Sniffer 大作业项目文档

费扬 519021910917

朱文骏 519021911292

本次项目的内容在如下的github仓库中：

2020dfff/Sniffer_IS301: Project on sniffer, course design(IS301). (github.com)

项目分工：
朱文骏：完成了sniffer类的设计与实现，实现了包的过滤和查找功能，实现了保存包的功能，实现了IP分组重组。
费扬：完成了页面的设计与实现，实现了对于data_info_table的监听更新功能，对软件进行了测试并撰写了实验报告。

一. 概述

此次大作业项目实现了一个网络嗅探器Sniffer，其主要功能有：对用户选定的本机网卡抓包并解析、实现包过滤、查找、IP包分片重组、文件的保存等。此外，本项目实现了一个用户友好的GUI，基于linux系统和Python3环境，使用scapy库进行抓包、解析、过滤，使用PyQt5库实现GUI。

开发背景说明：

> **Requirements:**

环境/库	版本及说明
Python3	3.6.9及以上
Scapy	2.4.5
PyQt5	5.15.4
PyQt5-Qt5	5.15.2
PyQt5-sip	12.9.0

文件列表：

文件夹	文件名	说明
/src	README.txt	项目文档
	data_info_list.py	定义数据信息列表
	main.py	主程序入口
	main_window.py	主窗口UI
	sniffer.py	Sniffer类，嗅探线程
	function.py	定义各类函数使用
/testfile	test.py	测试UI文件
	scapy-test.py	scapy函数调用测试
	test.pcap	测试保存文件
	result.out	\

二. 内容结构：

UI页面：

> **1. Activate：**

```
#user@username:
../Sniffer_IS301/src$ sudo python3 main.py
```

此时，GUI启动，窗口初始化，载入应用程序，稍等即可进入主界面MainWindow；

> **2. MainWindow：**

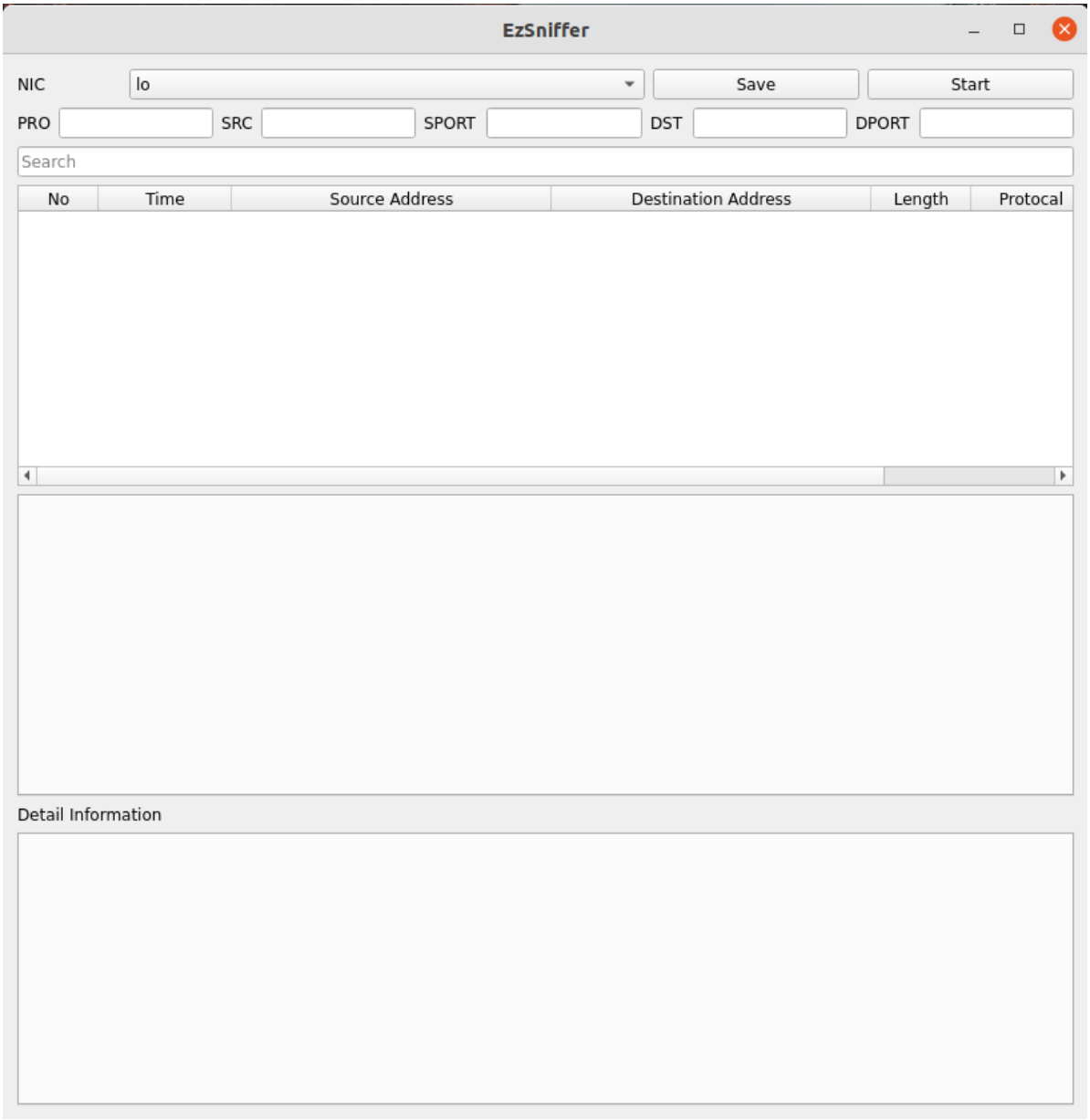

图1：初始主界面

主界面按照GridLayout铺开，默认布局(从上至下从左到右)依次为：

> **网卡选择下拉框**
>
> **保存和开始按钮**
>
> **协议，源和目的地址，端口筛选文本框**
>
> **可输入文本搜索框**
>
> **QtableWidget按照抓包的时间顺序依次显示报文记录**
>
> **QtextBrowser文本内容显示框，显示报文解析后的具体信息**
>
> **详细信息，包括GB2312和UTF-8解码信息**

在选择虚拟网卡时，会有下拉框提示：

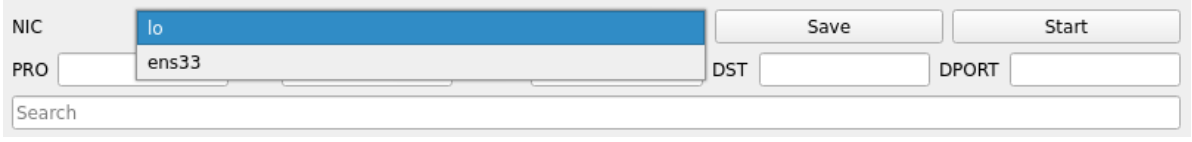

图2：虚拟网卡选择

输入要解析的数据报文类型，如ICMP。

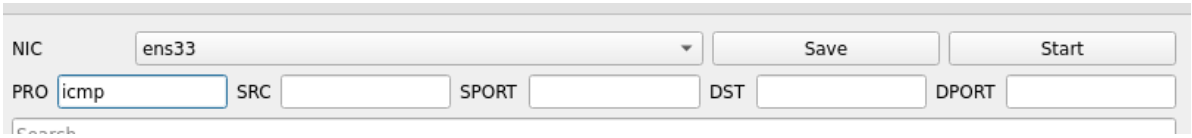

图3：筛选解析的报文类型

3. Start:

单击Start按钮开始对选定网卡抓包并解析，此时按钮变为Stop，再次单击可停止本次抓包。

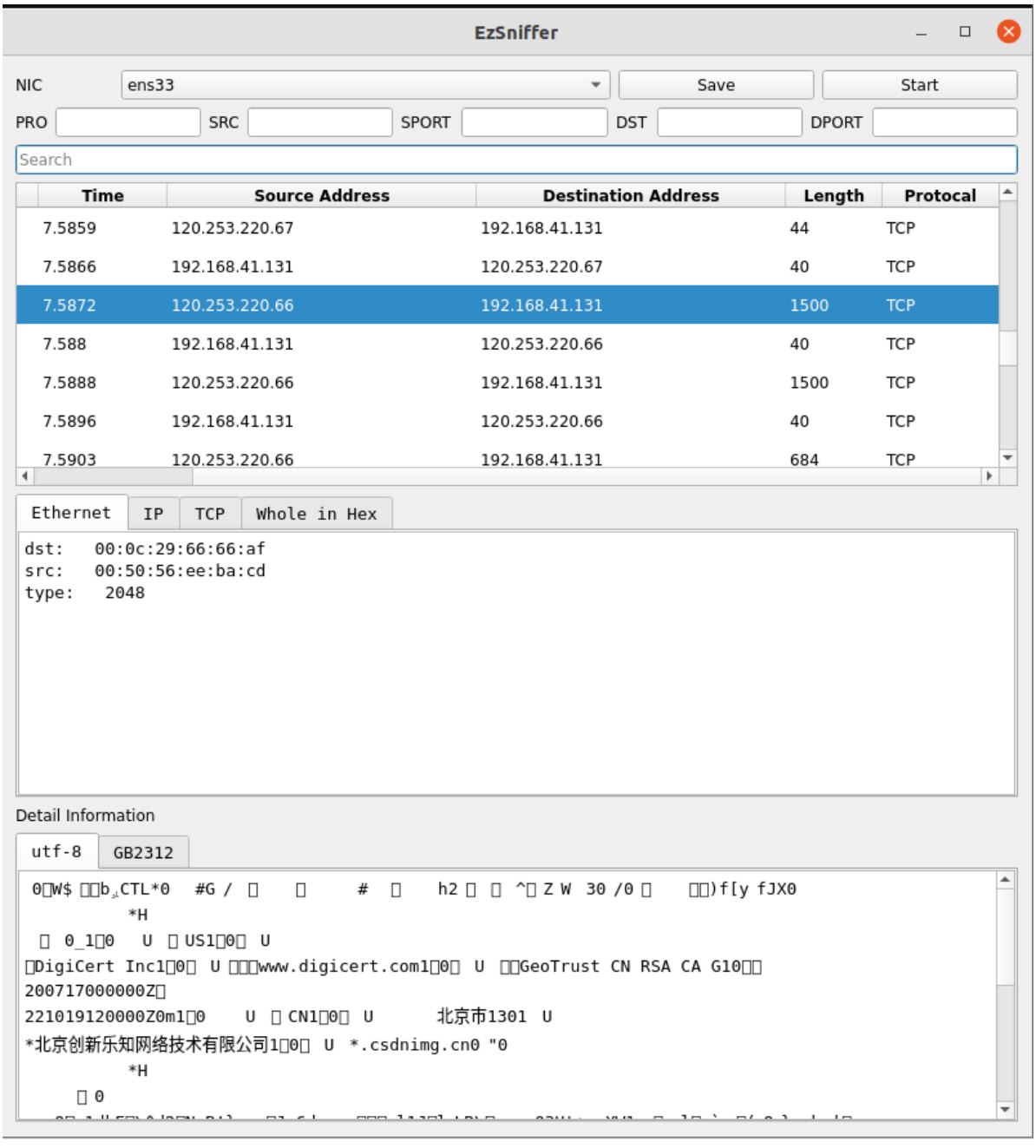

图3：对ens33网卡抓包

4. 查看信息:

在图3的界面中，使用TabWidget按标签排列选中的数据报文的各种类别信息，这里不同类型的报文所打开的TabWidget标签不同。

> **5. Detail Information:**

在图3中，可以通过两种编码方式，在QTextBrowser区域中查看详细信息，做到一定的信息可读性。

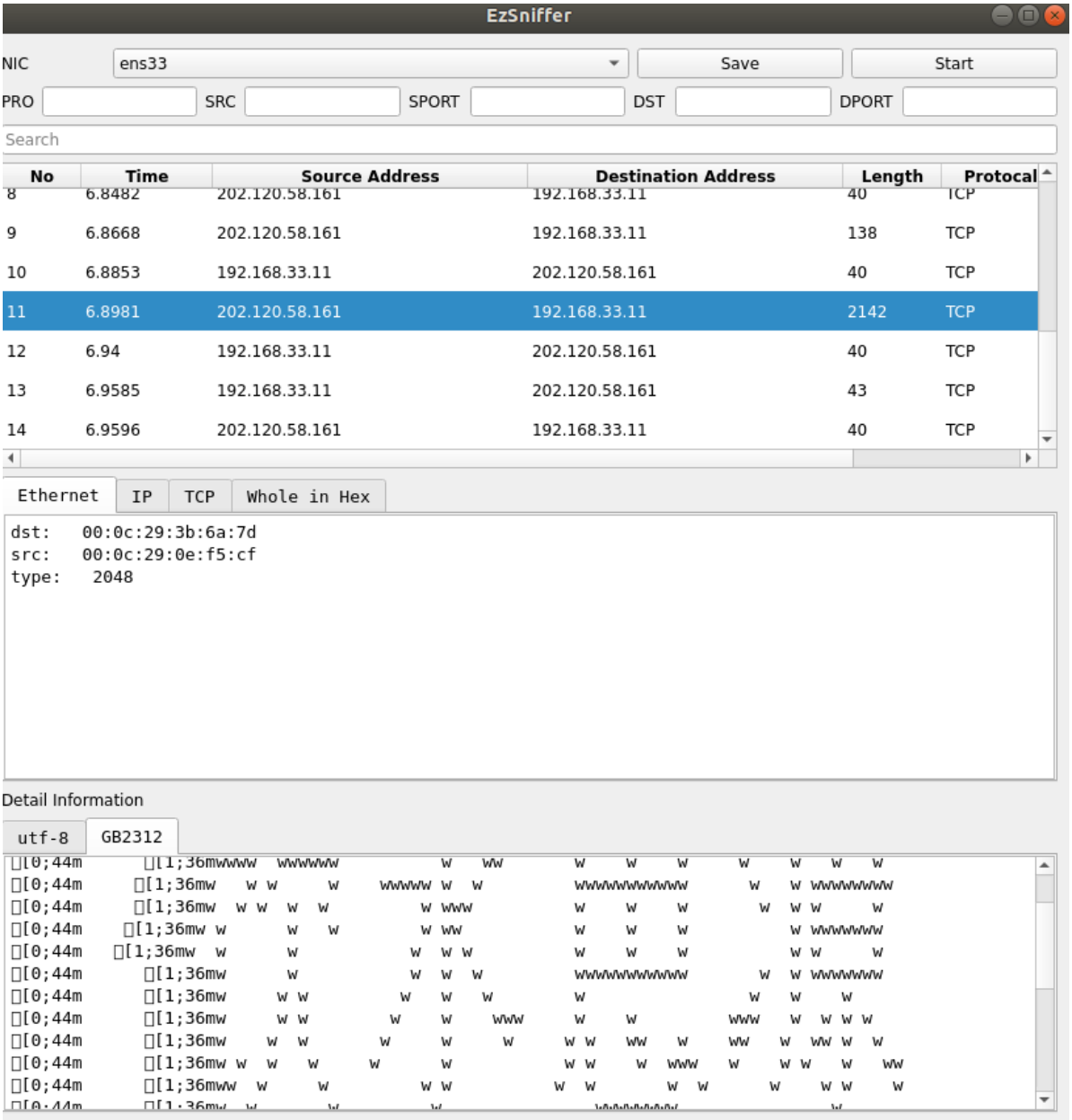

图4：可读的bbs网站内容

算法结构：

data_info_list.py

用于存储的list对应的内容：

list名	作用
src_list	sort ip address
src_port_list	source port of packets
dst_list	destination ip address
dst_port_list	destination port of packets
pro_list	protocal number(which can be converted to protocal name by dict_pro) 1(which is ICMP)
len_list	length of a packet
layer_list	a list of protocal which a packet includes ['Ethernet', 'IP', 'TCP']
field_list	a list of property of each layer
packet_list	a list of all packet received
detail_info_utf8_list	information decoded by utf8
detail_info_gb_list	information decoded by gb2312
time_list	time when received the packet
hex_list	hex of packet

methods:

methods	作用
append_xxx_list	be used to append those list above
clear	clear all list(used when switch net interface)

附录代码如下:

```
class data_info_list():

    def __init__(self):
        self.src_list = []
        self.src_port_list = []
        self.dst_list = []
        self.dst_port_list = []
        self.pro_list = []
        self.len_list = []
        self.layer_list = []
        self.field_list = []
        self.packet_list = []
        self.detail_info_utf8_list = []
        self.detail_info_gb_list = []
        self.hex_list = []
        self.time_list = []

    def append_src_list(self, src):
        self.src_list.append(src)
```

```
    def append_src_port_list(self, src_port):
        self.src_port_list.append(src_port)

    def append_dst_list(self, dst):
        self.dst_list.append(dst)

    def append_dst_port_list(self, dst_port):
        self.dst_port_list.append(dst_port)

    def append_pro_list(self, pro):
        self.pro_list.append(pro)

    def append_len_list(self, length):
        self.len_list.append(length)

    def append_layer_list(self, layer):
        self.layer_list.append(layer)

    def append_field_list(self, field):
        self.field_list.append(field)

    def append_packet_list(self, packet):
        self.packet_list.append(packet)

    def append_detail_info_utf8_list(self, detail_info):
        self.detail_info_utf8_list.append(detail_info)

    def append_detail_info_gb_list(self, detail_info):
        self.detail_info_gb_list.append(detail_info)

    def append_hex_list(self, hex_):
        self.hex_list.append(hex_)

    def append_time_list(self, time):
        self.time_list.append(time)

    def clear(self):
        self.src_list = []
        self.src_port_list = []
        self.dst_list = []
        self.dst_port_list = []
        self.pro_list = []
        self.len_list = []
        self.layer_list = []
        self.field_list = []
        self.packet_list = []
        self.detail_info_utf8_list = []
        self.detail_info_gb_list = []
        self.hex_list = []
        self.time_list = []
```

sniffer.py

class sniffer:

Property	Explanation
interface	network card name "ens33"
on_off_flag	control stop and run of sniffer
frag_set	used to store those packets which need to be reassemble
count	number of packet recieved, useless
window	class main_window

Methods	作用
begin_sniffer	just begin sniffer
is_off	control stop and run by on_off_flag (if True: stop)

重要函数功能介绍：

> **packetHandler：**

这里的packetHandler用于处理收到的数据包，首先检查frag_set，判断数据包是否可以和别的包组合，如果为真并且数据包的flag为DF，则组合并且将其加到data_info_list中，返回值为真；如果判断为真并且数据包的flag为MF，则加入frag_set，返回真；如果假则返回假。

如果check_merge返回为真，则直接跳过这个数据包，识别为已处理。如果check_merge返回为假，则检查flag是否为MF，若是，则加到frag_set中并跳过此数据包；

若数据包经过了所有上述步骤，则可以将SPORT和DPORT等信息加入data_info_list；

注意细节信息列表和细节信息列表有点复杂，这么多的尝试很复杂，这样做只是想在开始时过滤一些无效的字节，比如“0xff”，所以不需要关心如何实现这些细节，只要通过这种方式可以获得信息。

> **checkmerge：**

检查一个数据包是否可以与其他碎片数据包合并，将数据包添加到帧集将数据包添加到帧集。

附录代码如下：

```
import sys
import codecs
import time
from scapy.all import *


def to_hex(pkt):
    return hexdump(pkt)

class sniffer():

    def __init__(self):
        self.interface = "ens33"
        self.on_off_flag = 0;
        self.count = 0;
```

```
        self.frag_set = []

    def begin_sniff(self, window):
        self.window = window
        self.interface = window.combo_box_nic.currentText()
        window.start_time = time.time()
        sniff(iface = self.interface, prn = self.packetHandler, stop_filter =
self.is_off)
        print("sniffer finished.")


    def packetHandler(self, pkt):
        if(IP in pkt):
            if(self.check_merge(pkt)):
               return

            if(pkt['IP'].flags == 'MF'):
                self.add_to_frag_set(pkt)
                # print(pkt['IP'].flags)
                return
        else: return

        if(TCP in pkt):
            self.window.m_data_info_list.append_src_port_list(pkt['TCP'].sport)
            self.window.m_data_info_list.append_dst_port_list(pkt['TCP'].dport)
        elif(UDP in pkt):
            self.window.m_data_info_list.append_src_port_list(pkt['UDP'].sport)
            self.window.m_data_info_list.append_dst_port_list(pkt['UDP'].dport)
        else:
            self.window.m_data_info_list.append_src_port_list(0)
            self.window.m_data_info_list.append_dst_port_list(0)

        #print("------------------")
        # tmp = to_hex(pkt)
        tmp = str(pkt)
        tmp_time = time.time() - self.window.start_time
        tmp_time = round(tmp_time, 4)
        # print("tmp = " + tmp)
        # print(tmp)
        # tmp = 0
        #print("------------------")
        self.window.m_data_info_list.append_hex_list(tmp)
        self.window.m_data_info_list.append_time_list(tmp_time)
        flag = 0
        flag2 = 0
        if(Raw in pkt):
            try:
                tmp1 = pkt[Raw].load.decode('utf-8')
                tmp2 = pkt[Raw].load.decode('GB2312')
                flag = 1
            except:
                pass
            if(flag):
                self.window.m_data_info_list.append_detail_info_utf8_list(tmp1)
                self.window.m_data_info_list.append_detail_info_gb_list(tmp2)
            else:
                try:
                    flag2 = 1
```

```
                except:
                    pass

            if(flag2):

 self.window.m_data_info_list.append_detail_info_utf8_list(pkt[Raw].load[10:].de
code('utf-8', errors='ignore'))

 self.window.m_data_info_list.append_detail_info_gb_list(pkt[Raw].load[10:].deco
de('GB2312', errors='ignore'))
            else:
                self.window.m_data_info_list.append_detail_info_utf8_list('No
extra information')
                self.window.m_data_info_list.append_detail_info_gb_list('No
extra information')
        else:
            self.window.m_data_info_list.append_detail_info_utf8_list('No extra
information')
            self.window.m_data_info_list.append_detail_info_gb_list('No extra
information')

        self.window.m_data_info_list.append_src_list(pkt['IP'].src)
        self.window.m_data_info_list.append_dst_list(pkt['IP'].dst)
        self.window.m_data_info_list.append_pro_list(pkt['IP'].proto)
        self.window.m_data_info_list.append_len_list(pkt['IP'].len)
        self.window.m_data_info_list.append_packet_list(pkt)
        layer = []
        field = []
        while(pkt.payload):
            layer.append(pkt.name)
            field.append(pkt.fields)
            # print(pkt.name)
            # print(pkt.fields)
            # print(pkt.payload)
            pkt = pkt.payload

        self.window.m_data_info_list.append_layer_list(layer)
        self.window.m_data_info_list.append_field_list(field)
        # print(self.window.m_data_info_list.layer_list[self.count])
        # print(self.window.m_data_info_list.field_list[self.count])
        # print(self.window.m_data_info_list.detail_info_gb_list[self.count])
        # print(self.window.m_data_info_list.hex_list[self.count])
        self.count = self.count + 1

    def is_off(self, pkt):
        # print(self.on_off_flag)
        if(self.on_off_flag == 0):
            return True
        return False

    def add_to_frag_set(self, pkt):
        self.frag_set.append([pkt])
        return

    def check_merge(self, pkt):
        for i in self.frag_set:
            # print(i)
            if(i[0]['IP'].id == pkt['IP'].id):
```

```
                if(pkt['IP'].flags == 0):
                    self.window.m_data_info_list.append_src_list(pkt['IP'].src)
                    self.window.m_data_info_list.append_dst_list(pkt['IP'].dst)
                    self.window.m_data_info_list.append_pro_list(i[0]
['IP'].proto)
                    total_len = pkt['IP'].len
                    for x in i:
                        total_len = total_len + x['IP'].len
                    self.window.m_data_info_list.append_len_list(total_len)
                    i[0]['IP'].len = total_len
                    i[0]['IP'].flags = "DF"

                    layer = []
                    field = []
                    tmp_pkt = i[0]
                    while(tmp_pkt.payload):
                        layer.append(tmp_pkt.name)
                        field.append(tmp_pkt.fields)
                        # print(pkt.name)
                        # print(pkt.fields)
                        # print(pkt.payload)
                        tmp_pkt = tmp_pkt.payload
                    self.window.m_data_info_list.append_layer_list(layer)
                    self.window.m_data_info_list.append_field_list(field)
                    self.window.m_data_info_list.append_packet_list(i[0])
                else:
                    i.append(pkt)
                return True
        return False
```

---

function.py

property	作用
dict_pro	a dict used to convert protocol number to protocol number

重要函数功能介绍：

> **data_info_table_listener：**

用于填充数据信息表（仍需改进）。

它需要检测搜索条件是否已更改，这是由按ENTER键触发的。因此，latest_xxx用于保存历史条件，并与当前条件（即窗口）进行检查。check_parameter_changed是一个不太好看的函数。但它完成了它的工作，可以考虑这样写：`def check_parameter_changed(*args)`

> **check_invalid：**

只需检查数据包的属性是否包含需求（如protocal是ICMP）

> **check_msg_not_in_detail**

就像上面一样，检查详细信息列表或详细信息列表中的消息，如果不是，则返回True，否则返回False。

附录代码如下：

```
import time
```

```
import sys
import threading
from scapy.all import *
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton,QHBoxLayout,
QVBoxLayout, QGridLayout, QTableWidgetItem
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QAbstractItemView
from PyQt5.QtCore import Qt
import netifaces
import time
dict_pro = {0: 'HOPOPT', 1: 'ICMP', 2: 'IGMP', 3: 'GGP', 4: 'IP-in-IP', 5: 'ST',
6: 'TCP', 7: 'CBT', 8: 'EGP', 9: 'IGP', 10: 'BBN-RCC-MON', 11: 'NVP-II', 12:
'PUP', 13: 'ARGUS', 14: 'EMCON', 15: 'XNET', 16: 'CHAOS', 17: 'UDP', 18: 'MUX',
19: 'DCN-MEAS', 20: 'HMP', 21: 'PRM', 22: 'XNS-IDP', 23: 'TRUNK-1', 24: 'TRUNK-
2', 25: 'LEAF-1', 26: 'LEAF-2', 27: 'RDP', 28: 'IRTP', 29: 'ISO-TP4', 30:
'NETBLT', 31: 'MFE-NSP', 32: 'MERIT-INP', 33: 'DCCP', 34: '3PC', 35: 'IDPR', 36:
'XTP', 37: 'DDP', 38: 'IDPR-CMTP', 39: 'TP++', 40: 'IL', 41: 'IPv6', 42: 'SDRP',
43: 'IPv6-Route', 44: 'IPv6-Frag', 45: 'IDRP', 46: 'RSVP', 47: 'GREs', 48:
'DSR', 49: 'BNA', 50: 'ESP', 51: 'AH', 52: 'I-NLSP', 53: 'SWIPE', 54: 'NARP',
55: 'MOBILE', 56: 'TLSP', 57: 'SKIP', 58: 'IPv6-ICMP', 59: 'IPv6-NoNxt', 60:
'IPv6-Opts', 62: 'CFTP', 64: 'SAT-EXPAK', 65: 'KRYPTOLAN', 66: 'RVD', 67:
'IPPC', 69: 'SAT-MON', 70: 'VISA', 71: 'IPCU', 72: 'CPNX', 73: 'CPHB', 74:
'WSN', 75: 'PVP', 76: 'BR-SAT-MON', 77: 'SUN-ND', 78: 'WB-MON', 79: 'WB-EXPAK',
80: 'ISO-IP', 81: 'VMTP', 82: 'SECURE-VMTP', 83: 'VINES', 84: 'IPTM', 85:
'NSFNET-IGP', 86: 'DGP', 87: 'TCF', 88: 'EIGRP', 89: 'OSPF', 90: 'Sprite-RPC',
91: 'LARP', 92: 'MTP', 93: 'AX.25', 94: 'OS', 95: 'MICP', 96: 'SCC-SP', 97:
'ETHERIP', 98: 'ENCAP', 100: 'GMTP', 101: 'IFMP', 102: 'PNNI', 103: 'PIM', 104:
'ARIS', 105: 'SCPS', 106: 'QNX', 107: 'A/N', 108: 'IPComp', 109: 'SNP', 110:
'Compaq-Peer', 111: 'IPX-in-IP', 112: 'VRRP', 113: 'PGM', 115: 'L2TP', 116:
'DDX', 117: 'IATP', 118: 'STP', 119: 'SRP', 120: 'UTI', 121: 'SMP', 122: 'SM',
123: 'PTP', 124: 'IS-IS over IPv4', 125: 'FIRE', 126: 'CRTP', 127: 'CRUDP', 128:
'SSCOPMCE', 129: 'IPLT', 130: 'SPS', 131: 'PIPE', 132: 'SCTP', 133: 'FC', 134:
'RSVP-E2E-IGNORE', 135: 'Mobility Header', 136: 'UDPLite', 137: 'MPLS-in-IP',
138: 'manet', 139: 'HIP', 140: 'Shim6', 141: 'WESP', 142: 'ROHC'}

def check_invalid(window, pro, src, src_port, dst, dst_port, num):
    if(dict_pro[window.m_data_info_list.pro_list[num]] != pro.upper() and pro !=
''):
        return True

    src = src.strip()

    if(window.m_data_info_list.src_list[num] != src and src != ''):
        return True

    if(str(window.m_data_info_list.src_port_list[num]) != src_port and src_port
!= ''):
        return True

    dst = dst.strip()

    if(window.m_data_info_list.dst_list[num] != dst and dst != ''):
        return True

    if(str(window.m_data_info_list.dst_port_list[num]) != dst_port and dst_port
!= ''):
        return True
```

```
    return False

def check_msg_not_in_detail(window, num):
    search_str = window.search_text.strip()
    if(search_str == ''):
        return False
    if(search_str in window.m_data_info_list.detail_info_utf8_list[num]):
        return False
    if(search_str in window.m_data_info_list.detail_info_gb_list[num]):
        return False
    return True


def check_parameter_changed(a, a1, b, b1, c, c1, d, d1, e, e1, f, f1):
    if((a == a1) and (b == b1) and (c == c1) and (d == d1) and (e == e1) and (f
== f1)):
        return False
    return True


def data_info_table_listener(window):
    cur_num = 0
    row_num = 0
    latest_pro = window.parameter_pro
    latest_src = window.parameter_src
    latest_src_port = window.parameter_src_port
    latest_dst = window.parameter_dst
    latest_dst_port = window.parameter_dst_port
    latest_search_text  = window.search_text
    latest_on_off_flag = window.m_sniffer.on_off_flag

    while(True):
        if(window.close_flag == True):
            break

        if(check_parameter_changed(latest_pro, window.parameter_pro, latest_src,
window.parameter_src, latest_src_port, window.parameter_src_port, latest_dst,
window.parameter_dst, latest_dst_port, window.parameter_dst_port,
latest_search_text, window.search_text)):
            cur_num = 0
            row_num = 0
            latest_pro = window.parameter_pro
            latest_src = window.parameter_src
            latest_src_port = window.parameter_src_port
            latest_dst = window.parameter_dst
            latest_dst_port = window.parameter_dst_port
            latest_search_text = window.search_text
            window.data_info_table.clearContents()
            # window.data_info_table.setHorizontalHeaderLabels(["No", "Time",
"Source Address", "Destination Address", "Length", "Protocal"])

        if(latest_on_off_flag != window.m_sniffer.on_off_flag and
latest_on_off_flag == 0):
            cur_num = 0
            row_num = 0
            latest_on_off_flag = window.m_sniffer.on_off_flag
            window.data_info_table.clearContents()
        elif(latest_on_off_flag != window.m_sniffer.on_off_flag):
```

```
            latest_on_off_flag = window.m_sniffer.on_off_flag

        if(cur_num < len(window.m_data_info_list.src_list)):
            if(check_invalid(window, latest_pro, latest_src, latest_src_port,
latest_dst, latest_dst_port, cur_num)):
                cur_num = cur_num + 1
                continue

            if(check_msg_not_in_detail(window, cur_num)):
                cur_num = cur_num + 1
                continue

            window.data_info_table.insertRow(row_num)
            # tmp_time = time.time() - window.start_time
            # tmp_time = round(tmp_time, 4)
            window.data_info_table.setItem(row_num, 0,
QTableWidgetItem(str(cur_num)))
            window.data_info_table.setItem(row_num, 1,
QTableWidgetItem(str(window.m_data_info_list.time_list[cur_num])))
            window.data_info_table.setItem(row_num, 2,
QTableWidgetItem(str(window.m_data_info_list.src_list[cur_num])))
            window.data_info_table.setItem(row_num, 3,
QTableWidgetItem(str(window.m_data_info_list.dst_list[cur_num])))
            window.data_info_table.setItem(row_num, 4,
QTableWidgetItem(str(window.m_data_info_list.len_list[cur_num])))
            window.data_info_table.setItem(row_num, 5,
QTableWidgetItem(str(dict_pro[window.m_data_info_list.pro_list[cur_num]])))
            cur_num = cur_num + 1
            row_num = row_num + 1

    print("data_info_table_listener finished!")
```

main_window.py

定义了主窗口的GUI，使用QWidget。

它包含一个包含数据的数据信息列表，以及一个嗅探器来嗅探数据包。然后，我们启动一个线程来侦听是否需要将新行添加到data_info_list（在function.py中实现）中，使用while true循环进行简单的判断，这种方法不一定妥当但却确实有效。当用户单击开始按钮时，我们启动一个线程进行嗅探。关闭时，我们关闭所有线程并退出。

附录代码如下：

```
import sys
import threading
from scapy.all import *
from PyQt5.QtGui import QFont
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton,QHBoxLayout,
QVBoxLayout, QGridLayout, QTableWidgetItem, QTabWidget, QMessageBox
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QAbstractItemView
from PyQt5.QtCore import Qt
from data_info_list import *
from function import *
from sniffer import *
import netifaces
```

```
import time



class main_window(QWidget):
    def __init__(self):
        super().__init__()
        # self.on_off_flag = 0
        self.close_flag = False
        self.m_data_info_list = data_info_list()
        self.m_sniffer = sniffer()

        self.parameter_pro = ''
        self.parameter_src = ''
        self.parameter_src_port = ''
        self.parameter_dst = ''
        self.parameter_dst_port = ''
        self.search_text = ''
        # self.src_list = []
        # self.dst_list = []
        # self.pro_list = []
        # self.len_list = []

        self.init_window()

    def init_window(self):
        self.main_layout = QGridLayout()
        self.main_layout.setAlignment(Qt.AlignTop)

#       btn1 = QPushButton("button1")
#       btn2 = QPushButton("button2")

# first row of main_layout
# information about nic
        self.nic_info_layout = QGridLayout()

        self.label_nic = QtWidgets.QLabel("NIC")
        # label_nic.setText("NIC")

        self.combo_box_nic = QtWidgets.QComboBox()
        for item in netifaces.interfaces():
            self.combo_box_nic.addItem(item)
        # self.combo_box_nic.addItem(netifaces.interfaces()[0])
        # self.combo_box_nic.addItem(netifaces.interfaces()[1])

        self.on_off_button = QPushButton("Start")
        self.on_off_button.clicked.connect(self.on_off)

        self.save_button = QPushButton("Save")
        self.save_button.clicked.connect(self.save_packets)

        # just for a test
        self.nic_info_layout.addWidget(self.label_nic, 0, 0, 1, 1)
        self.nic_info_layout.addWidget(self.combo_box_nic, 0, 1, 1, 5)
        self.nic_info_layout.addWidget(self.save_button, 0, 8, 1, 2)
        self.nic_info_layout.addWidget(self.on_off_button, 0, 10, 1, 2)
# end of first row
```

```
# second row of main_layout
# protocal source destination and port
        self.ip_info_layout = QHBoxLayout()

        self.label_pro = QtWidgets.QLabel("PRO")
        self.label_source = QtWidgets.QLabel("SRC")
        self.label_source_port = QtWidgets.QLabel("SPORT")
        self.label_dst = QtWidgets.QLabel("DST")
        self.label_dst_port = QtWidgets.QLabel("DPORT")

        self.pro = QtWidgets.QLineEdit()
        self.source = QtWidgets.QLineEdit()
        self.source_port = QtWidgets.QLineEdit()
        self.dst = QtWidgets.QLineEdit()
        self.dst_port = QtWidgets.QLineEdit()

        self.ip_info_layout.addWidget(self.label_pro)
        self.ip_info_layout.addWidget(self.pro)

        self.ip_info_layout.addWidget(self.label_source)
        self.ip_info_layout.addWidget(self.source)

        self.ip_info_layout.addWidget(self.label_source_port)
        self.ip_info_layout.addWidget(self.source_port)

        self.ip_info_layout.addWidget(self.label_dst)
        self.ip_info_layout.addWidget(self.dst)

        self.ip_info_layout.addWidget(self.label_dst_port)
        self.ip_info_layout.addWidget(self.dst_port)

# end of second row

# third row of main_layout
# search bar
        self.search_layout = QHBoxLayout()

        self.line_search = QtWidgets.QLineEdit()
        self.line_search.setPlaceholderText("Search")

        self.search_layout.addWidget(self.line_search)
# end of third row

# fourth row of main_layout
# information of datagram
        self.data_info_layout = QHBoxLayout()

        self.data_info_table = QtWidgets.QTableWidget()
        self.data_info_table.setShowGrid(False)
        self.data_info_table.horizontalHeader().setStretchLastSection(True)
        self.data_info_table.setColumnCount(6)
        self.data_info_table.verticalHeader().setVisible(False)
        self.data_info_table.setSelectionBehavior(QAbstractItemView.SelectRows)
        self.data_info_table.setHorizontalHeaderLabels(["No", "Time", "Source
Address", "Destination Address", "Length", "Protocal"])
        self.data_info_table.setColumnWidth(0, 60);
        self.data_info_table.setColumnWidth(1, 100);
        self.data_info_table.setColumnWidth(2, 240);
```

```
        self.data_info_table.setColumnWidth(3, 240);
        self.data_info_table.setColumnWidth(4, 75);
        self.data_info_table.setColumnWidth(5, 90);
        self.data_info_table.clicked.connect(self.display_detail_info)

        self.data_info_layout.addWidget(self.data_info_table)

        # t = threading.Thread(target=self.data_info_table_listener)
        t = threading.Thread(target = data_info_table_listener, args=(self,))
        t.start()
        # t.join()
# end of fourth row

# fifth row of main_layout
        self.layer_info_layout = QHBoxLayout()

        self.layer_info_tab = QtWidgets.QTabWidget()
        self.layer_info_tab.setFont(QFont('Consolas', 10, QFont.Normal))

        self.layer_info_layout.addWidget(self.layer_info_tab)


# end of fifth row



# sixth row of main_layout
# detail information like header of packet

        self.detail_layout = QVBoxLayout()

        self.detail_label = QtWidgets.QLabel("Detail Information")
        # self.detail_of_packet = QtWidgets.QTextBrowser()
        self.detail_of_packet = QtWidgets.QTabWidget()
        self.detail_of_packet.setFont(QFont('Consolas', 10, QFont.Light))

        self.detail_layout.addWidget(self.detail_label)
        self.detail_layout.addWidget(self.detail_of_packet)
# end of sixth row

        self.main_layout.addLayout(self.nic_info_layout, 0, 0)
        self.main_layout.addLayout(self.ip_info_layout, 1, 0)
        self.main_layout.addLayout(self.search_layout, 2, 0)
        self.main_layout.addLayout(self.data_info_layout, 3, 0)
        self.main_layout.addLayout(self.layer_info_layout, 4, 0)
        self.main_layout.addLayout(self.detail_layout, 5, 0)

        self.setLayout(self.main_layout)
        self.setGeometry(300, 300, 815, 800)
        self.setWindowTitle("EzSniffer")
        self.show()

    def on_off(self):
        if(self.m_sniffer.on_off_flag == 0):
            # self.on_off_flag = 1;
            self.m_data_info_list.clear()
            # self.data_info_table.clearContents()
```

```
            # self.data_info_table.setHorizontalHeaderLabels(["No", "Time",
"Source Address", "Destination Address", "Length", "Protocal"])
            self.m_sniffer.on_off_flag = 1;
            self.on_off_button.setText("Stop")
            # t = threading.Thread(target=self.begin_sniff)
            t = threading.Thread(target = self.m_sniffer.begin_sniff, args=
(self, ))
            t.start()
        else:
            self.on_off_button.setText("Start")
            self.m_sniffer.on_off_flag = 0;
            # send end packet
            send(IP(dst = '127.0.0.1')/ICMP())
            send(IP(dst = '220.181.38.251')/ICMP())
            # self.on_off_flag = 0;

    def display_detail_info(self):
        num = int(self.data_info_table.selectedItems()[0].text())
        self.layer_info_tab.clear()
        self.detail_of_packet.clear()

        i = 0
        for x in self.m_data_info_list.layer_list[num]:
            temp_tab_text = QtWidgets.QTextBrowser()
            for key,value in self.m_data_info_list.field_list[num][i].items():
                temp_tab_text.append(str(key) + ":    " + str(value))

            self.layer_info_tab.addTab(temp_tab_text, x)
            i = i + 1

        temp_tab_text = QtWidgets.QTextBrowser()
        # print(self.m_data_info_list.hex_list[0])
        temp_tab_text.setText(self.m_data_info_list.hex_list[num])
        self.layer_info_tab.addTab(temp_tab_text, "Whole in Hex")

        temp_tab_text = QtWidgets.QTextBrowser()
        temp_tab_text.setText(self.m_data_info_list.detail_info_utf8_list[num])
        self.detail_of_packet.addTab(temp_tab_text, "utf-8")

        temp_tab_text = QtWidgets.QTextBrowser()
        temp_tab_text.setText(self.m_data_info_list.detail_info_gb_list[num])
        self.detail_of_packet.addTab(temp_tab_text, "GB2312")
        #
self.detail_of_packet.setText(self.m_data_info_list.detail_info_gb_list[num])

    def update_parameter(self):
        self.parameter_pro = self.pro.text()
        self.parameter_src = self.source.text()
        self.parameter_src_port = self.source_port.text()
        self.parameter_dst = self.dst.text()
        self.parameter_dst_port = self.dst_port.text()
        self.search_text = self.line_search.text()

    def save_packets(self):
        QMessageBox.information(self, "Oops!", "Save successfully!",
QMessageBox.Yes)
        wrpcap("output" + time.strftime("%Y%m%d%H%M%S", time.localtime()) +
".pcap", self.m_data_info_list.packet_list)
```

```
    def keyPressEvent(self, event):
        if(event.key() ==  16777220 or event.key() == 16777221):
            self.update_parameter()

    def closeEvent(self, event):
        self.close_flag = True
        self.m_sniffer.on_off_flag = 0
        # time.sleep(0.5)
```

三. 功能说明

本次项目构建参照大作业要求，完成了以下基础功能，并在其功能基础上搭建了用户友好的GUI图形界面，完成了一个可视化的嗅探器程序。在具体功能设计上以wireshark为目标。

- 实现了侦听指定网卡的基本功能，对进出指定网卡的数据包进行了抓捕和解析，并尽可能包含了较多类型的报文（ICMP，UDP，TCP等），具有良好的可用可读性。
- 对TCP、UDP数据报文实现了分片重组，具体可以参考第四部分实验测试中的分片重组部分。
- 实现了数据的查找和过滤，对报文的内容按照一定的条件进行筛查，参考实验测试。
- 可以在抓捕时采取包过滤，指定协议类型，源，目的地址，端口号等。
- 保存数据包为可打开的格式，这里保存为pcap格式，可以由wireshark软件打开。
- 我们的Sniffer的网络状态监视功能强大，可以监视流量、带宽、协议等信息,并且以图形界面形式显示出来。功能细致，严格按照协议进行分层，细节考虑完整。
- 执行ping，telnet，浏览网页或传输文件时，可以对应的返回正确的结果。
- 文件传输功能有待开发，目前暂未实现稳定可靠的功能，这部分开发尚未push在repo中。

四. 实验测试

> **注：由于程序功能较为复杂，在此部分提供截图来显示测试部分，详细功能测试可参考录屏文件或自行运行程序测试。**

抓包解析，更换网卡：

打开程序，在NIC接口列表中选择虚拟网卡，点击开始键开始抓包。观察详细信息解析，单击某个包，可选择不同选项卡点击各栏目查看。

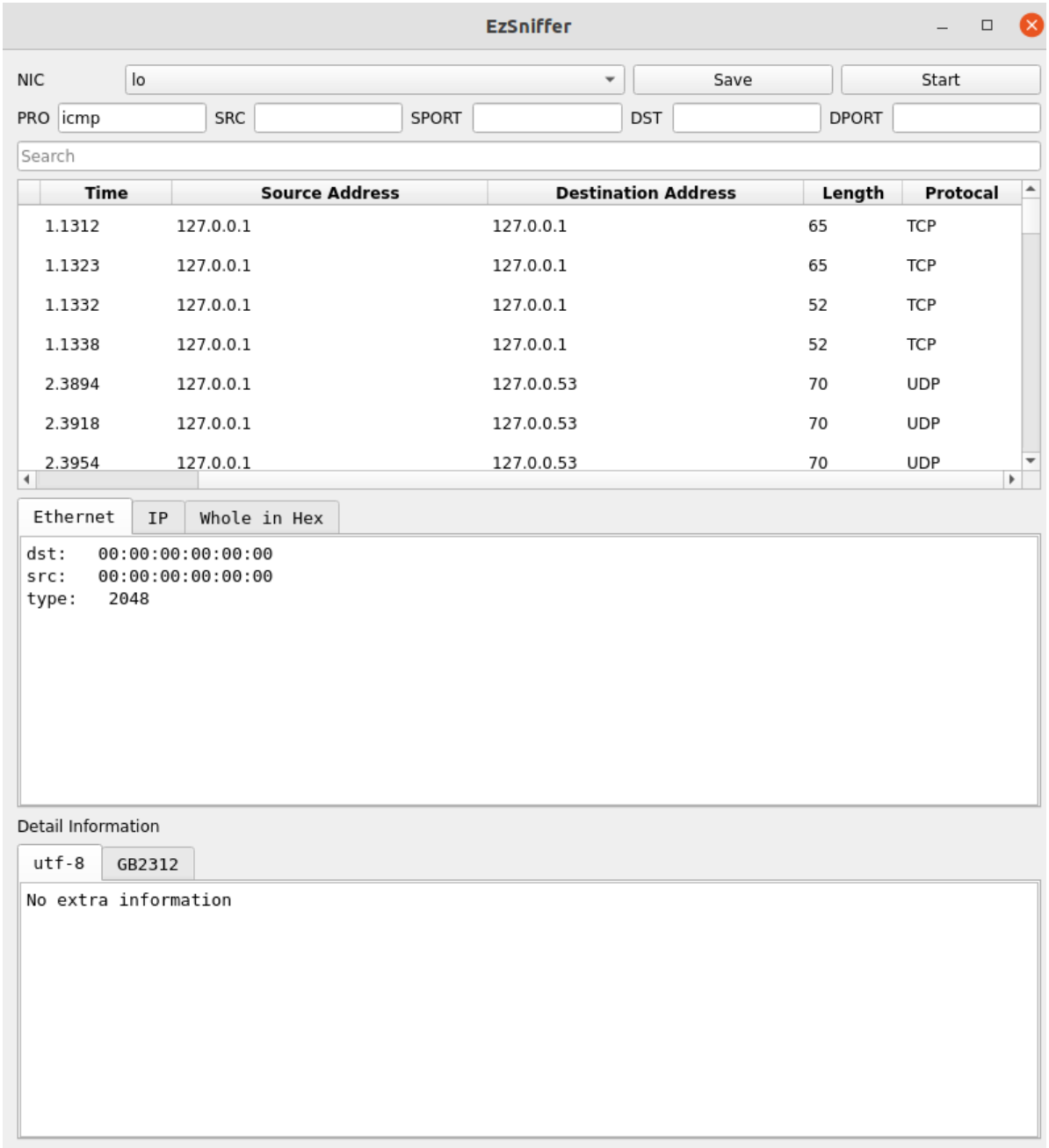
EzSniffer
NIC
lo
Save
Start
PRO
icmp
SRC
SPORT
DST
DPORT
Search
Time
Source Address
Destination Address
Length
Protocal
1.1312 127.0.0.1 127.0.0.1 65 TCP
1.1323 127.0.0.1 127.0.0.1 65 TCP
1.1332 127.0.0.1 127.0.0.1 52 TCP
1.1338 127.0.0.1 127.0.0.1 52 TCP
2.3894 127.0.0.1 127.0.0.53 70 UDP
2.3918 127.0.0.1 127.0.0.53 70 UDP
2.3954 127.0.0.1 127.0.0.53 70 UDP
Ethernet
IP
Whole in Hex
dst: 00:00:00:00:00:00
src: 00:00:00:00:00:00
type: 2048
Detail Information
utf-8
GB2312
No extra information

图5：TCP报文

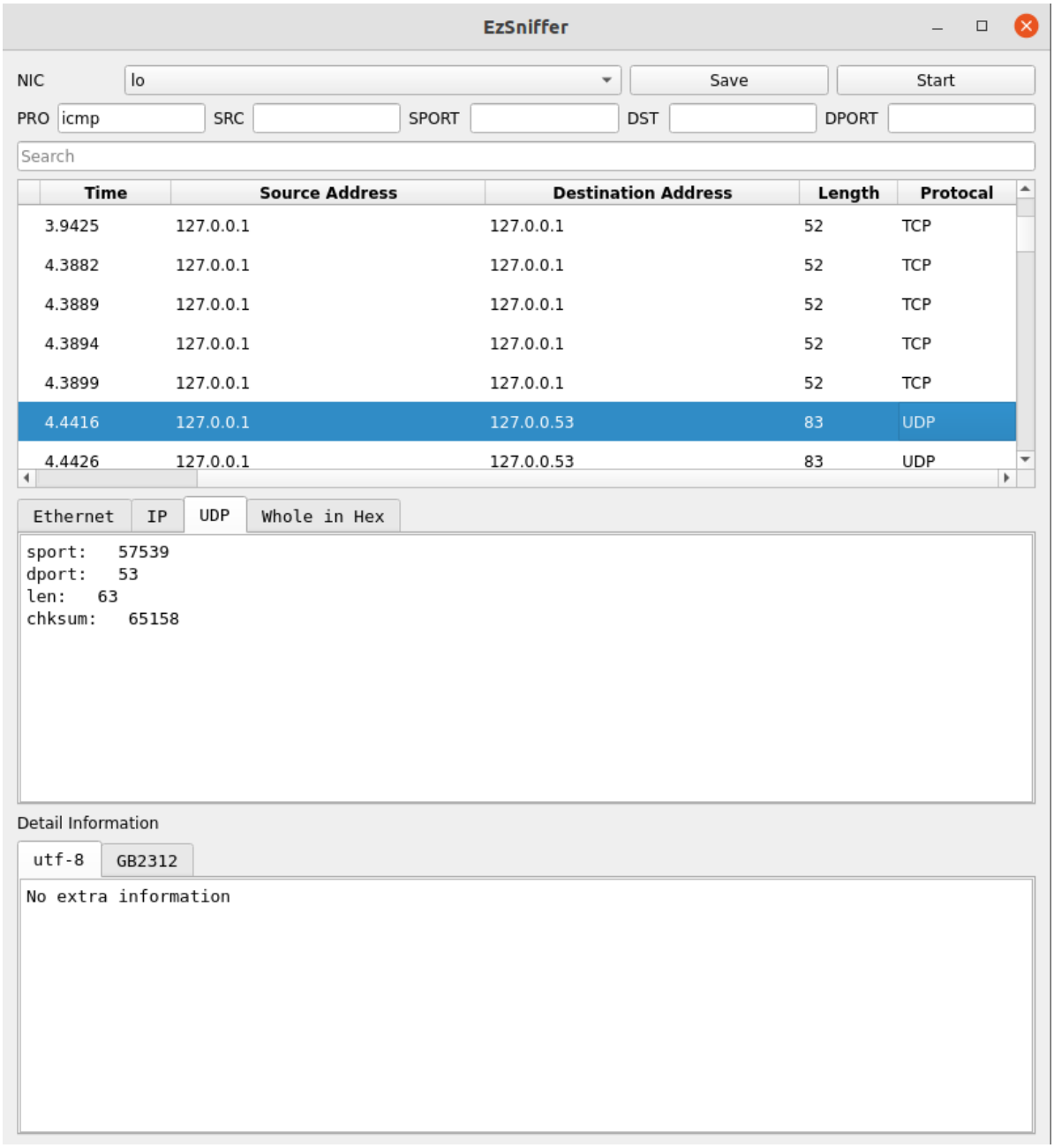

图6：UDP报文

可以通过观察数据报文的IP字段都是127网段，确认选择为lo网卡；

点击Stop，切换网卡，从lo自环切换到ens33虚拟网卡（NAT连接到Windows主机的网卡，这里即VMNet8）；继续点击Start抓包观察，此时可以打开Firefox或者ping一个外部网络进行观察。

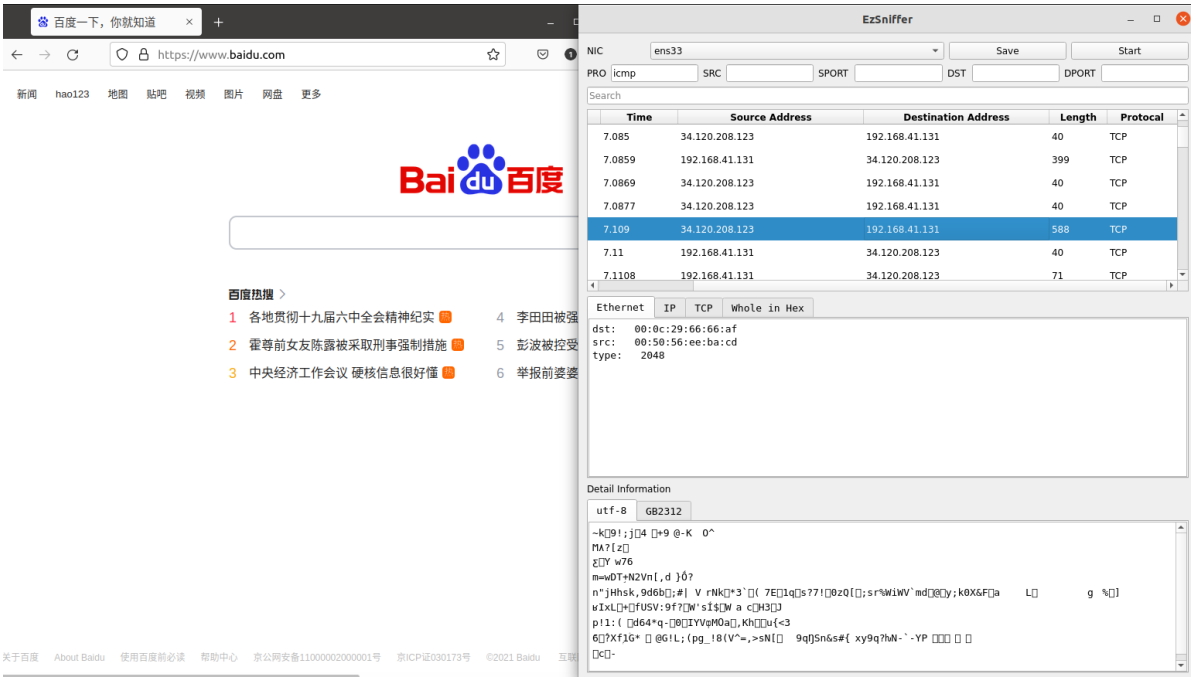

图7：切换到ens33网卡

包分片重组：

使用local网卡，ping指令发送长度为1800的数据包，在sniffer中抓包得到长度为1828的数据包，这里可以看到其已经重组完成。

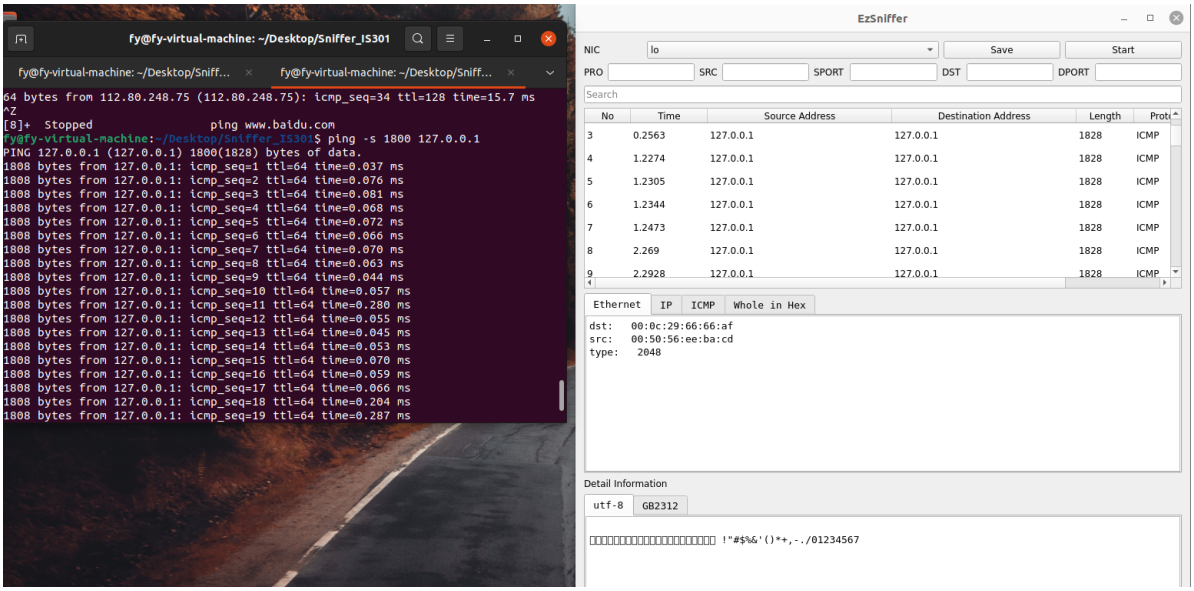

图8：报文分片重组

为了显示对比，将实现重组的frag_set去掉，再次抓包，此时得到的数据包将分片。

5	4	17.1512	192.168.33.11	220.181.38.251	1500	ICMP
6	5	17.1512	192.168.33.11	220.181.38.251	348	ICMP

图9：报文已分片

包过滤查询：

可以指定一些搜索条件，在搜索时得到需要的结果：这里我们在PRO筛选框中输入TCP（支持小写），并且**按下回车**确认条件赋予，点击start即可筛查出tcp报文。同理还可以加上源地址的筛选条件如192.168.41.131：

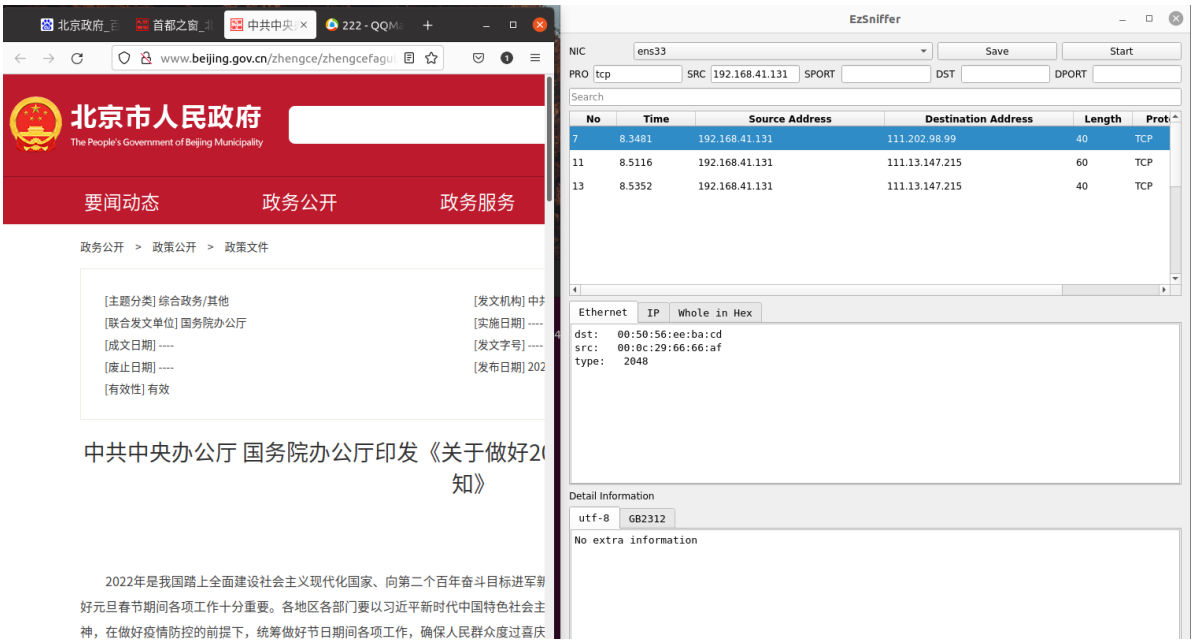

图10：包过滤抓包

数据包查询：

这里需要检测搜索条件是否已更改，由ENTER键触发。因此，latest_xxx用于保存历史条件，并与当前条件（即窗口）进行检查。

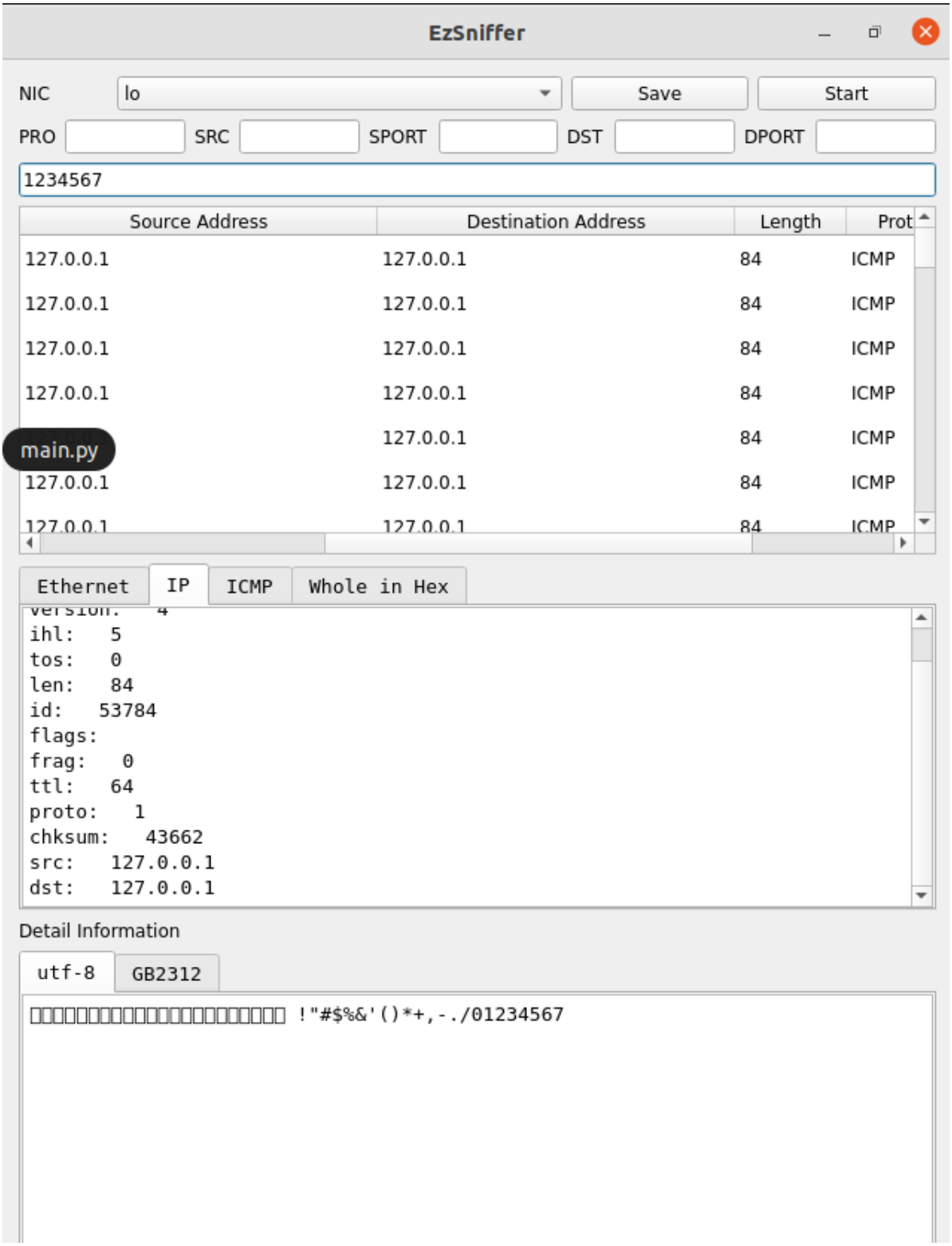

图11：在结果中查询内容

文件保存：

选中一些数据包，点击保存按钮，将刚才的数据包存为文件。

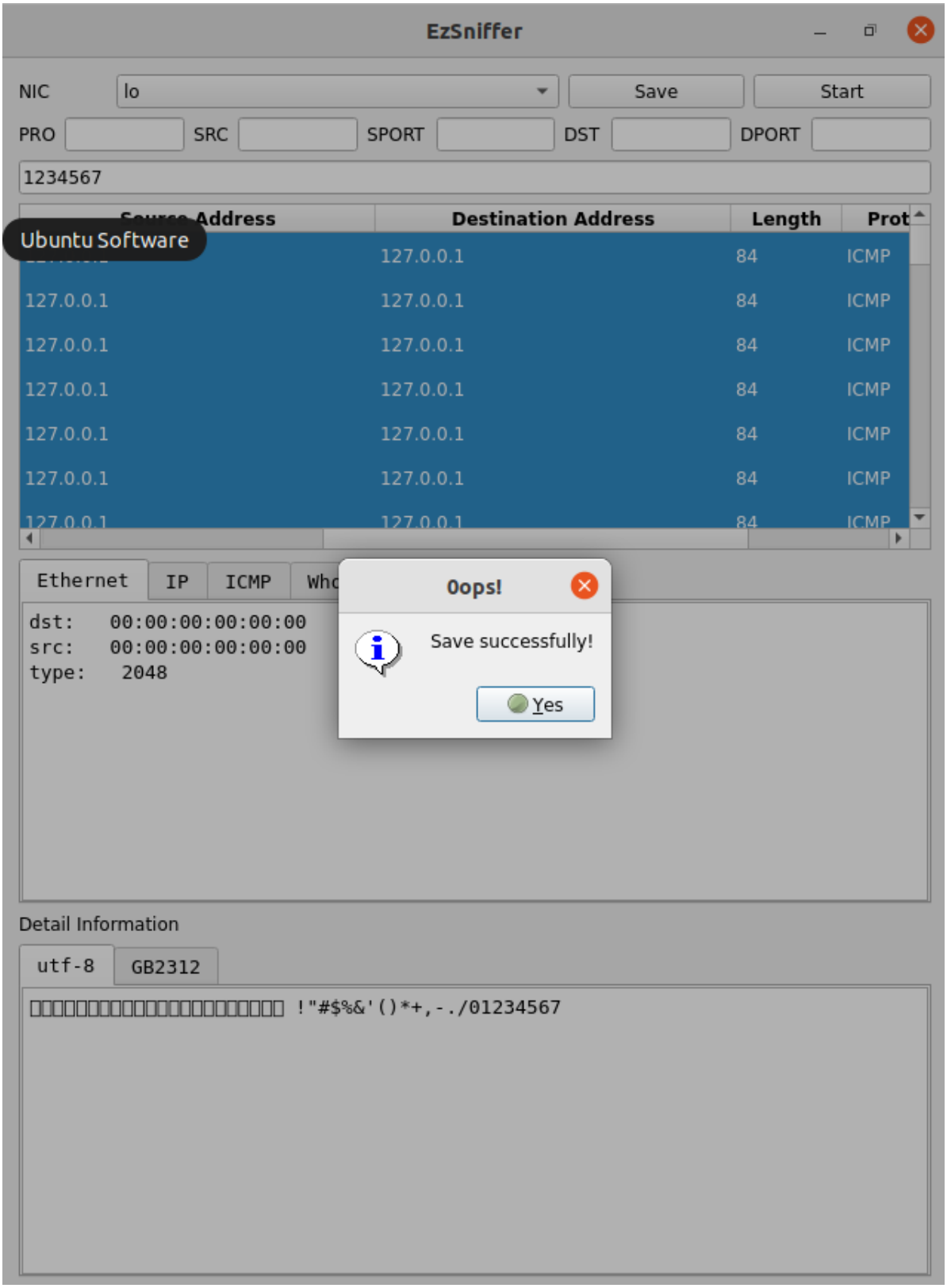

图12：选中内容并点击save

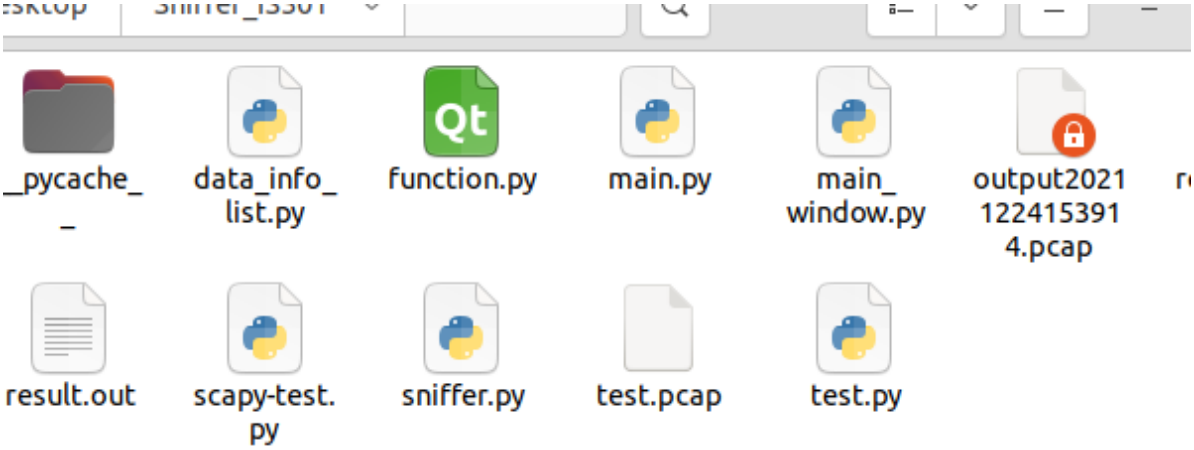

图13：指定目录下保存为pcap文件

使用本程序和WireShark分别打开该文件，可用性良好。

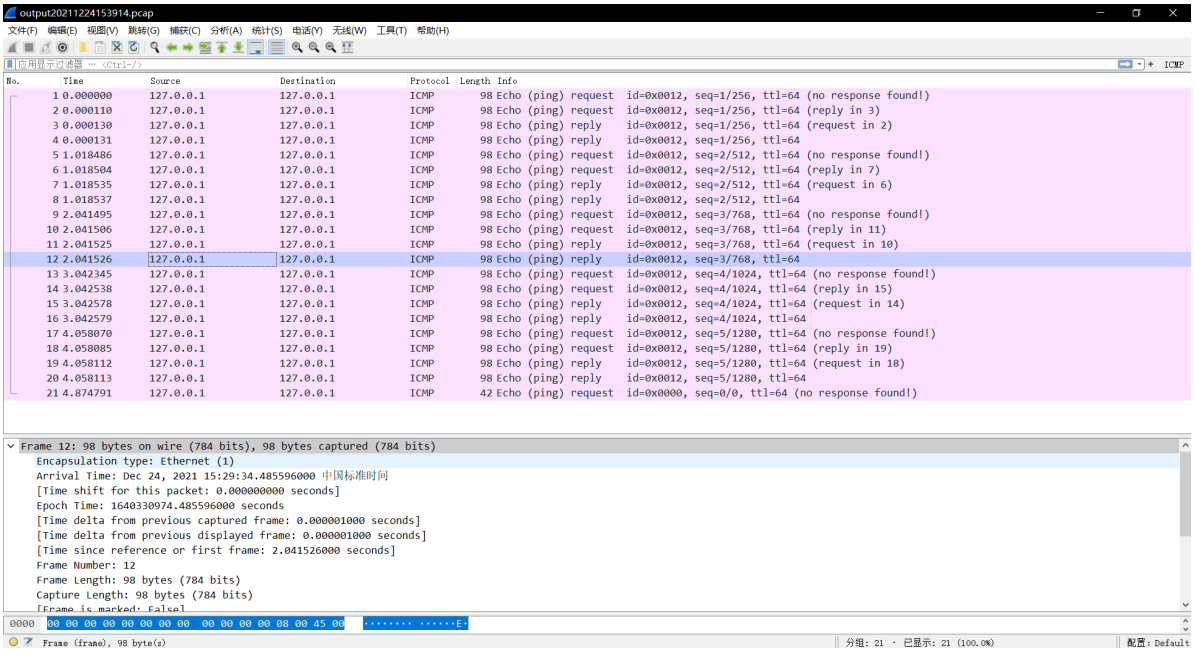

图14：使用wireshark打开pcap文件

五. 问题与解决

1.程序无法正常结束，每次退出需要Ctrl C退出。原因是scapy sniff方法中的stop_filter是针对包操作的，如果在抓包时没有收到包，它就不会执行stop_filter函数，导致不会执行结束判断，解决方法是在结束的时候主动发结束包，在这里我们发了两个icmp包，一个发往127.0.0.1，一个发往baidu.com，这样就可以正常结束sniff线程。

2.在实现过滤功能的时候，一开始只有protocol能够正常过滤，根据源地址，源端口，目的地址，目的端口都不能实现过滤。发现问题出在键盘的enter上，因为我的键盘是104键包含小键盘的，我输入协议（比如icmp）时，按的enter是中间那个enter，但我输入源地址（比如127.0.0.1）时，按的是小键盘的enter，这两个enter事件号不一样，导致按小键盘的enter不能触发过滤更新条件，解决方法就是在触发事件号里面多加一个enter。

3.在解析包里面的额外信息时，我们尝试使用utf8或者gb2312进行解码，但是经常失败，只有http请求报文的信息能够正常显示，其他比如telnet的信息都不能正常显示。这是因为有些协议的头部会含有例如0xff这样不能使用utf8或者gb2312进行解码的字符，但是又不希望使用decode('utf8', errors="ignore")的方式，这样会把一些不该显示的东西显示出来，所以最后采取了一个折中的办法，就

是我们去判断它不能用字符显示的部分是不是只在协议头，如果只在协议头我们就把协议头去掉然后decode剩余的部分，所以最后能够显示出一些额外信息。

4.在显示whole in hex的时候我尝试使用hexdump(packet)这样的形式来获取它的hex格式，但是这个版本的scapy的hexdump是直接将hex格式输出在标准输出流里，而且返回值并不会返回string，考虑的解决方法包括将输出流重定向到某个文件，然后根据序号依次读它的hex格式，但是操作了很久这块也没实现出来，所以最后采用了str(packet)这样简陋的实现方法。

5.保存pcap包的时候曾经考虑过加入读pcap包并显示的功能，但是我们强烈推荐使用wireshark打开保存的pcap包，界面简洁又好看，信息也很全。如果要做这个功能需要开发一个只读模式，重新开发一个界面，不然的话在读这个文件的时候万一又要监听网卡会导致行为很奇怪。

6.刷新中间显示详细信息的table的时候它会把长度重置的很奇怪，我调用了pyqt5 tablewidget的clearContent方法，这我真没办法解决，而且它右边的进度栏也不会自动跳到开头，只能手动往上拉。

7.详细信息栏里面的protocol是分辨不出来应用层协议的，因为使用的是packet['IP'].proto的协议号加上一个转换为协议名的字典dict_pro，这里面不会有应用层协议，但是如果要搜索http协议，可在搜索栏中搜索http，会在额外信息中输出http报文（强烈推荐搜索例如GET，POST之类，这样就肯定是http报文，有些很奇怪的东西里面也会有http）

8.包分片重组，有些包它的flag里面是空的我就没重组，最明显体现包重组的实验就是ping -s 1800 127.0.0.1，因为这样保证一定能ping通，如果ping -s 1800 baidu.com貌似ping不通，baidu不会回应这种大小的包。另外可能是电脑配置原因，我的环境在没有加入这个功能的时候就能够收到包大小大于mtu的包，不是很了解为什么会产生这种原因，但是当时使用ping的时候能很明显看出mtu的作用，所以实验也是使用ping。

9.在function.py里面的data_info_table_listener里面写的并不好，对于功能上没有任何问题但是在性能上不好，在这个线程里面我使用了一个while(True)循环来实时检测有无更新情况，感觉这里应该做成事件驱动的会在性能上好很多，在跑这个程序的时候我的风扇一度很大声^_^

10.文件重组为什么没实现，首先是想不到一个通用的方法来解决这个问题，感觉需要针对不同协议来实现不同的方法重组文件（比如ftp和http），又或者是重组某些特定文件，就是根据文件头的特点来做特殊操作（比如png,jpg之类的），另外主要就是代码水平有限，即使想到解决方法也不一定能够解决这个问题。

11.Start和Stop按钮按下时有小概率不会刷新中间的table，并且此时也不能抓包（或者是包是抓到了但没显示出来），我们怀疑是在线程里出现了奇怪的行为，比如包没收完就停了导致卡在里面了，这个bug甚至不太好复现，所以未解决这个问题。

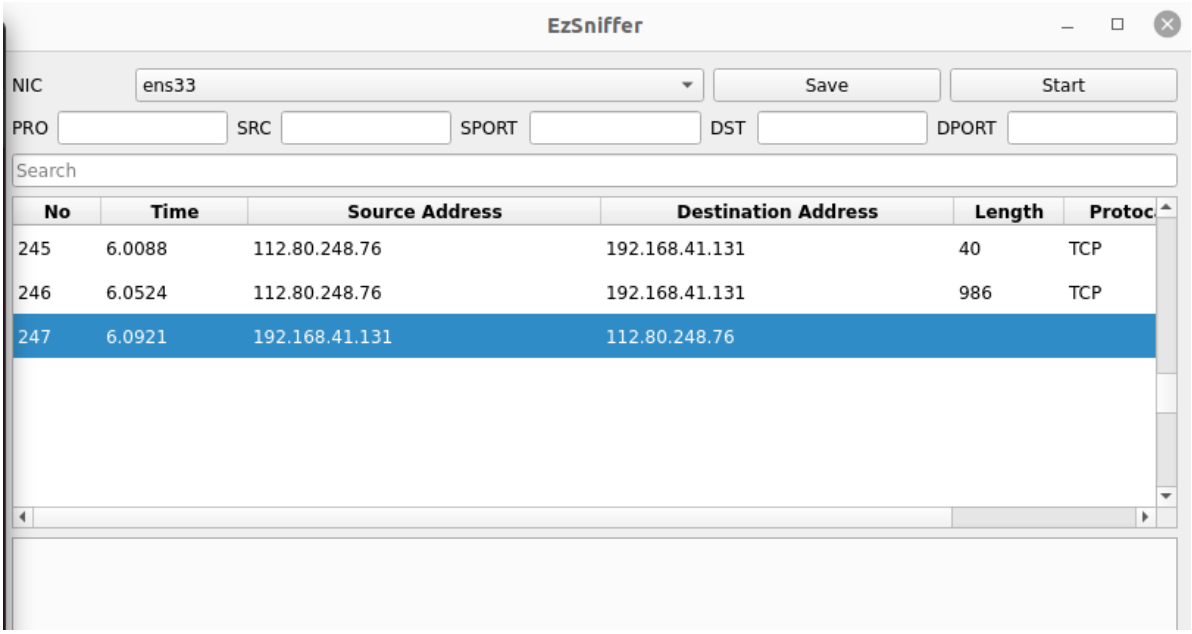

图15：出现的死锁bug

六. 项目心得

费扬：

本次大作业是几个选题中较有难度的一项，无论是从头学习PyQt5的UI设计还是使用Python的scapy库来抓包解析，都需要经历学习，模仿，构建，修正的全部学习流程。也就是说，为了达到预期效果，很多时候一行代码的事情需要花费数倍于此的学习成本。但当最后的UI设计展现出预期的样子，并通过接口实现功能的时候，觉得还是值得的。

这个项目是基于Linux的，更接近底层数据架构和各类协议，我们发现Sniffer这个项目可以帮助我们理解计算机通信网络的内容，无论是数据包的结构还是各类协议的特征，我们都可以通过实践加深知识储备。

虽然由于时间和水平的限制，在一些机制和算法运用上还有失妥当，在用户友好的UI设计中也尚未达到完善，希望在之后的空闲时间中，把数据信息和各类字段更好的安排在一个GUI中，可以使用designer进行可视化设计，运用connect的方式接入；同时可以设计自动补全，search框提示，鼠标触碰变化等细节功能，真正的考虑到用户的体验。

感谢老师在课上和课后的指导讲解，感谢助教学长的指导和介绍，通过这次大作业实践，我得到了代码能力和自主学习能力的极大提高锻炼。

朱文骏：

本次大作业我在时间分配上出了比较严重的问题，前期写的太慢了导致后期赶进度的时候写的很急，很多功能都是采取的最简单直接的解决方法。我觉得在这个项目的一开始我还是写的很好的，基本是一天写然后第二天重构第一天的代码，基本实现了面向对象，封装也做的还行，当时除了那个监听table变化while(True)循环我基本都挺满意的，但后面实在是急了，我觉得最明显的看sniffer.py里面那个packetHandler应该就可以看出我们的着急，正常写我绝不会写一个如此臃肿，复用性差的函数，但它的优点是快速可用，所以当时就是这么做的。

那么导致本次大作业搞成如此着急的样子的原因是我的一次巨大失误，我git reset错了，我git reset --hard HEAD^,我当时以为我存档了但是我没有，所以往前退了一个版本，基本就是一天白忙活，然后我心态就发生了一些微妙的变化（所以熟练git操作非常重要）。

在这次大作业中我更加理解了网络数据包的结构，对于计算机网络有了更深的理解。