocolNameFromNumber(ipHeader->proto); //协议
    u_short ipChecksum = ntohs(ipHeader->checksum);              //校验和
    //源IP地址
    QString srcIp = IpToQStr(ipHeader->srcAddress);
    //目的IP地址
    QString destIp = IpToQStr(ipHeader->destAddress);
}

```

## 2.3 解析传输层数据包

### 2.3.1 解析 TCP 数据包

通过调用 processTcp(TCPHeader\*)函数来解析 TCP 数据包，而

```

TCPHeader* tcpHeader = (TCPHeader*)((u_char*)ipHeader + ipHeaderLen); //获得tcp头部指针

```

其中 ipHeaderLen 为 IPV4 数据包头长度

```

void MainWindow::processTcp(const TCPHeader *tcpHeader){
    //解析tcp数据包
    u_short srcPort = ntohs(tcpHeader->srcPort);          //源端口
    u_short destPort = ntohs(tcpHeader->destPort);         //目的端口
    u_int sequenceNum = ntohl(tcpHeader->sequenceNum);     //序号
    u_int ackNum = ntohl(tcpHeader->ackNum);                //确认号
    u_short tcpHeaderLen = ((ntohs(tcpHeader->flags) & 0xF000)>>12) * 4; //tcp头部长度
    u_char tcpRemain = (ntohs(tcpHeader->flags) & 0x0FC0); //6位保留
    u_char tcpFlags = u_char(ntohs(tcpHeader->flags) & 0x003F); //6位标志位(ACK、SYN.....)
    u_short tcpWindows = ntohs(tcpHeader->window);         //窗口大小
    QString tcpChecksum = QString::number(ntohs(tcpHeader->checksum),16).toUpper(); //校验和
    u_short tcpUrgentPoint = ntohs(tcpHeader->urgentPointer); //紧急指针
}

```

### 2.3.2 解析 UDP 数据包

通过调用 processUDP(UDPHeader\*)函数来解析 UDP 数据包，而

```

UDPHeader* udpHeader = (UDPHeader*)((u_char*)ipHeader + ipHeaderLen);

```

其中 ipHeaderLen 为 IPV4 数据包头长度

```

void MainWindow::processUdp(const UDPHeader *udpHeader){
    //解析UDP数据包
    u_short srcPort = ntohs(udpHeader->srcPort);          //源端口
    u_short destPort = ntohs(udpHeader->destPort);         //目的端口
    u_short udpLength = ntohs(udpHeader->len);              //udp数据包总长度
    u_short udpChecksum = ntohs(udpHeader->checksum);       //校验和
}

```

## 2.4 解析应用层数据包

### 2.4.1 解析 Http 数据包

通过调用库函数 `httpData` 来解析 http 数据包，其中 `httpHeaderOffset` 为 http 数据包开始出指针，`httpLength` 为 http 数据包长度

```
void MainWindow::processHTTP(const u_char *httpHeaderOffset, u_int httpLength){
    const char* httpOffset = reinterpret_cast<const char*>(httpHeaderOffset);
    QByteArray httpData(httpOffset, httpLength);
    QString httpString(httpData);
}
```

### 2.4.2 解析 DNS 数据包

```
void MainWindow::processDNS(const DNSHeader *dnsHeader){
    u_short dnsIdentifier = ntohs(dnsHeader->id);
    u_short dnsFlags = ntohs(dnsHeader->flags);
    u_short dnsQuestCount = ntohs(dnsHeader->questCount);
    u_short dnsAnsCount = ntohs(dnsHeader->ansCount);
    u_short dnsAuthCount = ntohs(dnsHeader->authRecordCount);
    u_short dnsAddCount = ntohs(dnsHeader->addRecordCount);
}
```

## 2.5 过滤

`PacketList` 在 QT 中用 `TableView` 展示，过滤规则在自定义的 `CustomFilterProxyModel` 过滤模型中定义的，过滤规则就是比较输入的 IP 地址、端口号、以及选择的协议（若没有输入则默认是选择全部）



```

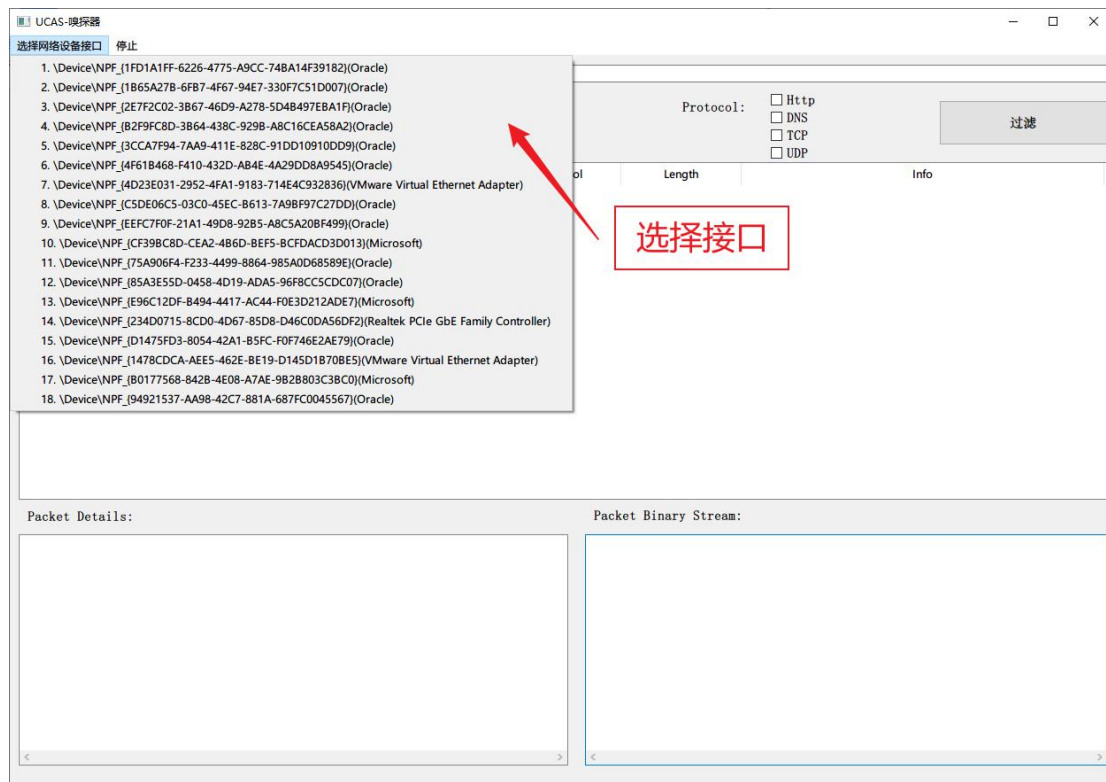
// 获取数据
QString srcIp = sourceModel()->data(index2).toString(); //源IP
QString destIp = sourceModel()->data(index3).toString(); //目的IP
QString protocol = sourceModel()->data(index4).toString(); //协议
QString srcPort = sourceModel()->data(index7).toString(); //源端口
QString destPort = sourceModel()->data(index8).toString(); //目的端口
int tcp_or_udp = sourceModel()->data(index9).toInt(); //传输层协议 tcp|udp|其他

bool addr=false,por=false,proto=false;
if(address=="") addr=true;
else if(address==srcIp||address==destIp) addr=true;
else addr=false;
if(port=="") por=true;
else if(port==srcPort||port==destPort) por=true;
else por=false;
if(!tcp && !udp && !http && !dns) proto=true;
if(protocol=="Http" && http) proto=true;
if(protocol=="DNS" && dns) proto=true;
if((tcp_or_udp==0) && tcp) proto=true;
if((tcp_or_udp==1) && udp) proto=true;
if (addr && por && proto) {
    return true;
} else {
    return false; |
}

```

## 三、捕获结果

### 3.1 DeviceList



### 3.2 PacketList

UCAS-嗅探器

选择网络设备接口 停止

你选择的接口是: \Device\NPF\_{CF39BC8D-CEA2-4B6D-BEF5-BCFDACD3D013} (Microsoft)

设备名+设备描述

IpAddress: Port: Protocol: ☐ Http ☐ DNS ☐ TCP ☐ UDP 过滤

PacketList:

No.	Time	Source	Destination	Protocol	Length	Info
51	16.365356s	123.126.97.71	192.168.0.104	Https	211	[协议: Https] 443 → 9940 [ACK, PSH] Seq=3256456249 ...
52	16.365359s	192.168.0.104	123.126.97.71	Https	787	[协议: Https] 9940 → 443 [ACK, PSH] Seq=2000209101 ...
53	16.365361s	123.126.97.71	192.168.0.104	Https	211	[协议: Https] 443 → 9940 [ACK, PSH] Seq=3256456406 ...
54	16.365364s	192.168.0.104	123.126.97.71	Https	787	[协议: Https] 9940 → 443 [ACK, PSH] Seq=2000209834 ...
55	16.365366s	123.126.97.71	192.168.0.104	Https	211	[协议: Https] 443 → 9940 [ACK, PSH] Seq=3256456563 ...
56	16.365368s	192.168.0.104	123.126.97.71	Https	54	[协议: Https] 9940 → 443 [ACK] Seq=2000210567 Ack=325645672
57	16.365371s	192.168.0.104	192.168.0.1	DNS	85	[协议: DNS] 62470 → 53
58	16.365373s	192.168.0.104	192.168.0.1	DNS	71	[协议: DNS] 60973 → 53
59	17.366151s	192.168.0.104	192.168.0.1	DNS	82	[协议: DNS] 61240 → 53
60	17.366189s	192.168.0.104	192.168.0.1	DNS	71	[协议: DNS] 60973 → 53
61	17.366193s	192.168.0.104	192.168.0.1	DNS	80	[协议: DNS] 55100 → 53

选中的数据包

### 3.3 PacketDetails

Packet Details:

Frame 57: 85 bytes captured (680 bits) on interface \Device\NPF\_{CF39BC8D-CEA2-4B6D-BEF5-B...}

▼ Ethernet II, SRC: (50:EB:71:14:36:5B), DST: (08:40:F3:57:A2:50)

Destination: 08:40:F3:57:A2:50

Source: 50:EB:71:14:36:5B

Type: 0x0800

▼ Internet Protocol Version 4, Src: 192.168.0.104, Dst: 192.168.0.1

0100 .... = Version: 4

.... 0101 = Header Length: 20 bytes (5)

IP Service Type: 0

Total Length: 71

Identification: 0xF8E7 (63719)

000. .... = Flags: 0x0

... 0 0000 0000 0000 = Fragment Offset: 0

Packet Details:

Source Address: 192.168.0.104

Destination Address: 192.168.0.1

▼ User Datagram Protocol, Src Port: 62470, Dst Port: 53

Source Port: 62470

Destination Port: 53

Length: 51

Checksum: 0x81FE

UDP payload: (43 bytes)

▼ Domain Name System

Transaction ID: 0xCF80

Flags: 0x0100

Question: 1

Answer RRs: 0

### 3.4 BinaryData in Hex

调用 `printDataBinaryInHex`

```
void MainWindow::printDataBinaryInHex(const u_char *packet, unsigned int packetLength){
    //以十六进制打印数据包的二进制流
    QByteArray plainText;
    plainText.append(reinterpret_cast<const char*>(packet), packetLength);
    QString formattedData = formatPacketData(plainText);
    ui->plainTextEdit->setPlainText(formattedData);
}
```

Packet Binary Stream:

```

0000  08 40 F3 57 A2 50 50 EB 71 14 36 5B 08 00 45 00  .@.W.PP.q.6[..E.
0010  00 47 F8 E7 00 00 80 11 00 00 C0 A8 00 68 C0 A8  .G.....h..
0020  00 01 F4 06 00 35 00 33 81 FE CF 80 01 00 00 01  .....5.3.....
0030  00 00 00 00 00 00 0A 69 6E 69 74 2D 70 30 31 73  .....init-p0ls
0040  74 04 70 75 73 68 05 61 70 70 6C 65 03 63 6F 6D  t.push.apple.com
0050  00 00 01 00 01                                     .....

```