IP 数据包捕获和分析实验报告

李伟　1711350　2017 级计算机一班

摘要

在对网络的安全性和可靠性进行分析的时候，网络管理员通常需要对网络中传输的数据包进行监听和分析，目前 Internet 中流行的数据包监听和分析工具有很多，本实验基于 WinPcap 编址一个简单的 IP 数据包捕获与分析的程序，学习 IP 数据报校验和计算方法，初步掌握网络监听和分析技术的实现，加深对网络协议的理解。

关键字： IP 数据报　Winpcap　数据报校验和　网络编程　数据报捕获

目录

一、实验环境 **3**

二、程序设计思路 **3**

2.1 设备列表获取 3

2.2 打开网络接口 4

2.3 在打开的网络接口上进行数据报捕获 8

2.4 前端界面与交互设计 9

三、数据报分析与校验和计算 **10**

3.1 帧头部与数据报文解析 11

3.2 IP 头部校验和计算 13

四、程序运行效果 **14**

五、总结与思考 **16**

一、 实验环境

本实验的目的是捕获以太网的数据包并对其进行分析，因此，以太网在本实验中不可缺少，本实验中使用的以太网可以是共享式以太网也可以是交换式以太网。共享式以太网采用广播方式在共享的数据通道中传播数据，因此能够捕获共享网段中的所有流量，交换式的以太网中由于交换机采用通信过滤技术，因此只能够捕获交换机发送过来的数据包。

本实验中基于 Winpcap 捕获数据包，可视化界面使用 VS 2015 版本 MFC 搭建，实验操作系统环境为 windows 10 专业版。

二、 程序设计思路

本实验主要包括数据包捕获和分析、前端界面交互两个主要模块，前者依靠 winpcap 进行数据包捕获，后者基于 MFC 的可视化界面制作框架完成。使用 winpcap 进行数据包捕获一般需要三个主要步骤，包括获取设备列表，打开网络接口，进行数据包捕获。

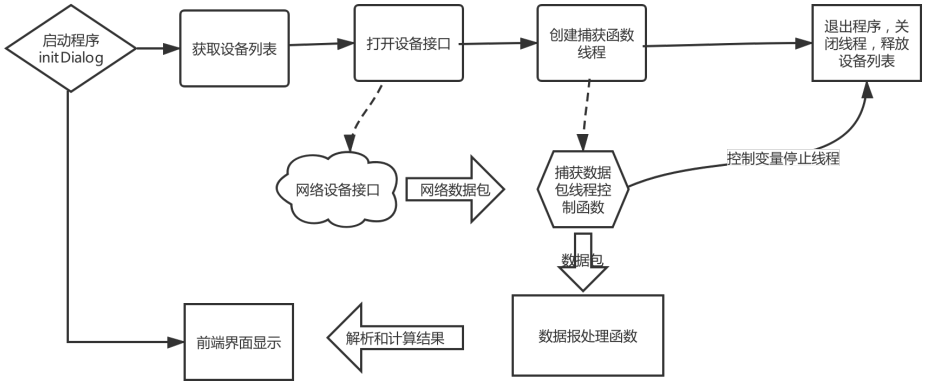

图 1　程序设计整体结构

本实验中网络数据包捕获和分析程序创建一个数据报捕获线程，利用这个线程在打开的网络设备接口上捕获该网络设备接口接收到的所有网络数据报，然后通过发送一个自定义的消息触发数据报分析函数，对收到的数据报进行解析和校验和的计算，收到的数据报通过一个缓冲队列传递给分析函数进行逐一分析。

2.1 设备列表获取

首先利用 winpcap 中的 pcap_findalldevs_ex 函数获取本机的网络接口设备列表。将本机网络列表存储在 alldevs 指针变量中。具体实现代码如下：

```
 /*
 * 获取当前主机的设备列表，基于pcap_findalldevs_ex()函数
 * 将返回的设备列表指针赋给类类中定义的变量alldevs
 */
 int flag = pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL,
    &alldevs, errbuf);//获取设备列表存储在变量
    中alldevs
 if (flag == -1)            //错误处理
 {
   MessageBox(CString(errbuf), L"ERROR", MB_OKCANCEL |
      MB_ICONERROR);
 }
 else
 {
   //显示设备接口列表
   for (d = alldevs; d != NULL; d = d->next)
   {
     EtherNet_interface_ctrl.InsertString(0, (CString)d->name);
       //为显示列表设备信息的控件控制变
         量EhterNet_interface_ctrllistBox负责显示设备列表,
   }
 }
```

在退出程序时候会调用 pcap_freealldevs 函数释放设备列表并停止正在运行的数据报捕获线程（通过对话框类的公有变量 stop_thread 变量设置），代码如下：

```
//点解返回按钮触发本函数，释放设备列表后退出程序
void CPacket_CaptureDlg::OnBnClickedCancel()
{
 // TODO: 在此添加控件通知处理程序代码

 //释放设备列表
 pcap_freealldevs(alldevs);
 //如果线程未关闭则关闭线程
 stop_thread = 1;       //重置参数为1
 CDialogEx::OnCancel();
```

```
}
```

2.2 打开网络接口

在获得网络设备接口之后，可以通过双击显示的网络设备接口，可以在其右侧的网络设备接口详细信息显示栏中显示该网络接口的具体信息。之后点击捕获数据报按钮会根据选择的网络设备接口信息，打开相应的网络设备接口，主要使用函数为pcap_open 函数，通过调用该函数打开网络设备接口并返回指向该接口的一个 pcap_t 类型指针，用于后续的数据报捕获工作。打开数据报的代码详细说明如下：

```
//点击捕获数据包按钮触发本函数
//从设备详细信息窗口获取名字信息然后打开相应网络设备接口
//创建数据包捕获线程
void CPacket_CaptureDlg::OnBnClickedButton1()
{
   // TODO: 在此添加控件通知处理程序代码
   if (interface_detail_ctrl.GetCount() == 0)    //未选择设备接口，退出函数，进行错误提示
   {
      MessageBox(L"未选择设备接口，请双击设备接口列表选项进行选择！",L"WARNING", MB_OKCANCEL | MB_ICONWARNING);
      return;
   }
   //清空之前数据包信息栏中的信息
   while (packet_list_ctrl.GetCount())
      packet_list_ctrl.DeleteString(0);
   //打开设备接口
   opened_pcap = pcap_open(d->name,    //设备接口名，直接从类成员指针中获取，一旦选择之后，指针指向选择的设备接口dd
                  4096,        //获取数据包的最大长度
                  PCAP_OPENFLAG_PROMISCUOUS, //打开设备接口获取网络数据包方式，改参数为混杂模式，获取所有流经该网络接口的数据包
                  1000,        //等待一个数据包的最大时间
                  NULL,        //远程设备捕获网络数据包使用，本实验只需设置为NULL
                  errbuf);     //错误信息缓冲区
   if (opened_pcap == NULL)  //错误处理
      MessageBox(CString(errbuf),L"ERROR", MB_OKCANCEL |
```

```
        MB_ICONERROR);
    else {
        //设备打开成功，创建线程进行数据包捕获返回一个线程指针，
        m_capture = AfxBeginThread(Capturer, //工作者线程的控制函数
                                   NULL,  //传给控制函数的参数，一般为某数据结构的指针，这里为空
                                   THREAD_PRIORITY_NORMAL); //线程优先级，默认为正常的优先级
        //成功创建线程之后将按钮设置为不可点击状态
        GetDlgItem(IDC_BUTTON1)->EnableWindow(false);
        GetDlgItem(IDC_BUTTON2)->EnableWindow(true);
        //输出日志信息
        packet_list_ctrl.InsertString(packet_list_ctrl.GetCount(), L"监听 " + (CString)d->description);
        packet_list_ctrl.InsertString(packet_list_ctrl.GetCount(),
            L"_____\
_____\
_____");
        //MessageBox(CString(d->name));
    }
}


// 处理捕获数据包的函数对数据包进行解析，
LRESULT CPacket_CaptureDlg::OnPacket(WPARAM wParam, LPARAM lParam)
{
    TRY{
    //从缓冲队列中提取数据报文
    pcap_pkthdr* pkt_header = pkthdr_list.GetHead();  //记录捕获的数据包头
    const u_char* pkt_data = pktdata_list.GetHead();   //记录捕获的数据包
    //获取帧头部保存的时间戳数据和类型数据
    CString header_s,data_s;
    char timebuf[128];
    time_t t(pkt_header->ts.tv_sec);
    strftime(timebuf, 64, "%Y-%m-%d %H:%M:%S", localtime(&t));
```

```
54    //数据格式化
55    header_s.Format(L"%s.%d,len:%d",CString(timebuf),\
56                    pkt_header->ts.tv_usec,pkt_header->caplen);
57    FrameHeader_t* FHeader = (FrameHeader_t*)pkt_data;
58    //过滤协议报文
59    if(ntohs(FHeader->FrameType) == WORD(0x0800))
          //ntohs(FHeader->FrameType) == WORD(0x0800)
60    {
61       Data_t* Data = (Data_t*)pkt_data;
62       //显示时间数据报头部信息
63       packet_list_ctrl.InsertString(packet_list_ctrl.GetCount(),
             header_s);
64       //数据格式化
65       data_s.Format(L"标识:0x%04x 头部校验和     :0x%04x 计算出的头部校验
             和        :0x%04x", ntohs(Data->IPHeader.ID),
             ntohs(Data->IPHeader.Checksum),
             IPHeader_ckeck((WORD*)(&Data->IPHeader)));
66       //数据阵头部校验信息等
67       packet_list_ctrl.InsertString(packet_list_ctrl.GetCount(),
             data_s);
68       packet_list_ctrl.InsertString(packet_list_ctrl.GetCount(),
             L"_____\
69    _____\
70    _____");
71       /*if (ntohs(Data->IPHeader.Checksum) !=
             IPHeader_ckeck((WORD*)(&Data->IPHeader)))
72       {
73          packet_list_ctrl.InsertString(packet_list_ctrl.GetCount(),
               L数据报错
               误"###################!#########################");
74       }*/
75    }
76
77    pkthdr_list.RemoveHead();              //移除头部数据包
78    pktdata_list.RemoveHead();
79
```

```
80    }CATCH(CException, e)
81    {
82       TCHAR error[1024](L"\0");
83       e->GetErrorMessage(error,1024);
84       packet_list_ctrl.InsertString(packet_list_ctrl.GetCount(), L"处
            理异常$$$$$"+(CString)error);
85       memset(error, 0, 1024);
86    }END_CATCH;
87    //MessageBox(L这里是分析函数！"");
88    return LRESULT();
89 }
```

2.3 在打开的网络接口上进行数据报捕获

在 2.2 节的打开网络设备接口的基础上进行数据包的捕获工作，具体的捕获数据包功能实现基于 pcap_next_ex 函数进行，相关的工作在数据包捕获线程的控制函数 Capture() 中完成，具体代码实现如下：

```
1  //线程执行函数，进行数据包的捕获工作
2  UINT Capturer(PVOID hwnd) {
3     pcap_pkthdr* pkt_header = NULL;  //记录捕获的数据包头
4     const u_char* pkt_data = NULL;   //记录捕获的数据包
5     CPacket_CaptureDlg* Dlg =
         (CPacket_CaptureDlg*)AfxGetApp()->m_pMainWnd; //获取主窗口句
         柄
6     while (!Dlg->stop_thread)   //循环捕获网络数据包，通过参
         数stop_thread 控制停止捕获
7     {
8        //捕获数据包的主体代码，利用函数执行pcap_next_ex
9        int flag = 0;
10       TRY{
11          flag = pcap_next_ex(Dlg->opened_pcap, //指针，说明捕获那个网卡
               上的数据包pcap_t
12             &pkt_header,            //pcap_pkthdr 指针，记录数据包相关信
                  息
13             &pkt_data               //uchar指针，保存数据包数据*
14             );
```

```
    }CATCH(CException, e)
    {
      TCHAR error[1024](L"\0");
      e->GetErrorMessage(error, 1024);
      Dlg->packet_list_ctrl.InsertString(Dlg->packet_list_ctrl.GetCount(),\
            L"捕获异常$$$$$"+(CString)error);
      memset(error, 0, 1024);
      continue;
    }END_CATCH;
    //成功捕获数据并对捕获的函数进行分类,
    if (flag == 1)
    {
      Dlg->pkthdr_list.AddTail(pkt_header);
      Dlg->pktdata_list.AddTail(pkt_data);
      Dlg->PostMessageW(WM_PACKET, 0, 0);
    }
    else if (flag == 0)
    {
      //Dlg->MessageBox(L未在规定时间内捕获数据包"");
    }
    else if (flag == -1)
    {
      Dlg->MessageBox(L"数据报捕获函数执行错误! ");
    }
  }
  Dlg->stop_thread = 0;      //重置参数为0
  Dlg->packet_list_ctrl.InsertString(Dlg->packet_list_ctrl.GetCount(),
    L"捕获数据包线程停止!
    ");
  Dlg->packet_list_ctrl.InsertString(Dlg->packet_list_ctrl.GetCount(),
    L"_____________________________________\
_____________________________________________\
_____________________________________");;  //提示停止捕获数据包
  return 1;
}
```

数据包捕获操作通过一个独立的线程进行，其控制函数为 Capture 函数，这一函数负责通过不断的循环调用 pcap_next_ex 函数捕获数据包，如果捕获成功则通过自定义的 WM_PACKET 消息通知主线程的分析函数分析捕获的数据报，同时将捕获的数据报数据存入一个缓冲队列中留待主线程分析处理函数进行处理。同时，当控制变量 stop_thread 值为 true 的时候，线程将会退出执行，从而实现关闭捕获数据包线程的功能。

2.4 前端界面与交互设计

前端界面设计主要分为信息展示区、操作区、数据报捕获和日志显示区三个主要模块。具体页面效果如下：

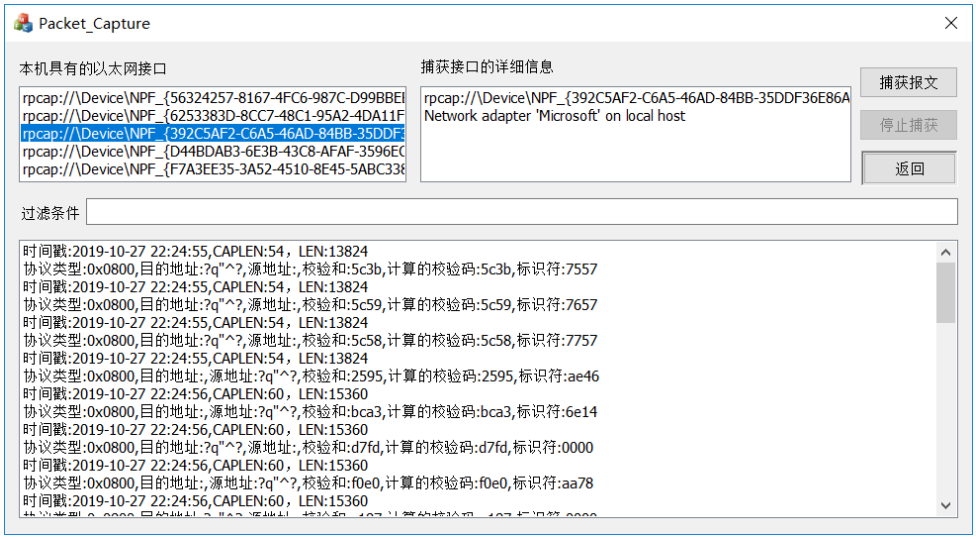

图 2　界面显示效果

当用户启动程序之后，将会扫描本机并显示设备列表，双击列表项之后，设备详细信息信息将会显示在其右侧的接口详细信息显示窗口中，点击捕获报文按钮之后将会打开设备接口并创建线程捕获数据包，同时锁定捕获报文按钮，激活终止捕获按钮。在下方的信息显示框中显示捕获的数据报文分析结果以及日志信息。

三、 数据报分析与校验和计算

本实验在捕获数据报之后需要对捕获的数据报文进行分析，筛选 IP 数据报并进行头部校验和的计算，在本程序中，对于捕获数据包的分析定义了三个数据结构，分别是 FrameHeader、IPHeader_t、Data_t，这三个数据结构分别对应于帧结构头部、IP 头部、以及包含帧头部和 IP 数据报得数据报文结构，通过将获取的 pkt_data 指针分别转为这三种类型便能够对数据包进行解析和处理。三种结构定义如下所示：

```
//帧数据，数据包数据结构定义IP
#pragma pack(1)
typedef struct FrameHeader_t{  //帧首部
   BYTE DesMAC[6];          //目的地址
   BYTE SrcMAC[6];          //原地址
   WORD FrameType;          //帧类型
}FrameHeader_t;

typedef struct IPHeader_t {
   BYTE Ver_HLen;
   BYTE TOS;
   WORD TotalLen;
   WORD ID;
   WORD Flag_Segment;
   BYTE TLL;
   BYTE Protocol;
   WORD Checksum;
   ULONG SrcIP;
   ULONG DstIP;
}IPHeader_t;

typedef struct Data_t {
   FrameHeader_t FrameHeader;
   IPHeader_t IPHeader;
}Data_t;

#pragma pack()
```

3.1 帧头部与数据报文解析

通过定义的数据结构对数据报文进行解析，输出数据包的帧长度与数据包接收时间等信息，同时通过读取帧头部中的 frametype 数据判断协议类型，通过是否为 0x0800 判断捕获的数据报文是否为 IP 协议报文，同缓冲队列中读取 pkt_header 指针中获取时间戳和数据包长度的信息并显示, 具体代码如下：

```
// 处理捕获数据包的函数对数据包进行解析,
LRESULT CPacket_CaptureDlg::OnPacket(WPARAM wParam, LPARAM lParam)
{
   TRY{
   //从缓冲队列中提取数据报文
   pcap_pkthdr* pkt_header = pkthdr_list.GetHead();  //记录捕获的数据包头
   const u_char* pkt_data = pktdata_list.GetHead();   //记录捕获的数据包
   //获取帧头部保存的时间戳数据和类型数据
   CString header_s,data_s;
   char timebuf[128];
   time_t t(pkt_header->ts.tv_sec);
   strftime(timebuf, 64, "%Y-%m-%d %H:%M:%S", localtime(&t));
   //数据格式化
   header_s.Format(L"%s.%d,len:%d",CString(timebuf),\
       pkt_header->ts.tv_usec,pkt_header->caplen);
   FrameHeader_t* FHeader = (FrameHeader_t*)pkt_data;
   //过滤协议报文
   if(ntohs(FHeader->FrameType) == WORD(0x0800))
      //ntohs(FHeader->FrameType) == WORD(0x0800)
   {
      Data_t* Data = (Data_t*)pkt_data;
      //显示时间数据报头部信息
      packet_list_ctrl.InsertString(packet_list_ctrl.GetCount(),
         header_s);
      //数据格式化
      data_s.Format(L"标识:0x%04x 头部校验和     :0x%04x 计算出的头部校验和        :0x%04x", ntohs(Data->IPHeader.ID),
         ntohs(Data->IPHeader.Checksum),
         IPHeader_ckeck((WORD*)(&Data->IPHeader)));
      //数据阵头部校验信息等
      packet_list_ctrl.InsertString(packet_list_ctrl.GetCount(),
         data_s);
      packet_list_ctrl.InsertString(packet_list_ctrl.GetCount(),
         L"_____\
```

```
_____\
_____");

    }

    pkthdr_list.RemoveHead();                //移除头部数据包
    pktdata_list.RemoveHead();

    }CATCH(CException, e)
    {
        TCHAR error[1024](L"\0");
        e->GetErrorMessage(error,1024);
        packet_list_ctrl.InsertString(packet_list_ctrl.GetCount(), L"处
            理异常$$$$$"+(CString)error);
        memset(error, 0, 1024);
    }END_CATCH;
    //MessageBox(L这里是分析函数！"");
    return LRESULT();
}
```

3.2 IP 头部校验和计算

校验和通过 IP 头部反码求和的操作实现数据的差错检验功能，在接收到数据报进行处理的过程中调用函数 IPHeader_check() 函数计算校验和并将计算的校验和显示出来和数据报中的校验和进行比较。具体计算的函数代码实现如下：

```
// 头部检验算法IP
WORD CPacket_CaptureDlg::IPHeader_ckeck(WORD* IPHeader)
{
    WORD ans = 0;
    CString str;
    for (int i = 0; i < 10; i++)
    {
        if (i == 5) continue;    //跳过检验和字
        if ((ans + IPHeader[i])%0x10000 < ans)
            ans = (ans + IPHeader[i]) % 0x10000 + 1;
```

```
    else
        ans += IPHeader[i];
  }
  return WORD(ntohs(~ans)); //求反码，并转为格式word
}
```

四、 程序运行效果

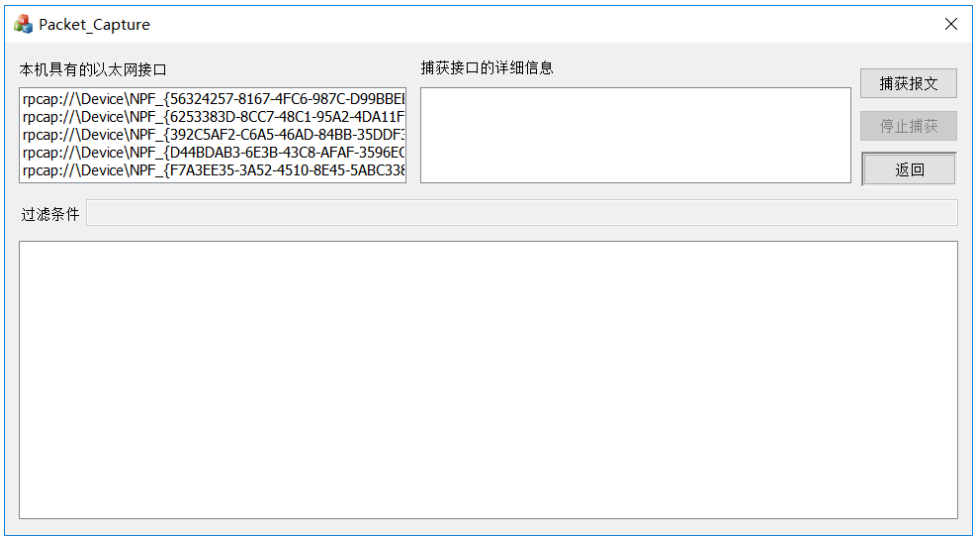

图 3　程序启动界面

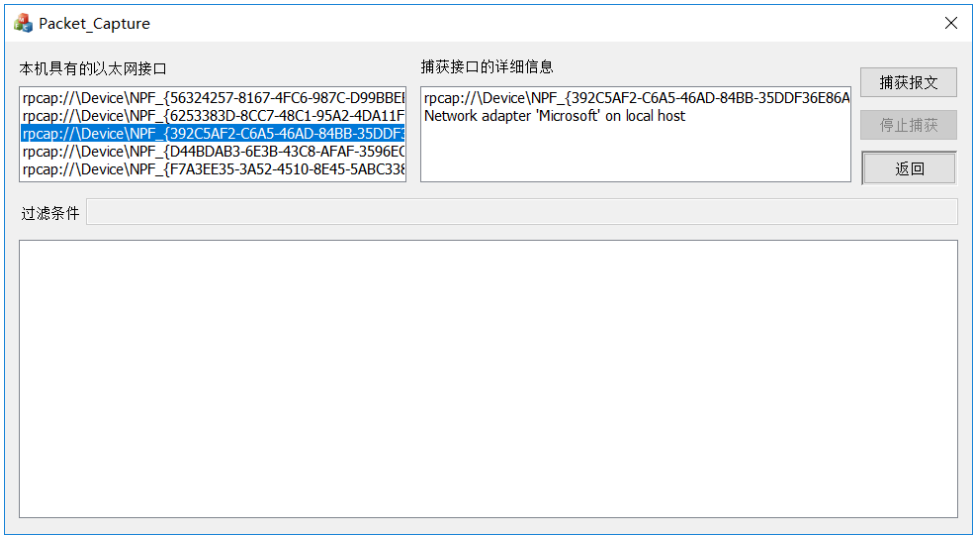

图 4　双击设备项显示设备接口详情

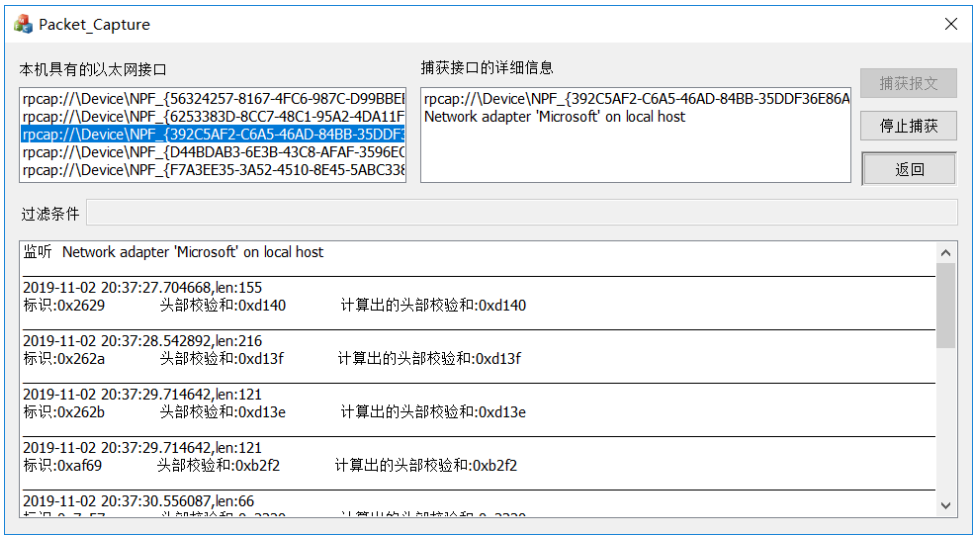

图 5　捕获数据

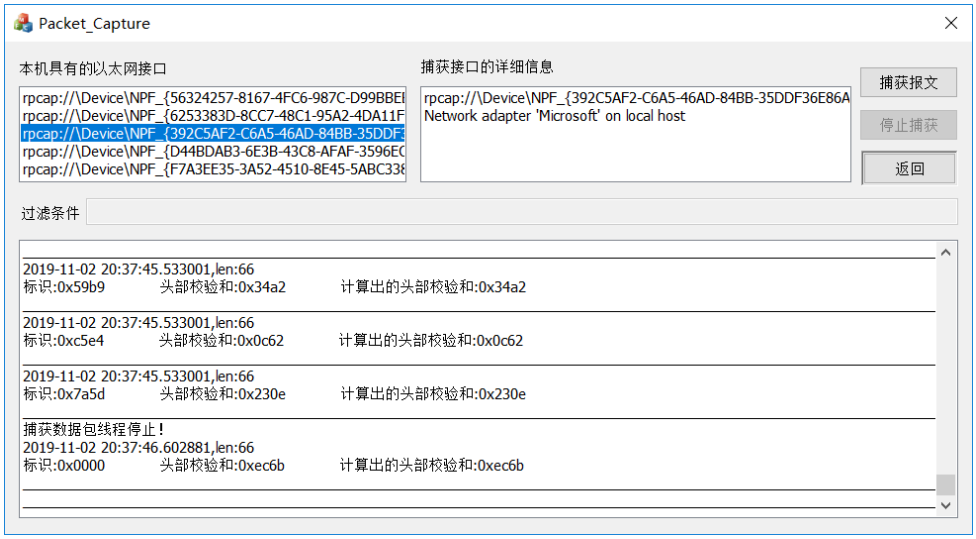

图 6　停止捕获

如果计算校验和时不将校验和位置为 0，则计算出来的校验和为全 0 则表明传输过程中应该没有差错，实现的效果如下图所示。

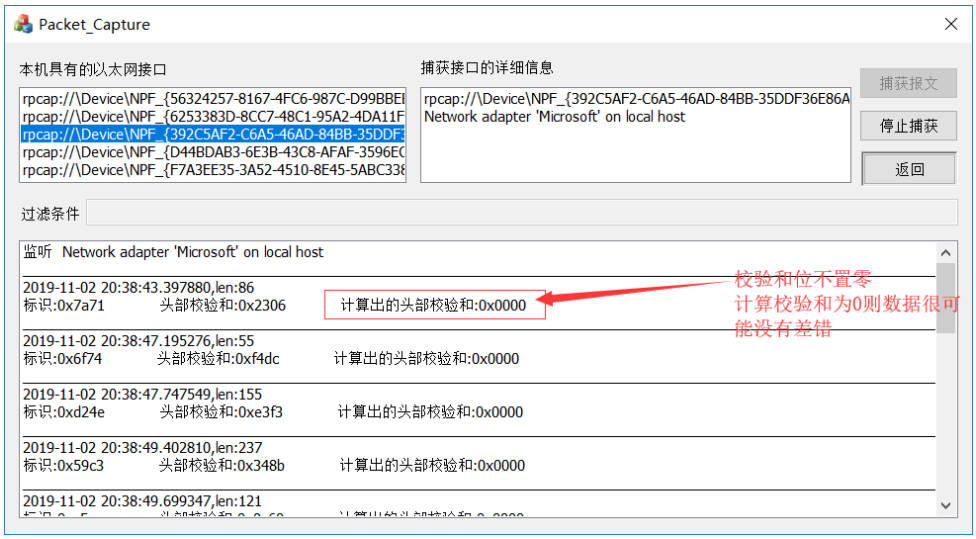

图 7　校验和位不置零执行效果显示

五、 总结与思考

通过这次的数据包捕获实验让我更深刻的理解了数据包传输和接收的过程以及 IP 协议的具体实现，同时通过直接操作 IP 数据报的解析过程，更加深刻的认识了 IP 数据包的格式设计和相关功能设计的思想，同时通过这一次的编程操作，使得我对于网络编程更加的熟练。

在这一次的实验过程中，遇到了一些问题，但是也做出了一些探索和思考，比如在计算校验和的时候，是否将校验和位清零，产生的不同结果让我对于 IP 协议的差错检验方式有了更清楚的认识。总而言之，通过这一次的实验，我感到自己在网络编程能力和网络通信的理论知识上都有了提升。