

1 实验目的

1. 学习通过使用WinPcap捕获IP网络数据报。
2. 学习IP数据报校验和计算方法。
3. 初步掌握网络监听和分析技术的实现过程。
4. 加深对网络协议的理解。

2 实验要求

1. 本实验要求利用 WinPcap 提供的功能获取网络接口设备列表和各接口的详细信息，同时可以对任意一块网络接口卡进行 IP 数据报进行校验和验证。
2. 程序需拥有可视化界面，可以用Qt/MFC等框架。参考界面如下：

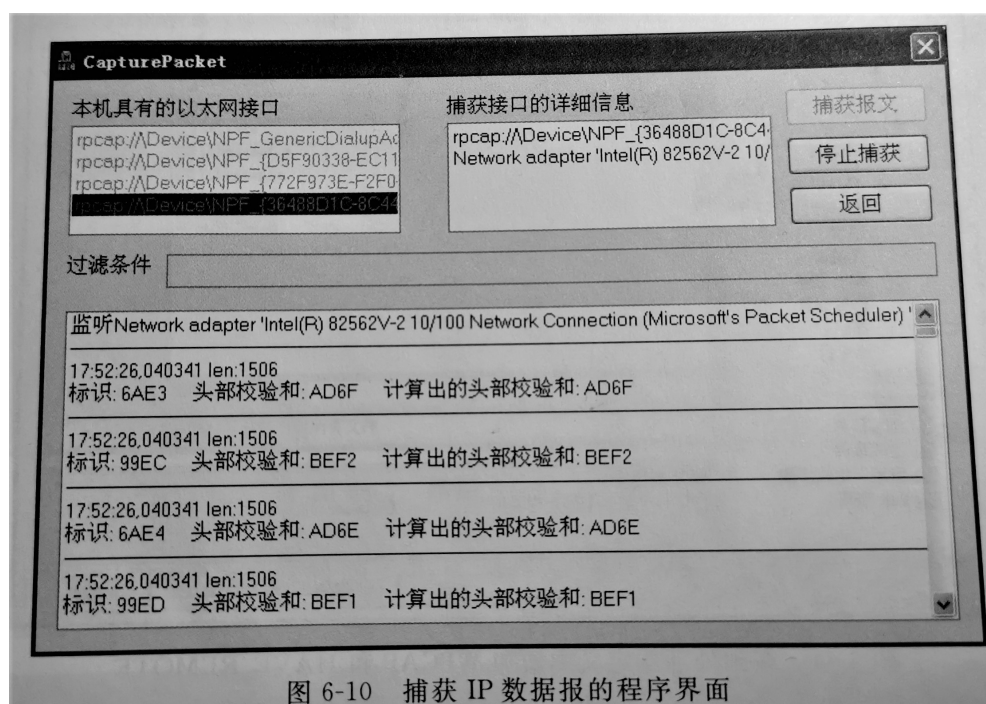


图 6-10 捕获 IP 数据报的程序界面

3. 所写的程序必须可以在实验室机器的环境下独立运行，现场测试时不接受调试项目文件，只允许运行exe程序。（配置：Windows 7 SP1 32位操作系统，已安装Winpcap）

3 实验环境

Windows 10、visual studio 2019 preview、MFC、release X86

4 实验原理

4.1 WinPcap简介

WinPcap 是由伯克利分组捕获库派生而来的分组捕获库，它是在Windows 操作平台上来实现对底层包的截取过滤。WinPcap 是 BPF 模型和 Libpcap 函数库在 Windows 平台下网络数据包捕获和网络状态分析的一种体系结构，这个体系结构是由一个核心的包过滤驱动程序，一个底层的动态连接库 Packet.dll 和一个高层的独立于系统的函数库 Libpcap 组成。底层的包捕获驱动程序实际为一个协议网络驱动程序，通过对 NDIS 中函数的调用为 Win95、Win98、WinNT、和 Win2000 提供一类似于 UNIX 系统下 Berkeley Packet Filter 的捕获和发送原始数据包的能力。Packet.dll 是对这个 BPF 驱动程序进行访问的 API 接口，同时它有一套符合 Libpcap 接口（UNIX 下的捕获函数库）的函数库。

WinPcap 包括三个部分：第一个模块NPF(Netgroup Packet Filter)，是一个虚拟设备驱动程序文件。它的功能是过滤数据包，并把这些数据包原封不动地传给用户态模块，这个过程中包括了一些操作系统特有的代码。第二个模块packet.dll为win32平台提供了一个公共的接口。不同版本的Windows系统都有自己的内核模块和用户层模块。Packet.dll用于解决这些不同。调用Packet.dll的程序可以运行在不同版本的Windows平台上，而无需重新编译。第三个模块 Wpcap.dll是不依赖于操作系统的。它提供了更加高层、抽象的函数。 packet.dll和Wpcap.dll： packet.dll直接映射了内核的调用。 Wpcap.dll提供了更加友好、功能更加强大的函数调用。WinPcap的优势提供了一套标准的抓包接口，与libpcap兼容，可使得原来许多UNIX平台下的网络分析工具快速移植过来便于开发各种网络分析工具，充分考虑了各种性能和效率的优化，包括对于NPF内核层次上的过滤器支持，支持内核态的统计模式，提供了发送数据包的能力

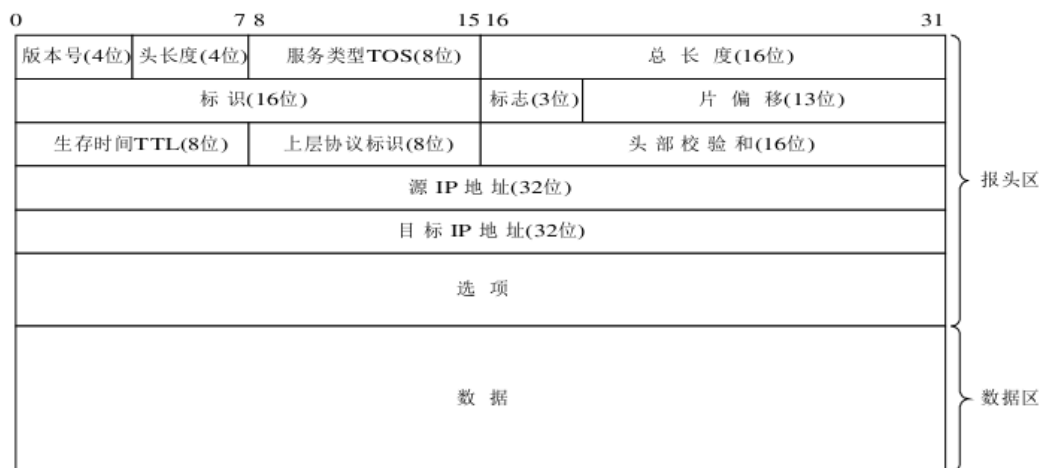
4.2 网络数据报捕获的原理

以太网（Ethernet）具有共享介质的特征，信息是以明文的形式在网络上传输，当网络适配器设置为监听模式（混杂模式，Promiscuous）时，由于采用以太网广播信道争用的方式，使得监听系统与正常通信的网络能够并联连接，并可以捕获任何一个在同一冲突域上传输的数据包。IEEE802.3 标准的以太网采用的是持续 CSMA 的方式，正是由于以太网采用这种广播信道争用的方式，使得各个站点可以获得其他站点发送的数据。运用这一原理使信息捕获系统能够拦截的我们所要的信息，这是捕获数据包的物理基础。以太网是一种总线型的网络，从逻辑上来看是由一条总线和多个连接在总线上的站点所组成各个站点采用上面提到的CSMA/CD 协议进行信道的争用和共享。每个站点（这里特指计算机通过的接口卡）网卡来实现这种功能。网卡主要的工作是完成对于总线当前状态的探测，确定是否进行数据的传送，判断每个物理数据帧目的地是否为本站地址，如果不匹配，则说明不是发送到本站的而将它丢弃。如果是的话，接收该数据帧，进行物理数据帧的 CRC 校验，然后将数据帧提交给LLC 子层。网卡具有如下的几种工作模式：1) 广播模式（Broad Cast Model）：它的物理地址（MAC）地址是 0xffffffff 的帧为广播帧，工作在广播模式的网卡接收广播帧。2) 多播传送（MultiCast Model）：多播传送地址作为目的物理地址的帧可以被组内的其它主机同时接收，而组外主机却接收不到。但是，如果将网卡设置为多播传送模式，它可以接收所有的多播传送帧，而不论它是不是组内成员。3) 直接模式（Direct Model）：工作在直接模式下的网卡只接收目地址是自己 Mac地址的帧。4) 混杂模式（Promiscuous Model）：工作在混杂模式下的网卡接收所有的流过网卡的帧，信包捕获程序就是在这种模式下运行的。网卡的缺省工作模式包含广播模式和直接模式，即它只接收广播帧和发给自己的帧。如果采用混杂模式，一个站点的网卡将接受同一网络内所有站点所发送的数据包这样就可以到达对于网络信息监视捕获的目的。

4.3 IP数据报

IP协议提供不可靠无连接的数据报传输服务，IP层提供的服务是通过IP层对数据报的封装与拆封来实现的。IP数据报的格式分为报头区和数据区两大部分，其中报头区是为了正确传输高层数据而加的各种控制信息，数据区包括高层协议需要传输的数据。

4.3.1 IP数据报的格式如下：



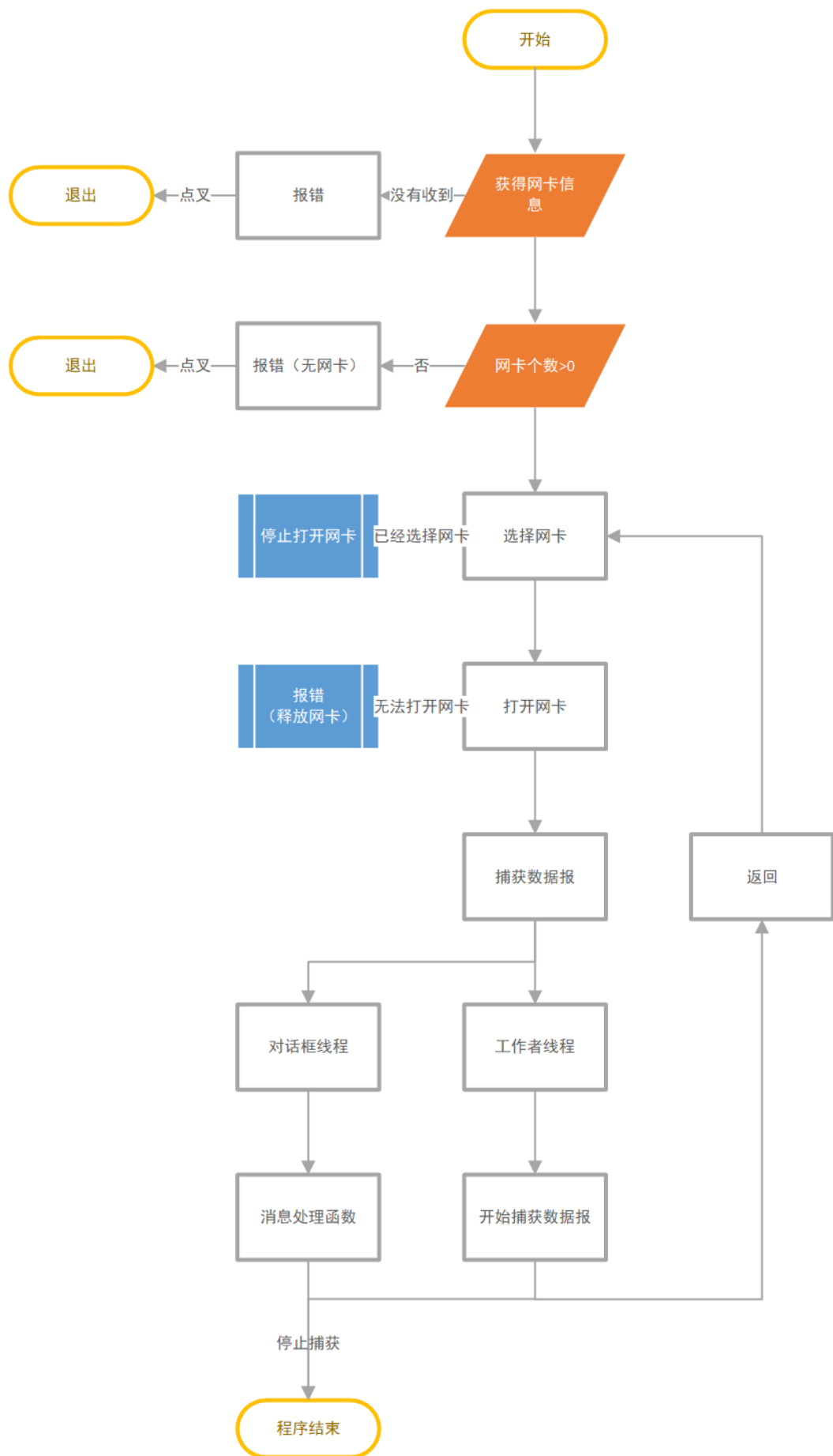
注意，上图表示的数据，最高位在左边，记为0位；最低位在右边，记为31位。在网络中传输数据时，先传输0~7位，其次是8~15位，然后传输16~23位，最后传输24~31位。由于TCP/IP协议头部中所有的二进制数在网络中传输时都要求以这种顺序进行，因此把它称为网络字节顺序。在实际编程中，以其他形式存储的二进制数必须在传输数据前使用网络编程API相应的函数把头部转换成网络字节顺序。

4.3.2 校验和

校验和：占用16位二进制数，用于协议头数据有效性的校验，可以保证IP报头区在传输时的正确性和完整性。头部检验和字段是根据IP协议头计算出的检验和，它不对头部后面的数据进行计算。

原理：发送端首先将检验和字段置0，然后对头部中每16位二进制数进行反码求和的运算，并将结果存在校验和字段中。由于接收方在计算过程中包含了发送方放在头部的校验和，因此，如果头部在传输过程中没有发生任何差错，那么接收方计算的结果应该是全1。

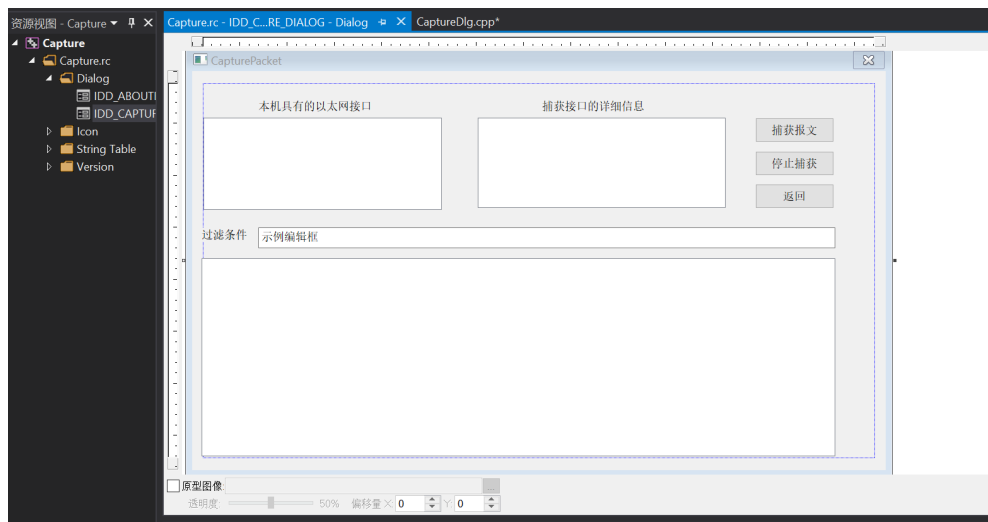
5 实验设计思路



6 程序实现步骤

6.1 创建MFC应用程序

1. 编译环境修改为release X86。
2. 在资源视图中构建以下界面。



3. 修改控件的ID与其他属性值，这里不再赘述。

6.2 创建基于WinPcap的应用程序

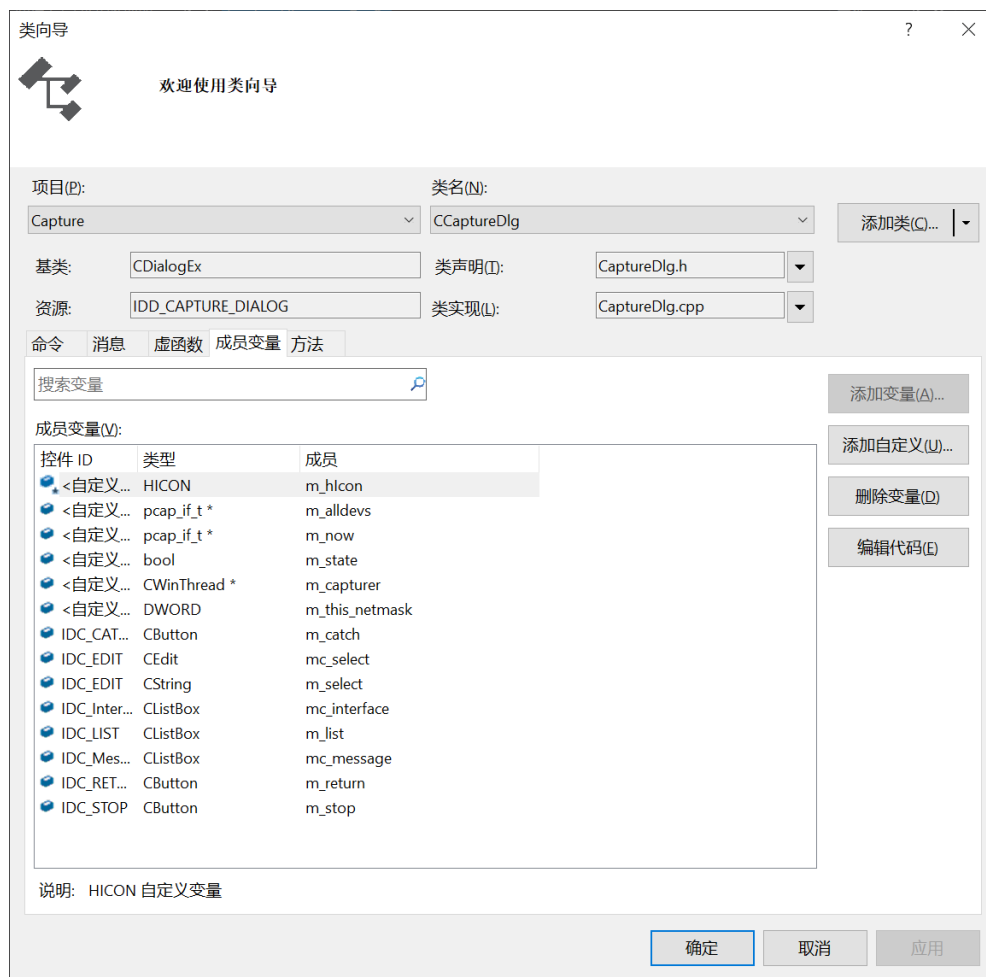
1. 添加 pcap.h 包含文件：如果一个源文件使用了 WinPcap 提供的函数，那么就要在该文件的开始位置增加 pcap.h 包含文件。
2. 增加与 WinPcap 有关的预处理定义：需要将 WPCAP 和 HAVE_REMOTE 两个标号添加到预处理定义中。
3. 把WinPcap的开发者工具包里的包含文件目录，与库文件添加到项目。

6.3 变量声明

1. 先申明全局变量

```
//全局变量
pcap_t* afx_adhandle;           //当前打开的网络接口
struct pcap_pkthdr* afx_header; //截获帧头部
const u_char* afx_pkt_data;    //截获帧数据
```

2. 然后使用类向导申明成员变量绑定相应的组件。



6.4 创建数据结构

```
#pragma pack(1)           //进入字节对齐方式
typedef struct FrameHeader_t { //帧首部
    BYTE    DesMAC[6]; // 目的地址
    BYTE    SrcMAC[6]; // 源地址
    WORD    FrameType; // 帧类型
} FrameHeader_t;
typedef struct IPHeader_t { //IP首部
    BYTE    Ver_HLen;
    BYTE    TOS;
    WORD    TotalLen;
    WORD    ID;
    WORD    Flag_Segment;
    BYTE    TTL;
    BYTE    Protocol;
    WORD    Checksum;
    ULONG    SrcIP;
    ULONG    DstIP;
} IPHeader_t;
typedef struct Data_t { //包含帧首部和IP首部的数据包
    FrameHeader_t    FrameHeader;
    IPHeader_t        IPHeader;
} Data_t;
#pragma pack() //恢复缺省对齐方式
```

6.5 构造函数。

```
CCaptureDlg::CCaptureDlg(CWnd* pParent /*=nullptr*/)
: CDialogEx(IDD_CAPTURE_DIALOG, pParent)
, m_select(_T(""))
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    m_this_netmask = 0;
    //获得本机的设备列表
    char errbuf[PCAP_ERRBUF_SIZE]; //错误信息缓冲区
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, //获取本机的接口设备
        NULL, //无需认证
        &m_alldevs, //指向设备列表首部
        errbuf //出错信息保存缓存区
    ) == -1)
    {
        MessageBox(L"获取本机设备列表失败: " + CString(errbuf),
        MB_OK);/*错误处理*/
    }
    m_now = m_alldevs;
}
```

6.6 初始化函数

在 `BOOL CCaptureDlg::OnInitDialog()` 函数中添加以下代码。

```
// TODO: 在此添加额外的初始化代码
//获取本机接口和IP地址
pcap_if_t* d; //指向设备链表首部的指针

for (d = m_alldevs; d != NULL; d = d->next) //显示接口列表
{
    mc_interface.AddString(CString(d->name)); //利用d->name获取
    该网络接口设备的名字
}

Update_Message();//更新信息
mc_interface.SetCurSel(0);
m_stop.EnableWindow(FALSE); //开始使“停止捕获”按钮失效
m_return.EnableWindow(FALSE); //开始使“返回”按钮失效
```

6.7 新增本地网络接口区域发生改变的函数

```
void CCaptureDlg::OnSelchangeInterfaceList()
{
    // TODO: 在此添加控件通知处理程序代码
    if (!m_state) {
        int N = mc_interface.GetCurSel(); //获取listbox被选中的行的数
        目
        m_now = m_alldevs;
    }
}
```

```

        while (N--)
        {
            m_now = m_now->next;
        }
        Update_Message();
    }
}

```

6.8 新增更新捕获接口的详细信息框函数

```

void CCaptureDlg::Update_Message()
{
    // TODO: 在此处添加实现代码。
    //更新捕获接口的详细信息
    mc_message.ResetContent(); //清除原有框的内容
    mc_message.AddString(CString(m_now->name)); //显示该网
    络接口设备的名字
    mc_message.AddString(CString(m_now->description)); //显示该网
    络接口设备的描述信息

    pcap_addr_t* a;
    a = m_now->addresses;
    for (a = m_now->addresses; a != NULL; a = a->next) {
        if (a->addr->sa_family == AF_INET) { //判断该地址是否IP地址
            CString output;
            DWORD temp_IP;

            temp_IP = ntohl((((struct sockaddr_in*)a->addr)-
            >sin_addr.s_addr); //显示IP地址
            output.Format(L"IP地址: %s", long2ip(temp_IP));
            mc_message.AddString(output);

            m_this_netmask = ntohl((((struct sockaddr_in*)a-
            >netmask)->sin_addr.s_addr); //显示地址掩码
            output.Format(L"地址掩码: %s",
            long2ip(m_this_netmask));
            mc_message.AddString(output);

            temp_IP = ntohl((((struct sockaddr_in*)a->broadaddr)-
            >sin_addr.s_addr); //显示广播地址
            output.Format(L"广播地址: %s", long2ip(temp_IP));
            mc_message.AddString(output);

        }
    }
}

```

6.9 点击开始捕获按钮函数


```

void CCaptureDlg::OnClickedCatch()
{
    // TODO: 在此添加控件通知处理程序代码
    m_state = true; //将是否进入捕获状态标记打开
    mc_interface.EnableWindow(FALSE);
    mc_select.EnableWindow(FALSE);
    //调整按钮状态
    m_catch.EnableWindow(FALSE);
    m_stop.EnableWindow(TRUE);
    m_return.EnableWindow(FALSE);

    m_list.ResetContent(); //清除原有框的内容

    //创建工作线程
    m_capturer = AfxBeginThread((AFX_THREADPROC)Capturer, NULL,
    THREAD_PRIORITY_NORMAL);
    if (m_capturer == NULL) {
        AfxMessageBox(L"启动捕获数据包线程失败!", MB_OK |
    MB_ICONERROR);
        return;
    }
    else /*打开选择的网卡 */
    {
        m_list.AddString(L"-----
    -----");
        m_list.AddString(L"监听" + CString(m_now->description) +
    L" 开始! ");
        m_list.AddString(L"-----
    -----");
        m_list.AddString(L"-----");
    }
    UpdateData(true);
}

```

6.10 点击停止捕获按钮函数

```

void CCaptureDlg::OnClickedStop()
{
    // TODO: 在此添加控件通知处理程序代码
    //调整按钮状态
    m_catch.EnableWindow(TRUE);
    m_stop.EnableWindow(FALSE);
    m_return.EnableWindow(TRUE);
    mc_select.EnableWindow(TRUE);
    m_state = false;
}

```

6.11 点击返回按钮函数

```

void CCaptureDlg::OnClickedReturn()

```

```

{
    // TODO: 在此添加控件通知处理程序代码
    //调整按钮状态
    m_state = false;
    mc_interface.EnableWindow(TRUE);
    m_catch.EnableWindow(TRUE);
    m_stop.EnableWindow(FALSE);
    m_return.EnableWindow(FALSE);
    mc_select.EnableWindow(TRUE);
    m_list.ResetContent(); //清除原有框的内容
    m_select = L"";
    UpdateData(false);
}

```

6.12 数据包捕获工作者进程

```

// 线程函数的定义
UINT Capturer(LPVOID pParm)
{
    // TODO: 在此处添加实现代码。

    CCaptureDlg* dlg = (CCaptureDlg*)theApp.m_pMainWnd; //获取对话框句柄

    char errbuff[1000];
    memset(errbuff, 0, sizeof(errbuff));

    if ((afx_adhandle = pcap_open(dlg->m_now->name, // 设备名称
        65536, // WinPcap获取网络数据包的最大长度
        PCAP_OPENFLAG_PROMISCUOUS, // 混杂模式
        1000, // 读超时为1秒
        NULL,
        errbuff // error buffer
    )) == NULL)
    {
        AfxMessageBox(L"打开该设备网卡接口失败!", MB_OK | MB_ICONERROR);
        return -1;
    }
    if (!dlg->m_select.IsEmpty())
    {
        struct bpf_program fcode; //pcap_compile所调用的结构体
        //达式编译成能够被过滤引擎所解释的低层的字节码
        char str[20];
        memset(str, 0, sizeof(str));
        for (int i = 0; i < dlg->m_select.GetLength(); i++)
            str[i] = dlg->m_select[i];

        if (pcap_compile(afx_adhandle, &fcode, str, 1, dlg->m_this_netmask) < 0)
            AfxMessageBox(L"过滤出现问题!", MB_OK | MB_ICONERROR);
        if (pcap_setfilter(afx_adhandle, &fcode) < 0)

```

```

        AfxMessageBox(L"过滤出现问题!", MB_OK | MB_ICONERROR);
    }

    //利用pcap_next_ex函数捕获数据包
    /* 此处循环调用 pcap_next_ex来接受数据报*/
    int res;
    while (dlg->m_state && (res = pcap_next_ex(afx_adhandle,
&afx_header, &afx_pkt_data)) >= 0) {
        if (res == 0) //超时情况
            continue;
        //利用窗口的PostMessage函数发送消息
        AfxGetApp()->m_pMainWnd->PostMessage(WM_PACKET, 0, 0);
        //memset(afx_header, 0, sizeof(afx_header));
        //memset(afx_pkt_data, 0, sizeof(afx_pkt_data));
    }
    if (res == -1) //获取数据包错误
    {
        AfxGetApp()->m_pMainWnd->PostMessage(WM_PACKET, 1, 1);
        dlg->m_state = false;
    }

    return 0;
}

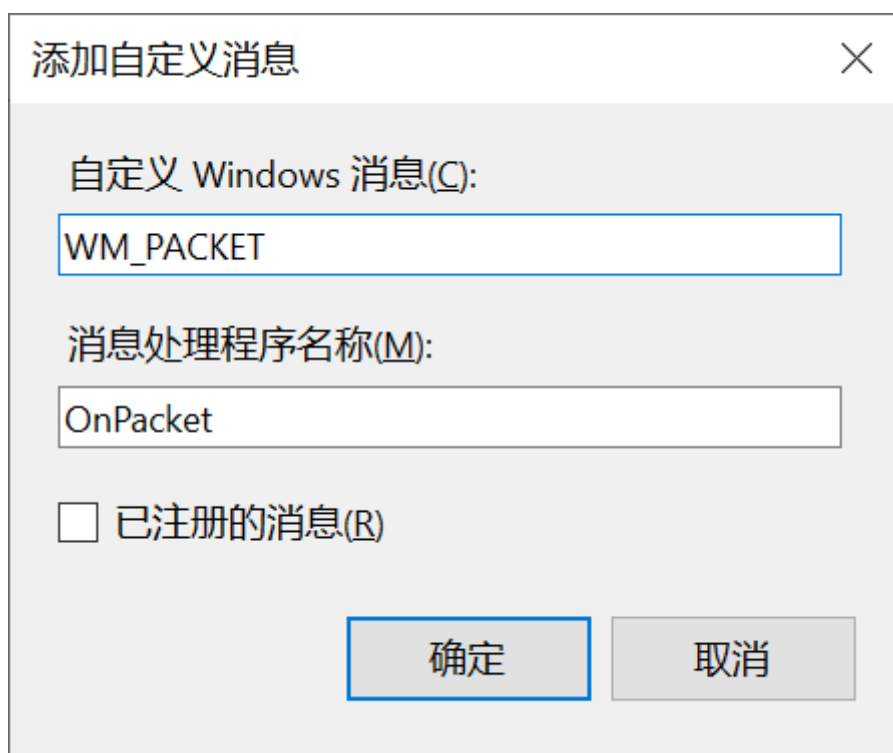
```

6.13 消息处理线程

6.13.1 自定义消息

```
#define WM_PACKET WM_USER+1
```

6.13.2 然后通过类向导创建自定义消息与处理函数。



添加自定义消息

自定义 Windows 消息(C):
WM_PACKET

消息处理程序名称(M):
OnPacket

☐ 已注册的消息(R)

确定 取消

6.13.3 编写消息处理程序

```

afx_msg LRESULT CCaptureDlg::OnPacket(WPARAM wParam, LPARAM
lParam)
{
    /*.....*/ //处理捕获到的数据包
    if (wParam == 0 && lParam == 0 && m_state == true)
    {

        //显示目的地址, 源地址, 帧类型

        Data_t* IPPacket;
        ULONG      SourceIP, DestinationIP;
        IPPacket = (Data_t*)afx_pkt_data;
        SourceIP = ntohl(IPPacket->IPHeader.SrcIP);
        DestinationIP = ntohl(IPPacket->IPHeader.DstIP);

        WORD Kind = ntohs(IPPacket->FrameHeader.FrameType);
        WORD Len = afx_header->caplen;

        USHORT modify = checksum(IPPacket->IPHeader);

        time_t time = afx_header->ts.tv_sec;
        struct tm* ltime = new struct tm;
        localtime_s(ltime, &time);
        char timestr[16];
        strftime(timestr, sizeof timestr, "%H:%M:%S", ltime);

        CString TIME(timestr);

        CString output1, output2, output3;
        //CString
        TIME=CTime::GetCurrentTime().Format("%H:%M:%S");
        output1.Format(L"%s , len: %d", TIME, Len);
        output2.Format(L"目的地址: %s      源地址: %s      帧类型:
0x%04X", char2mac(IPPacket->FrameHeader.DesMAC),
char2mac(IPPacket->FrameHeader.SrcMAC), Kind);
        output3.Format(L"标识: %04X      头部校验和: %04X      计算出
的头部校验和: %04X", ntohs(IPPacket->IPHeader.ID), ntohs(IPPacket-
>IPHeader.Checksum), modify);

        //将光标设定在最后一行
        m_list.AddString(output1);
        m_list.AddString(output2);
        m_list.AddString(output3);
        int num = m_list.GetCount();
        m_list.SetCurSel(num - 1);
    }
    else
    {
        m_list.AddString(L"获取数据包结束! ");
    }
}

```

```

    }
    m_list.AddString(L"-----
-----
-----");
    return 0;
}

```

6.14 计算校验和函数

```

USHORT checksum(IPHeader_t head)
{
    head.Checksum = 0;
    ULONG a = u_int8_t(head.SrcIP);
    ULONG b = head.DstIP;
    int size = head.TotalLen;
    USHORT hlen = head.Ver_HLen << 8;
    USHORT tos = head.TOS;
    USHORT tol = ntohs(head.TotalLen);
    USHORT id = ntohs(head.ID);
    USHORT seg = ntohs(head.Flag_Segment);
    USHORT ttl = head.TTL << 8;
    USHORT prot = head.Protocol;
    USHORT cks = head.Checksum;
    USHORT src[2];
    src[0] = ntohs(head.SrcIP >> 16);
    src[1] = ntohs(USHORT(head.SrcIP));
    USHORT dst[2];
    dst[0] = ntohs(head.DstIP >> 16);
    dst[1] = ntohs(USHORT(head.DstIP));
    USHORT buffer[12] = {
hlen, tos, tol, id, seg, ttl, prot, cks, src[0], src[1], dst[0], dst[1] };

    int i = 0; cksum = 0;
    while (i < 12)
    {
        cksum += buffer[i];
        i++;
    }

    cksum = (cksum >> 16) + (cksum & 0xffff);
    return ~(USHORT(cksum));
}

```

6.15 格式控制函数

6.15.1 将char*类型的MAC地址转换成字符串类型

```

CString CCaptureDlg::char2mac(BYTE* MAC)
{
    // TODO: 在此处添加实现代码.
    CString ans;
    ans.Format(L"%02X-%02X-%02X-%02X-%02X-%02X", int(MAC[0]),
int(MAC[1]), int(MAC[2]), int(MAC[3]), int(MAC[4]), int(MAC[5]));
    return ans;
}

```

6.15.2 将数字类型的IP地址转化为字符串类型

```

CString CCaptureDlg::long2ip(DWORD in)
{
    // TODO: 在此处添加实现代码.
    DWORD mask[] = { 0xFF000000, 0x00FF0000, 0x0000FF00, 0x000000FF
};
    DWORD num[4];

    num[0] = in & mask[0];
    num[0] = num[0] >> 24;

    num[1] = in & mask[1];
    num[1] = num[1] >> 16;

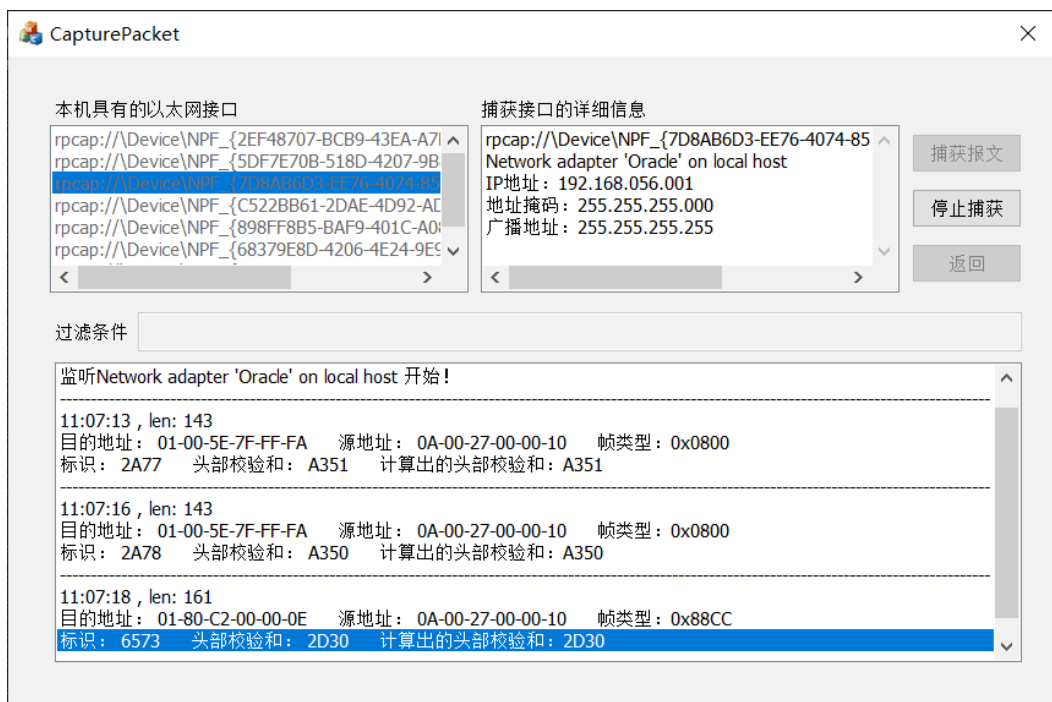
    num[2] = in & mask[2];
    num[2] = num[2] >> 8;

    num[3] = in & mask[3];

    CString ans;
    ans.Format(L"%03d.%03d.%03d.%03d", num[0], num[1], num[2],
num[3]);
    return ans;
}

```

7 效果演示



7.1 接口区

此部分显示当前运行机器上具有的网卡接口列表，并且可以已通过单击选中接口，并且在右侧的“捕获接口的详细信息”区域显示当前选中的接口的详细信息。当点击“捕获报文”按钮进行捕获之后，此部分变成失效状态，即不可更改。

7.2 接口信息区

此部分会显示在“本机具有的以太网接口”区域选中的接口的详细信息，会显示网卡接口的描述信息、IP 地址信息、地址掩码、广播地址。并且会随着接口区选定的接口的改变而改变。

7.3 按钮区

此部分具有“捕获报文”、“停止捕获”、“返回”三个按钮。负责捕获数据报和停止返回的功能。其中当点击“捕获报文”后就不可以更改网卡接口，并且会将经由该网卡接口的所有数据报逐条捕获显示特定信息在屏幕上。当点击“停止捕获”按钮后，会停止捕获选定网卡接口的数据报。当点击“返回”按钮后，会清空在“截获数据报”区域的数据报信息，并且此时可以更改接口列表。

7.4 过滤区

此部分可以输入一个大于等于 0 的整数，默认是 0 的话就会显示所有捕获选定网卡接口的数据报，如果是其他整数的话就会只显示该帧类型的数据报截获信息。

7.5 数据报区

此部分会逐条显示捕获数据报，会显示时间、帧长度、目的地址、源地址、帧类型、标识、头部校验和、计算出的头部校验和。

8 问题反思

8.1 传输方式

系统默认为广播模式（即目的mac是16f），虽然代码中我们通知他转为混杂模式，但是实际测试的时候发现，如果连接以太网，因为他直接和网线连接，所以仍以广播方式传输。

8.2 MAC地址补0操作

如果读取的mac地址只有一位，必须要设置补0操作，否则接下来获取的地址将面临错序的危机。

8.3 输出顺序不对

在控件属性里把排序置为false

8.4 类型转化

在进行计算时，注意把类型统一。

9 知识补充

什么是WinPcap做不到的

WinPcap能 独立地 通过主机协议发送和接受数据，如同TCP-IP。这就意味着WinPcap不能阻止、过滤或操纵同一机器上的其他应用程序的通讯： 它仅仅能简单地"监视"在网络上传输的数据包。所以，它不能提供类似网络流量控制、服务质量调度和个人防火墙之类的支持。