**网络嗅探软件设计**

# 1引言

### 1.1课程设计的目的

本课程设计的目的就是设计一个捕捉并解析IP数据包的程序，并根据这个程序，说明IP数据包的结构及网络各层协议的相关问题，从而对计算机网络各层的工作原理有更好的理解和认识。通过编写该程序的过程，可以加深对计算机网络各层协议功能的了解，提高C++编程语言的掌握程度，学习WinSock API函数，以及MFC图形界面的设计。

### 1.2课程设计的意义

计算机之间进行通信时，交互的所有信息都封装在数据包中。因此，通过采集网络数据并对其进行相应的分析，可以清楚地了解到进行通信的计算机的通信目的。首先，分析采集到的数据包，可以确定网络是否受到攻击入侵；其次，也可以使用采集到的数据包来分析网络应用程序可能出现的问题的原因；此外，通过网络数据采集和统计可以清楚的了解整个网络在各个时段内的网络负载情况，从而判断网络使用得是否合理。

### 1.3 设计平台

Visual C++ 2017 MFC

# 2设计原理

### 2.1分组捕获原理

使用原始套接字，绑定本机网卡具体的IP地址，并设为混杂模式后就可以使用recv()函数接收所有收到的IP数据包了。

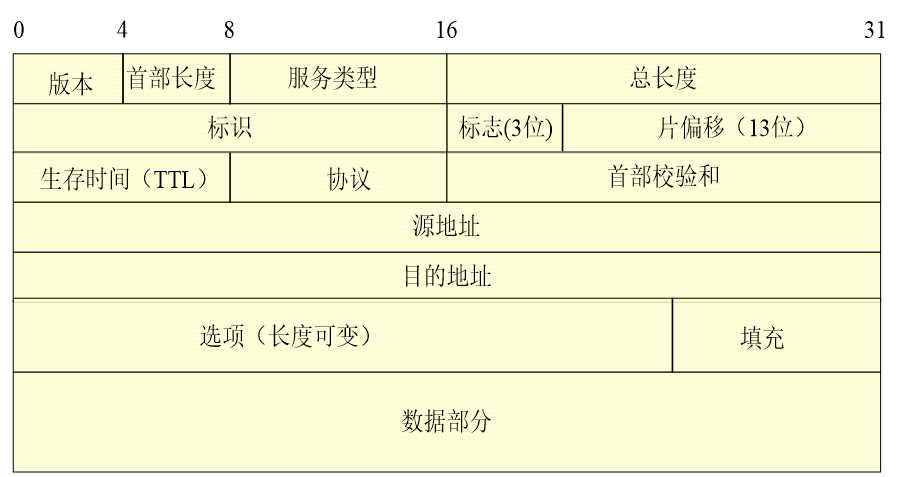
原始套接字收到的数据包括IP首部在内的完整的IP分组，并且首部所有数值字段均为网络字节顺序。

### 2.2分组解析原理

要解析各层协议，首先需要清楚各层协议包首部的格式，了解各字段的取值及含义。根据首部格式以及各自段所占长度，可以定义出相应的结构体，通过对应的结构体指针就可以读写首部各字段的值。

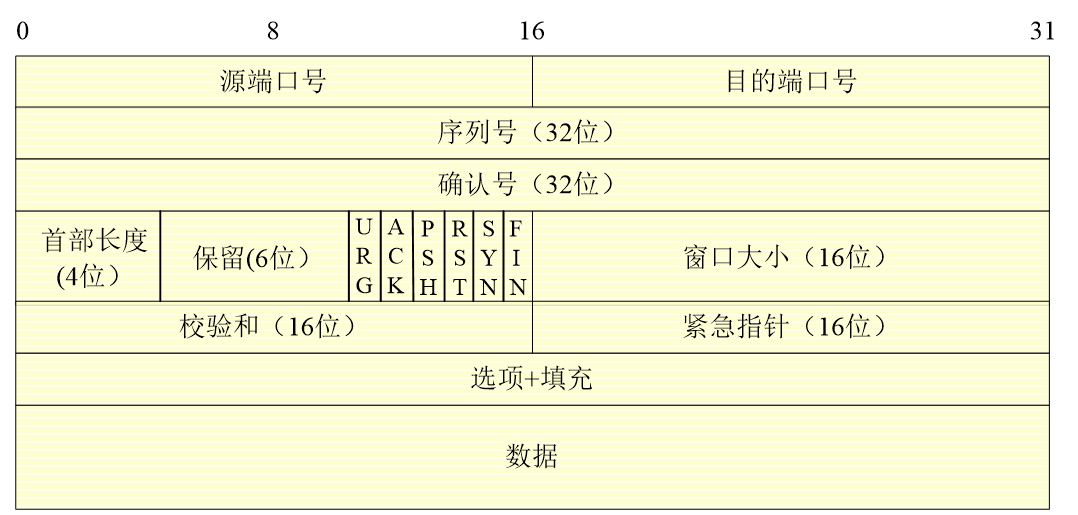
当对捕获的IP分组的各层协议进行解析时，从协议层次上说应从下及上，从数据包封装格式上说应由外及里逐层进行解析，即先解析IP层再解析传输层。首先解析IP首部，然后根据IP首部中协议字段的值判断IP分组的数据部分是TCP报文段、UDP数据报、ICMP报文还是其它协议的数据包，再根据不同类型的上层协议作进一步的解析。传输层若为TCP或UDP协议，又可根据其端口号判断其应用层使用的协议。（注：解析过程中整数字段要根据字段长度选择适当的函数从网络字节顺序转换为主机字节顺序）

##### 2.2.1 IP首部



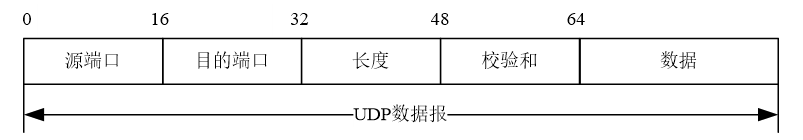
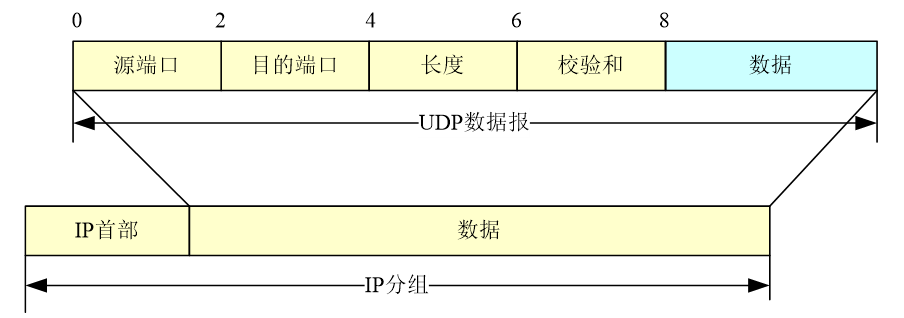
typedef struct \_IPHeader // 20字节的IP头{ unsigned char iphVerLen; // 版本号和头长度（各占4位） unsigned char ipTOS; // 服务类型 unsigned short ipLength; // 封包总长度，即整个IP报的长度 unsigned short ipID; // 封包标识，惟一标识每一个数据报 unsigned short ipFlags; // 标志 unsigned char ipTTL; // 生存时间，就是TTL unsigned char ipProtocol; // 协议，可能是TCP、UDP、ICMP等 unsigned short ipChecksum; // 校验和 unsigned long ipSource; // 源IP地址 unsigned long ipDestination; // 目的IP地址} IPHeader;

##### 2.2.2 TCP首部



typedef struct \_TCPHeader　　{　　 unsigned short sourcePort; // 16位源端口号　　 unsigned short destinationPort; // 16位目的端口号　　 unsigned long sequenceNumber; // 32位序列号　　 unsigned long acknowledgeNumber; // 32位确认号　　 char dataoffset; //高4位表示数据偏移　　 char flags; //低6位URG、ACK、PSH、RST、SYN、FIN标志位　　 unsigned short windows; // 16位窗口大小　　 unsigned short checksum; // 16位校验和　　 unsigned short urgentPointer; // 16位紧急数据偏移量 　　} TCPHeader;

##### 2.2.3 UDP首部



typedef struct \_UDPHeader{ unsigned short sourcePort; // 源端口号 unsigned short destinationPort; // 目的端口号 unsigned short len; // 包长度 unsigned short checksum; // 校验和} UDPHeader;

##### 2.2.4 应用层服务对应端口号

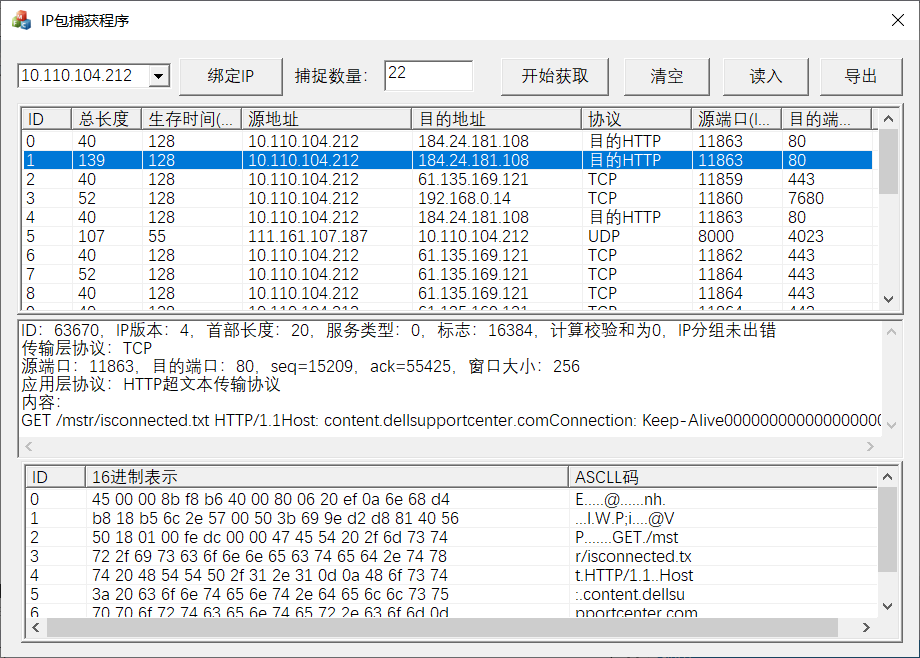
|  |  |  |
| --- | --- | --- |
| **常用服务** | **协议** | **端口号** |
| HTTP | TCP | 80 |
| HTTPS | TCP | 443 |
| DNS | TCP | 53 |
| DNS | UDP | 53 |
| POP3（邮件） | TCP | 110 |
| SMTP（邮件） | TCP | 25 |
| Telnet | TCP | 23 |
| OICQ（腾讯QQ） | 首选UDP | 8000 |
| IMAP | TCP | 143 |
| PPTP | TCP | 1723 |
| DHCP | UDP | 67 |
| IPSec | UDP | 500 |
| FTP | TCP | 20、21 |

### 2.3 字段提取与格式转换

根据变量存储位数的不同，对应首部字段长度，选择合适的变量存储相应字段。通过位运算进行字段提取、16进制转换和校验和的计算，根据ASCLL码表转换显示33到123可显示的ASCLL字符。

### 3软件设计模块3.1图形界面

1. 操作按钮部分（Button）
2. 主机IP地址下拉列表（Extended Combo Box）
3. IP分组首部信息列表（List Control）
4. IP分组详细信息部分（List Box）
5. 16进制及ASCLL码显示部分（List Control）



### 3.2功能模块

##### 1．初始化

加载WinSock动态连接库，获取本机可选IP地址并显示，初始化列表列名。

// TODO: 在此添加额外的初始化代码

CRect rect;

GetDlgItem(IDC\_LIST3)->GetWindowRect(&rect);

ScreenToClient(&rect);//转换为对话框上的客户坐标

int x = rect.Width();

list\_msg.SetExtendedStyle(list\_msg.GetExtendedStyle() | LVS\_EX\_FULLROWSELECT | LVS\_EX\_GRIDLINES);// 为列表视图控件添加全行选中和栅格风格

list\_msg.InsertColumn(0, "ID", LVCFMT\_LEFT, x / 8 - 60);

list\_msg.InsertColumn(1, "总长度", LVCFMT\_LEFT, x / 8 - 40);

list\_msg.InsertColumn(2, "生存时间(TTL)", LVCFMT\_LEFT, x / 8 - 10);

list\_msg.InsertColumn(3, "源地址", LVCFMT\_LEFT, x / 8 + 60);

list\_msg.InsertColumn(4, "目的地址", LVCFMT\_LEFT, x / 8 + 60);

list\_msg.InsertColumn(5, "协议", LVCFMT\_LEFT, x / 8);

list\_msg.InsertColumn(6, "源端口(ICMP类型)", LVCFMT\_LEFT, x / 8 - 20);

list\_msg.InsertColumn(7, "目的端口(ICMP代码)", LVCFMT\_LEFT, x / 8 - 20);

list\_sj.SetExtendedStyle(list\_sj.GetExtendedStyle() | LVS\_EX\_FULLROWSELECT | LVS\_EX\_GRIDLINES);// 为列表视图控件添加全行选中和栅格风格

list\_sj.InsertColumn(0, "ID", LVCFMT\_LEFT, 60);

list\_sj.InsertColumn(1, "16进制表示", LVCFMT\_LEFT, x / 2 + 40);

list\_sj.InsertColumn(2, "ASCLL码", LVCFMT\_LEFT, x / 2 - 110);

WSADATA wsaData;

int ret;

if ((ret = WSAStartup(MAKEWORD(2, 2), &wsaData)) != 0)

{

MessageBox("初始化WinSock出错!");

return 0;

}

/\*\*获取本地IP地址\*\*/

char sHostName[256];

struct hostent \*hptr;

char \*\*pptr;

gethostname(sHostName, sizeof(sHostName));

if ((hptr = gethostbyname(sHostName)) == NULL)

{

CString str1 = "未能获取本地IP地址！错误码：";

CString str2;

str2.Format("%d", WSAGetLastError());

MessageBox(str1 + str2);

WSACleanup();

return 0;

}

pptr = hptr->h\_addr\_list;

/\*\*在屏幕上显示本机所有IP地址\*\*/

list\_ip.InsertString(0, "请选择IP地址");

int i = 1;

while (\*pptr != NULL)

{

list\_ip.InsertString(i, inet\_ntoa(\*(struct in\_addr \*)(\*pptr)));

pptr++;

}

list\_ip.SetCurSel(0); //默认选中0号

memset((void\*)buff, 0, sizeof(buff));

n = 0;

unsigned long u1 = 1;

ioctlsocket(sRaw,FIONBIO,&u1);

##### 2．IP分组捕获

以按钮点击触发，根据用户输入的数量抓取相应的IP分组，暂时存储在已开辟的char类型数组空间中，并调用IP分组解析函数解析。

void CkcsjDlg::OnBnClickedButton1() //抓取IP包

{

// TODO: 在此添加控件通知处理程序代码

//抓取ip分组

list\_msg.DeleteAllItems();

memset((void\*)buff,0,sizeof(buff));

UpdateData(true);

n = atoi(number);

if (n == 0)

{

MessageBox("请输入你要获取IP包的数量！");

return;

}

if (n > 300)

{

MessageBox("请输入小于300的整数！");

return;

}

int nRet;

for (int i = 0; i < n; i++)

{

nRet = recv(sRaw, buff[i], 1520, 0);

if (nRet <= 0)

{

CString str1 = "抓取数据时出错！错误码：";

CString str2;

str2.Format("%d", WSAGetLastError());

//MessageBox(str1 + str2);

i--;

continue;

}

decode(i);

}

}

##### 3．IP分组首部解析

解析捕捉到的IP分组首部各字段，并添加到图形界面显示。

void CkcsjDlg::decode(int j) //解析IP首部

{

/\*\*解析IP包\*\*/

IPHeader \*pIPHdr = (IPHeader\*)buff[j];

in\_addr source, dest;

char szSourceIP[32], szDestIP[32];

/\*\*从IP头中取出源IP地址和目的IP地址\*\*/

source.S\_un.S\_addr = pIPHdr->ipSource;

dest.S\_un.S\_addr = pIPHdr->ipDestination;

strcpy(szSourceIP, inet\_ntoa(source));

strcpy(szDestIP, inet\_ntoa(dest));

int nHeaderLen = (pIPHdr->iphVerLen & 0xf) \* sizeof(ULONG); //IP头长度

CString id;

id.Format("%d", j);

list\_msg.InsertItem(j, id);

CString temp;

temp.Format("%d", ntohs(pIPHdr->ipLength));

list\_msg.SetItemText(j, 1, temp);

temp.Format("%d", (pIPHdr->ipTTL));

list\_msg.SetItemText(j, 2, temp);

list\_msg.SetItemText(j, 3, szSourceIP);

list\_msg.SetItemText(j, 4, szDestIP);

if (pIPHdr->ipProtocol == IPPROTO\_TCP)

{

TCPHeader \*pTCPHdr = (TCPHeader \*)(buff[j] + nHeaderLen);

CString str1, str2;

str1.Format("%d", ntohs(pTCPHdr->sourcePort));

str2.Format("%d", ntohs(pTCPHdr->destinationPort));

list\_msg.SetItemText(j, 6, str1);

list\_msg.SetItemText(j, 7, str2);

int dport = ntohs(pTCPHdr->destinationPort);

int sport = ntohs(pTCPHdr->sourcePort);

if (sport == 8080 || sport == 80)

{

list\_msg.SetItemText(j, 5, "源HTTP");

}

else if (dport == 8080 || dport == 80)

{

list\_msg.SetItemText(j, 5, "目的HTTP");

}

else if (dport == 53 || sport == 53)

{

list\_msg.SetItemText(j, 5, "DNS(TCP)");

}

else

{

list\_msg.SetItemText(j, 5, "TCP");

}

}

else if (pIPHdr->ipProtocol == IPPROTO\_UDP)

{

UDPHeader \*pUDPHdr = (UDPHeader \*)(buff[j] + nHeaderLen);

CString str1, str2;

str1.Format("%d", ntohs(pUDPHdr->sourcePort));

str2.Format("%d", ntohs(pUDPHdr->destinationPort));

list\_msg.SetItemText(j, 6, str1);

list\_msg.SetItemText(j, 7, str2);

int dport = ntohs(pUDPHdr->destinationPort);

int sport = ntohs(pUDPHdr->sourcePort);

if (dport == 53 || sport == 53)

list\_msg.SetItemText(j, 5, "DNS(UDP)");

else

list\_msg.SetItemText(j, 5, "UDP");

}

else if (pIPHdr->ipProtocol == IPPROTO\_ICMP)

{

list\_msg.SetItemText(j, 5, "ICMP");

ICMPHeader \*pICMPHdr = (ICMPHeader \*)(buff[j] + nHeaderLen);

CString str;

str.Format("%d", pICMPHdr->i\_code);

list\_msg.SetItemText(j, 7, str);

switch (pICMPHdr->i\_type)

{

case 0:

list\_msg.SetItemText(j, 6, "Echo Response.");

break;

case 8:

list\_msg.SetItemText(j, 6, "Echo request.");

break;

case 3:

list\_msg.SetItemText(j, 6, "Destination Unreachable.");

break;

case 11:

list\_msg.SetItemText(j, 6, "Datagram Timeout(TTL = 0).");

break;

default:

CString s;

s.Format("%d", pICMPHdr->i\_type);

list\_msg.SetItemText(j, 6, s);

}

}

else

{

list\_msg.SetItemText(j, 5, "其它协议");

}

}

##### 4．传输层及应用层协议解析（单击事件处理函数）

单击某一IP分组时触发，解析IP分组传输层协议首部各字段及应用层协议，并添加到图形界面显示。同时，调用进制转换及ASCLL码转换模块，将IP分组转换为16进制和ASCLL码，并添加到图形界面展示。

void CkcsjDlg::get\_ipmsg(int row) //解析传输层协议及应用层协议

{

CString ip\_msg = "";

IPHeader \*pIPHdr = (IPHeader\*)buff[row];

int nHeaderLen = (pIPHdr->iphVerLen & 0xf) \* sizeof(ULONG); //IP头长度

CString ver, hdrlen, ser, id, checknum, flag, checknum1;

ver.Format("%d", pIPHdr->iphVerLen >> 4);

hdrlen.Format("%d", (pIPHdr->iphVerLen & 0x0F) << 2);

ser.Format("%d", ntohs(pIPHdr->ipTOS));

id.Format("%d", ntohs(pIPHdr->ipID));

checknum.Format("%d", ntohs(pIPHdr->ipChecksum));

flag.Format("%d", ntohs(pIPHdr->ipFlags));

int check = ntohs(checksum((unsigned short\*)buff[row], sizeof(IPHeader)));

//checknum1.Format("%d", check);

if (check == 0)

list\_xx.AddString("ID：" + id + "，IP版本：" + ver + "，首部长度：" + hdrlen + "，服务类型：" + ser + "，标志：" + flag + "，计算校验和为0，IP分组未出错");

else

{

list\_xx.AddString("计算校验和不为0，IP分组出错！");

return;

}

if (pIPHdr->ipProtocol == IPPROTO\_TCP)

{

list\_xx.AddString("传输层协议：TCP");

TCPHeader \*pTCPHdr = (TCPHeader \*)(buff[row] + nHeaderLen);

int sport = ntohs(pTCPHdr->sourcePort);

int dport = ntohs(pTCPHdr->destinationPort);

CString s\_port, d\_port, seq, ack, winsize;

s\_port.Format("%d", sport);

d\_port.Format("%d", dport);

seq.Format("%d", ntohs(pTCPHdr->sequenceNumber));

ack.Format("%d", ntohs(pTCPHdr->acknowledgeNumber));

winsize.Format("%d", ntohs(pTCPHdr->windows));

list\_xx.AddString("源端口：" + s\_port + "，目的端口：" + d\_port + "，seq=" + seq + "，ack=" + ack + "，窗口大小：" + winsize);

if (sport == 8080 || sport == 80)

{

list\_xx.AddString("应用层协议：HTTP超文本传输协议");

list\_xx.AddString("内容：");

int tcpHeaderLen = (pTCPHdr->dataoffset >> 4 & 0xf) \* sizeof(ULONG);//TCP首部长度

char \*data = (buff[row] + nHeaderLen) + nHeaderLen;

int len1 = ntohs(pIPHdr->ipLength) - tcpHeaderLen;

data[len1] = '\0';

CString str = "";

for (int i = 0; i < len1; i++)

{

CString s;

s.Format("%x", data[i]);

str += s + " ";

}

list\_xx.AddString(data);

}

else if (dport == 8080 || dport == 80)

{

list\_xx.AddString("应用层协议：HTTP超文本传输协议");

list\_xx.AddString("内容：");

int tcpHeaderLen = (pTCPHdr->dataoffset >> 4 & 0xf) \* sizeof(ULONG);//TCP首部长度

char \*data = (buff[row] + nHeaderLen) + nHeaderLen;

int len1 = ntohs(pIPHdr->ipLength) - tcpHeaderLen;

data[len1] = '\0';

CString str = "";

for (int i = 0; i < len1; i++)

{

CString s;

s.Format("%x", data[i]);

str += s + " ";

}

list\_xx.AddString(data);

}

else if (dport == 110 || sport == 110)

{

list\_xx.AddString("应用层协议：POP3邮局协议");

}

else if (dport == 143 || sport == 143)

{

list\_xx.AddString("应用层协议：IMAP交互式邮件存取协议");

}

else if (dport == 25 || sport == 25)

{

list\_xx.AddString("应用层协议：SMTP简单邮件传输协议");

}

else if (dport == 21 || sport == 21)

{

list\_xx.AddString("应用层协议：FTP(控制)文件传输协议");

}

else if (dport == 20 || sport == 20)

{

list\_xx.AddString("应用层协议：FTP(数据)文件传输协议");

}

else if (dport == 443 || sport == 443)

{

list\_xx.AddString("应用层协议：HTTPS安全文本传输协议");

}

else if (dport == 3389 || sport == 3389)

{

list\_xx.AddString("应用层协议：终端服务");

}

else if (dport == 1723 || sport == 1723)

{

list\_xx.AddString("应用层协议：PPTP点对点隧道协议");

}

else if (dport == 23 || sport == 23)

{

list\_xx.AddString("应用层协议：Telnet远程登录");

}

else if (dport == 53 || sport == 53)

{

list\_xx.AddString("应用层协议：DNS域名解析");

}

else

{

list\_xx.AddString("应用层协议：其它");

}

}

else if (pIPHdr->ipProtocol == IPPROTO\_UDP)

{

list\_xx.AddString("传输层协议：UDP");

UDPHeader \*pUDPHdr = (UDPHeader \*)(buff[row] + nHeaderLen);

int dport = ntohs(pUDPHdr->destinationPort);

int sport = ntohs(pUDPHdr->sourcePort);

CString s\_port, d\_port, len, checknum;

s\_port.Format("%d", sport);

d\_port.Format("%d", dport);

len.Format("%d", ntohs(pUDPHdr->len));

checknum.Format("%d", ntohs(pUDPHdr->checksum));

list\_xx.AddString("源端口：" + s\_port + "，目的端口：" + d\_port + "，包长度：" + len + "，校验和：" + checknum);

if (dport == 53 || sport == 53)

{

list\_xx.AddString("应用层协议：DNS域名解析");

}

else if (dport == 123 || sport == 123)

{

list\_xx.AddString("应用层协议：NTP网络时间协议");

}

else if (dport == 1645 || sport == 1645)

{

list\_xx.AddString("应用层协议：RADIUS用户远程认证服务");

}

else if (dport == 67 || sport == 67)

{

list\_xx.AddString("应用层协议：DHCP动态主机配置协议");

}

else if (dport == 161 || sport == 161)

{

list\_xx.AddString("应用层协议：SNMP简单网络管理协议");

}

else if (dport == 500 || sport == 500)

{

list\_xx.AddString("应用层协议：Ipsec");

}

else if (dport == 8000 || sport == 8000)

{

list\_xx.AddString("应用层协议：OICQ腾讯QQ");

}

else

{

list\_xx.AddString("应用层协议：其它");

}

}

else if (pIPHdr->ipProtocol == IPPROTO\_ICMP)

{

list\_xx.AddString("ICMP");

ICMPHeader \*pICMPHdr = (ICMPHeader \*)(buff[row] + nHeaderLen);

switch (pICMPHdr->i\_type)

{

case 0:

list\_xx.AddString("ICMP协议类型：Echo Response.");

break;

case 8:

list\_xx.AddString("ICMP协议类型：Echo request.");

break;

case 3:

list\_xx.AddString("ICMP协议类型：Destination Unreachable.");

break;

case 11:

list\_xx.AddString("ICMP协议类型：Datagram Timeout(TTL = 0).");

break;

}

}

}

##### 5．进制转换及ASCLL码转换

通过位运算将2进制数据转换为10进制过渡，再根据10进制转换为16进制，以及通过10进制ASCLL码值转换为char类型的ASCLL码字符，并添加到图形界面展示。

void CkcsjDlg::get\_sj(int row) //转16进制和ascll码

{

CString ip\_sj[300], ip\_ascll[300];

IPHeader \*pIPHdr = (IPHeader\*)buff[row];

int nHeaderLen = (pIPHdr->iphVerLen & 0xf) \* sizeof(ULONG); //IP头长度

int len = ntohs(pIPHdr->ipLength);

int num = 0;

CString str[16] = { "0","1","2","3","4","5","6","7","8","9","a","b","c","d","e","f" };

for (int i = 0; i < len; i += 16)

{

CString ip\_msg1 = "", ip\_msg2 = "";

char ch[17];

for (int i = 0; i < 17; i++)

ch[i] = '.';

for (int j = 0; j < 16; j++)

{

if (i + j == len)

break;

ip\_msg1 += str[buff[row][i + j] >> 4 & 0xf];

ip\_msg1 += str[buff[row][i + j] & 0xf] + " ";

int t = buff[row][i + j] & 0xff;

if (t <= 126 && t >= 33)

ch[j] = char(t);

}

ch[16] = '\0';

ip\_sj[num] = ip\_msg1;

ip\_ascll[num] = ch;

num++;

}

for (int i = 0; i < num; i++)

{

CString t;

t.Format("%d", i);

list\_sj.InsertItem(i, t);

list\_sj.SetItemText(i, 1, ip\_sj[i]);

list\_sj.SetItemText(i, 2, ip\_ascll[i]);

}

}

# 6．校验和的计算

unsigned short CkcsjDlg::checksum(unsigned short\* buff, int size)

{

unsigned long cksum = 0;

while (size > 1)

{

cksum += \*buff++;

size -= sizeof(unsigned short);

}

if (size)

cksum += \*(char\*)buff; //当字节数是奇数时，最后一字节需要单独处理

//将32位的chsum高16位和低16位相加

cksum = (cksum >> 16) + (cksum & 0xffff);

cksum += (cksum >> 16);

return (unsigned short)(~cksum); //取值返回

}

##### 7．导入/导出功能

（1）导入：将数组中存储的捕捉到的原始IP数据包导出到文本文件；

void CkcsjDlg::OnBnClickedButton4() //导出

{

// TODO: 在此添加控件通知处理程序代码

if (n == 0)

{

MessageBox("无可导出内容！");

return;

}

UpdateData(true);

char szFilter[] = "文本文件(\*.txt)|\*.txt| All Files(\*.\*)|\*.\*||";

CFileDialog SaveDlg(false, ".txt", 0, 0, szFilter);

int x = SaveDlg.DoModal();

if (x == IDOK)

{

CFile fileSave;

try

{

fileSave.Open(SaveDlg.GetPathName(), CFile::modeCreate | CFile::modeWrite | CFile::typeBinary);

for (int i = 0; i < n; i++)

{

fileSave.Write(&buff[i], sizeof(buff[0]));

}

fileSave.Close();

MessageBox("导出成功！");

}

catch (CFileException \*e)

{

CString str;

str.Format("导出失败的原因是:%d", e->m\_cause);

fileSave.Abort();

e->Delete();

return;

}

}

}

（2）导出：从文本文件中读取IP原始数据包，并进行解析展示。

void CkcsjDlg::OnBnClickedButton6() //导入

{

// TODO: 在此添加控件通知处理程序代码

char szFilter[] = "文本文件(\*.txt)|\*.txt| All Files(\*.\*)|\*.\*||";

CFileDialog OpenDlg(true, ".txt", 0, 0, szFilter);

int x = OpenDlg.DoModal();

if (x == IDOK)

{

CFile fileOpen;

try

{

fileOpen.Open(OpenDlg.GetPathName(), CFile::modeRead | CFile::typeBinary);

n = fileOpen.GetLength() / 1520;

for (int j = 0; j < n; j++) {

fileOpen.Read(&buff[j], 1520);

}

fileOpen.Close();

}

catch (CFileException \*e)

{

CString temp;

temp.Format("读取失败原因是:%d", e->m\_cause);

MessageBox(temp);

fileOpen.Abort();

e->Delete();

return;

}

}

decode(n);

}

##### 8．清空功能

清空char数组中存取的IP数据包并把分组内容置0，清空图形界面显示内容。

void CkcsjDlg::OnBnClickedButton3() //清空按钮

{

// TODO: 在此添加控件通知处理程序代码

n = 0;

memset((void\*)buff, 0, sizeof(buff));

list\_msg.DeleteAllItems();

list\_sj.DeleteAllItems();

list\_xx.ResetContent();

}