项目名称：网络嗅探器

学号：521030910286　姓名：张瀚文

0. 项目概述

本项目开发的平台为Ubuntu22.04虚拟机，使用Python3.9编写

实现的功能包括：

- 指定监听的网卡捕获进出本机的数据报
- 对捕获的数据报进行分解，分析其各层使用的协议及各层的关键信息
- 将分片的IP数据报进行重组
- 将捕获的数据报信息保存至用户指定的文件中
- 设计了一个较为美观实用的UI界面

运行本项目需要额外安装的python库（在虚拟环境中安装）：

- scapy
- ttkthemes

1. 项目结构介绍

整个项目只包括一个代码文件。这个文件主要分为两部分：数据报处理部分、UI界面设计部分。

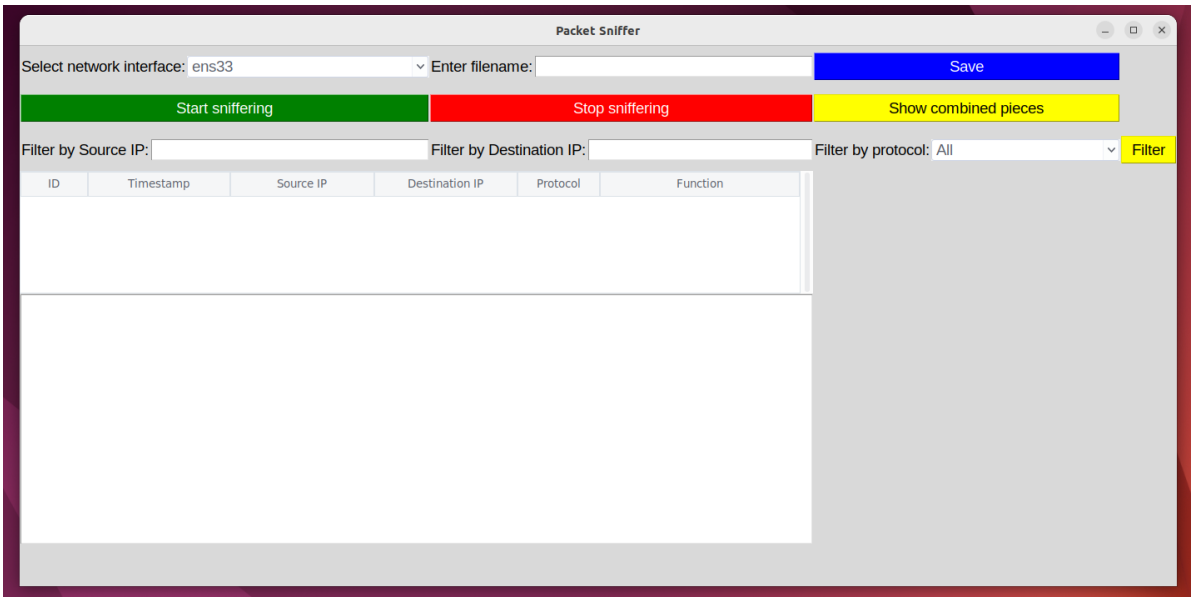
1.1 数据报处理部分

实现的功能主要有：

- 捕获数据报并判断其类型，将分析后的结果存入全局变量以供展示
- 根据用户的过滤条件筛选出符合要求的数据报，然后展示
- 在用户点击查看某一数据报时，显示数据报内具体内容
- 将分片的IP数据报重组，并可以筛选出所有重组后的数据报

1.2 UI界面设计部分

最终设计结果如下：



2. 项目功能分析

2.1 系统参数设定

在测试项目前，使用者需要查看本机网卡编号（例如我的是ens33）和MTU值。

```
# 需要用户修改，为项目指定一个正确的网卡
iface = tk.StringVar()
iface.set('ens33')
# 根据自己主机的适配器信息进行修改
iface_option['values'] = ('ens33', 'wlan0')
```

```
# 若数据报长度超过MTU，说明该数据报为重组后添加进全局变量。用户需要根据本机情况来修改
for packet in packets:
    if packet["Length"] > 1500:
...
```

2.2 对于全局变量的解释

```
keep_sniffing = True          # 是否处于工作模式标识
seen_packets = set()          # 用于记录已经捕获到的数据报，防止重复捕获造成冗余
packets = []                  # 记录捕获到的数据报的信息
filter_packets = False        # 若当前展示已过滤的数据报，则置为True
is_filtered = False           # 若当前展示已重组的数据报，则置为True
fragments = {}                # 用于记录捕获到的分片
reassembled_packets = []      # 用于记录重组后的数据报
packet_id = 0                 # 给每一个捕获到的数据报一个ID
```

2.3 捕获数据报

```
def packet_callback(packet):
    global ...                                      # 声明外部全局变量

    if not keep_sniffing:                           # 若不处于工作模式，则直接退出
        return
    packet_identifier = packet.summary()            # 生成数据报的摘要来过滤掉重复的数据报
    if packet_identifier not in seen_packets:       # 生成描述数据报的相关信息
        packet_dict = {}
        packet_dict["Packet Identifier"] = packet_identifier
        packet_dict["Timestamp"] = packet.time
        packet_dict["Length"] = len(packet)
        if packet.haslayer(Raw):
            packet_dict["Load"] = binascii.hexlify(packet.load).decode()     # 将
负载部分转换为十六进制数
        else:
            packet_dict["Load"] = None
        # 存储信息的局部变量
        source_ip = ''
        destination_ip = ''
        protocol = ''
        function = packet_identifier
        source_port = None
        destination_port = None
```

```
        # 分析数据报内容
        if packet.haslayer(Ether):
            ...
        if packet.haslayer(ARP):
            ...
        elif packet.haslayer(IPv6):
            ...
        elif packet.haslayer(IP):
            ...
            if packet.haslayer(ICMP):
                ...
            elif packet.haslayer(TCP):
                ...
            elif packet.haslayer(UDP):
                ...
            # 处理分片
            ip_id = packet[IP].id
            ip_frag = packet[IP].frag
            if packet[IP].flags.DF == False:                    # DF = False时才
分片
                if ip_id not in fragments:
                    fragments[ip_id] = {}
                fragments[ip_id][ip_frag] = packet[IP].payload  # 将标识字段相同的分
片进行存储
        # 将收集到的信息存储起来
        packet_id += 1
        packet_dict["ID"] = packet_id
        packet_dict["Source IP"] = source_ip
        packet_dict["Destination IP"] = destination_ip
        packet_dict["Protocol"] = protocol
        packet_dict["Function"] = function
        packet_dict["Source Port"] = source_port
        packet_dict["Destination Port"] = destination_port
        packet_dict["IP ID"] = ip_id  # 将IP ID添加到数据包字典中
        packet_dict["IP DF flag"] = packet[IP].flags.DF if packet.haslayer(IP)
else None
        packet_dict["IP MF flag"] = packet[IP].flags.MF if packet.haslayer(IP)
else None
        # 将该数据报添加到全局变量
        packets.append(packet_dict)
        # 将该数据报标记为已收到
        seen_packets.add(packet_identifier)
```

2.4 过滤数据报

```
def toggle_filter():
    global ...                                  # 声明全局变量
    filter_packets = not filter_packets         # 修改标记

    # 清空 Treeview，即将当前展示的数据报信息清空
    for i in tree.get_children():
        tree.delete(i)

    # 重新处理已经接收到的数据包
    for packet in packets:
```

```
        source_ip = packet["Source IP"]
        destination_ip = packet["Destination IP"]
        protocol = packet["Protocol"]
        # 根据过滤条件筛选
        if filter_packets:
            if source_ip_filter.get() and source_ip != source_ip_filter.get():
                continue
            if destination_ip_filter.get() and destination_ip !=
destination_ip_filter.get():
                continue
            if protocol_filter.get() != 'All' and protocol !=
protocol_filter.get():
                continue

        # 将符合条件的数据包添加到 Treeview
        tree.insert('', 'end', values=(packet["ID"], packet["Timestamp"],
source_ip, destination_ip, packet["Protocol"], packet["Function"]))
```

2.5 展示数据报细节

```
def show_packet(event):
    global ...                      # 声明全局变量
    curselection = tree.focus()     # 获取当前选中的项
    if curselection:                # 如果有选中的项
        packet_text.delete(1.0, tk.END)
        selected_id = tree.item(curselection)['values'][0]  # 获取选中项的 'ID' 列
的值
        # 使用 selected_id 查找 packets 列表中的相应数据包
        packet = next((p for p in packets if p["ID"] == selected_id), None)
        if packet is not None:
        timestamp = packet["Timestamp"]          # 将Unix时间戳转换为可读的日期和时间
        if isinstance(timestamp, str):           # 检查timestamp的类型
            # 如果timestamp是字符串，那么我们假设它是一个日期时间字符串
            dt_object = datetime.datetime.strptime(timestamp, "%Y-%m-%d
%H:%M:%S")
        else:
            # 否则，我们假设timestamp是一个Unix时间戳，并对其进行格式化
            dt_object = datetime.datetime.fromtimestamp(int(timestamp))
            formatted_time = dt_object.strftime("%Y-%m-%d %H:%M:%S")
            packet["Timestamp"] = formatted_time
            packet_text.insert(tk.END, json.dumps(packet, indent=4))

    else:  # 如果没有选中的项
        packet_text.delete(1.0, tk.END)
        packet_text.insert(tk.END, "No packet selected.")
```

2.6 过滤重组后的数据报

```
def filter_by_length():
    global is_filtered
    # 清空 Treeview
    for i in tree.get_children():
        tree.delete(i)
```

```
    if is_filtered:
        # 如果当前是过滤状态，那么显示所有数据包
        for packet in packets:
            tree.insert('', 'end', values=(packet["ID"], packet["Timestamp"],
packet["Source IP"], packet["Destination IP"], packet["Protocol"],
packet["Function"]))
    else:
        for packet in packets:
            if packet["Length"] > 1500:             # 根据MTU判断哪些数据报是重组后的
                tree.insert('', 'end', values=(packet["ID"],
packet["Timestamp"], packet["Source IP"],
packet["Destination IP"], packet["Protocol"], packet["Function"]))

    # 切换 is_filtered 的值
    is_filtered = not is_filtered
```

2.7 UI界面设计

```
root = ThemedTk(theme="arc")                    # 创建一个基础窗口
root.title("Packet Sniffer")                    # 窗口标题
root.geometry("1400x900")                       # 默认的窗口大小
my_font = font.Font(family="Arial", size=14)    # 设置个性化字体
```

```
# 设置行和列的权重，使得它们可以随窗口尺寸变化
for i in range(6):
    root.grid_rowconfigure(i, weight=1, minsize=50)
for i in range(2):
    root.grid_columnconfigure(i, weight=1, minsize=100)
```

其他组件的设计方法为：

- 创建一个Frame，设置该Frame在root窗口上的位置
- 将组件注册到Frame内以完成布局

所有组件的排布逻辑都是类似的，在此不一一赘述

3. 项目操作说明

3.1 开始捕获数据报

在命令行执行代码文件（虚拟机下需要使用管理员权限）

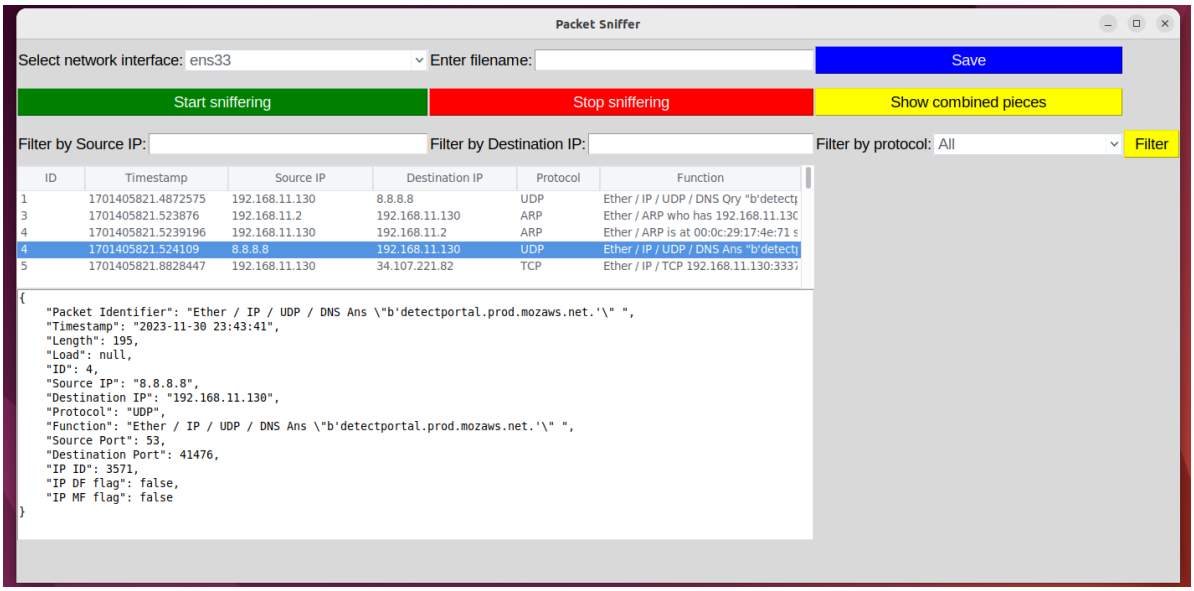
```
sudo python3 sniffer.py
```

点击窗口中的Start sniffering按钮，捕获到的数据报便会显示出来。点击某一条数据报信息，下面的文本框中便会显示细节.

点击按钮后程序调用$start_sniffering()$函数，创建新线程开始抓包

```
def start_sniffing():
    global sniff_thread
    global keep_sniffing
    keep_sniffing = True
    sniff_thread = Thread(target=sniff, kwargs={'iface': iface.get(), 'prn':
packet_callback, 'store': 0},
daemon=True)
    sniff_thread.start()
```

效果图如下：



其中细节部分包括：数据报的摘要、时间戳、数据报长度、负载、IP头的标识、源IP、目的IP、DF标志等等

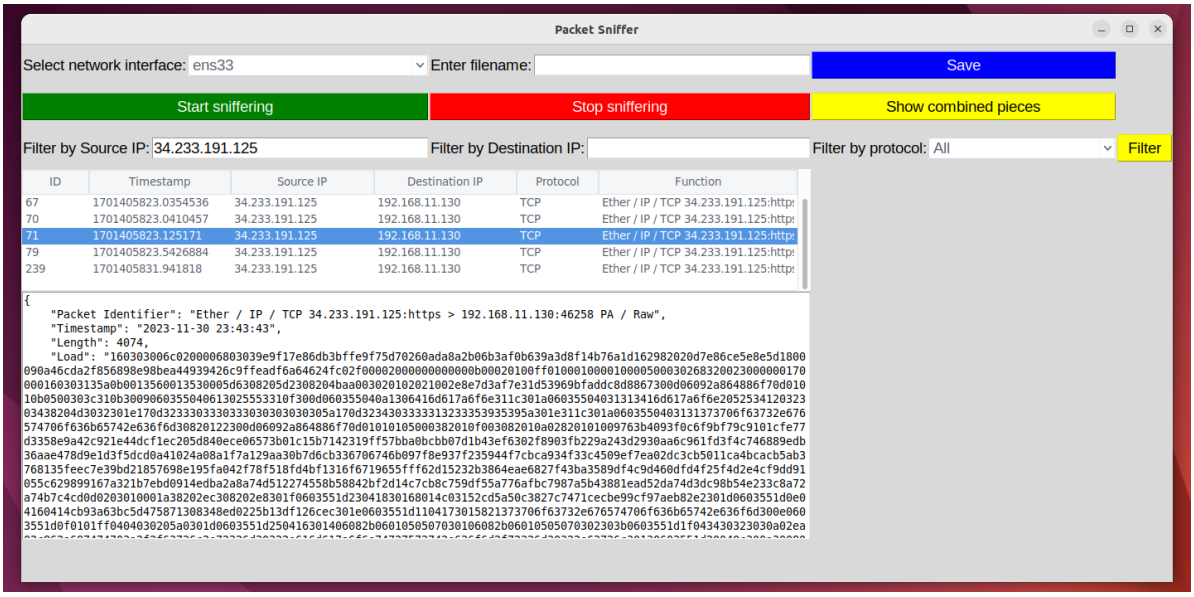
值得注意的是负载部分的值单独查看是没有意义的，我在设计时只令其按照十六进制数显示，若要分析可以将其复制出去

注意：**有时在显示捕获的数据报时ID会出现重复和跳过某些值的问题，此时点击一下Filter即可**

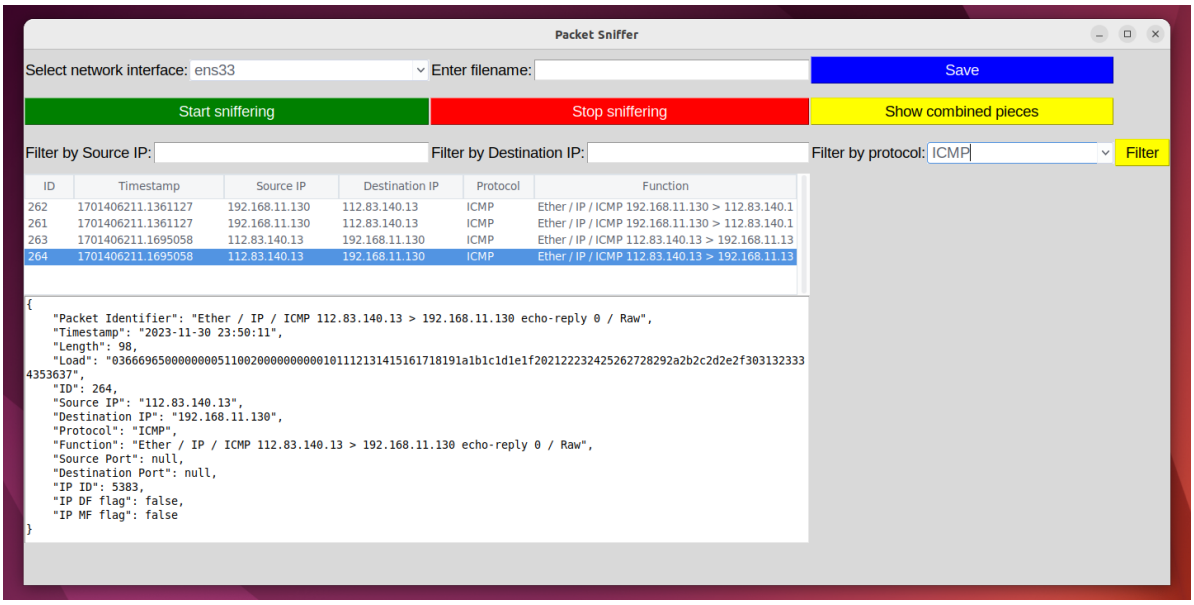
3.2 根据需求过滤数据报

在输入框输入目的IP和源IP，并选择感兴趣的协议类型（默认为全部协议），点击Filter按钮进行筛选

- 只根据源IP进行过滤

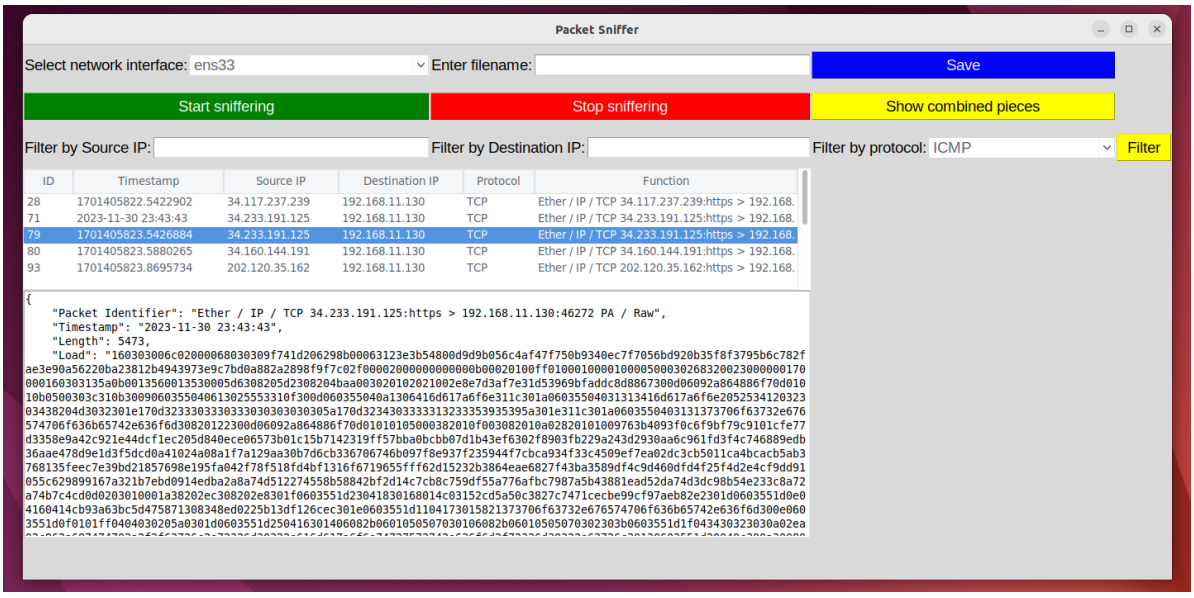


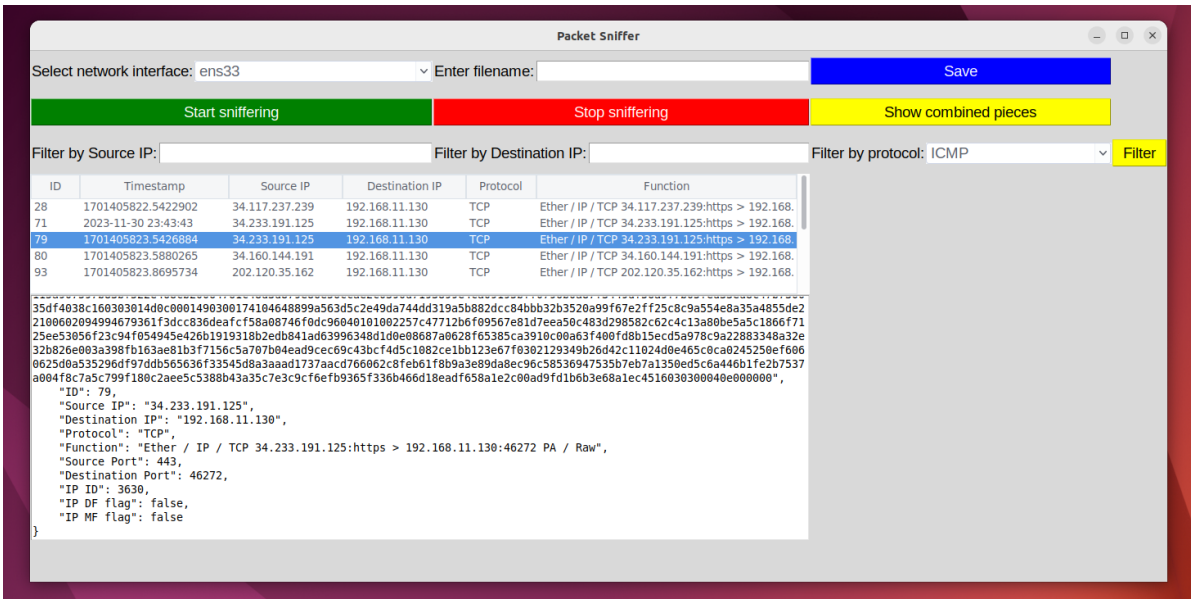
- 根据协议类型进行过滤



3.3 显示重组后数据报

点击Show combined pieces显示重组后的数据报



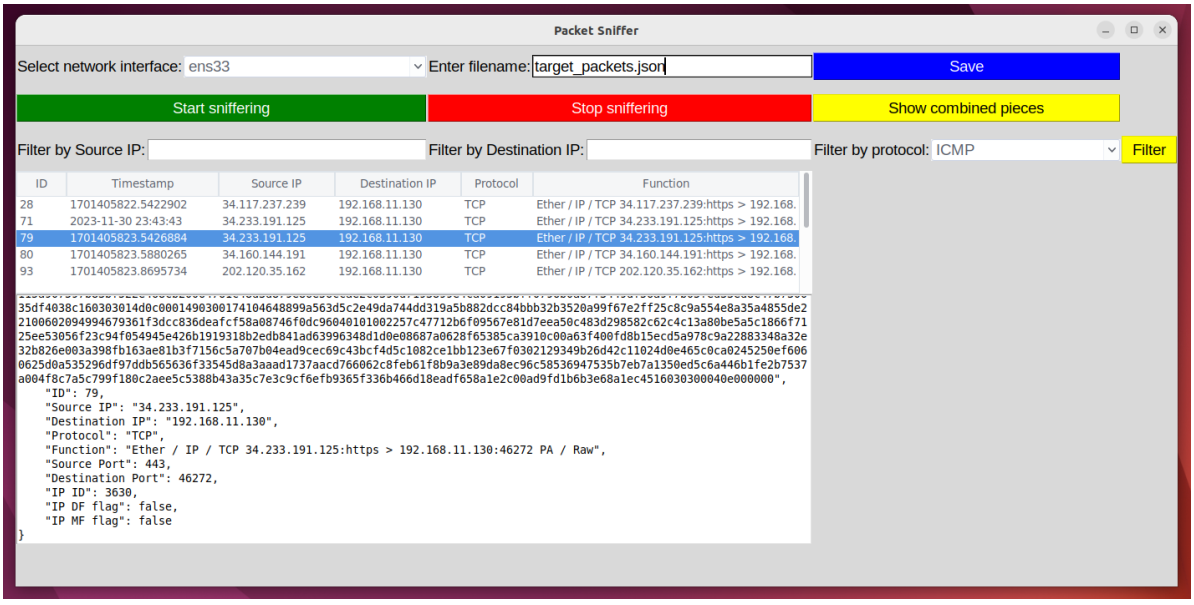


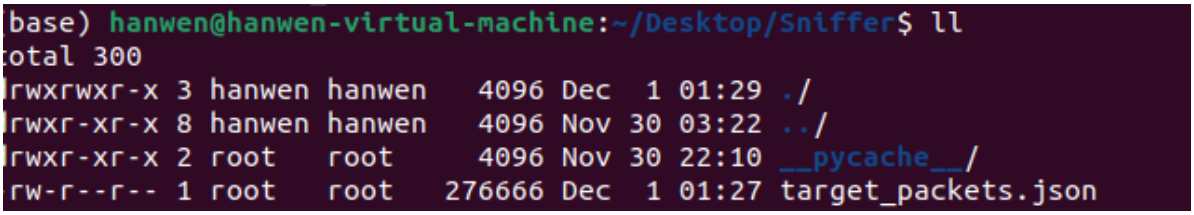
这里再次进行说明，本项目判断数据报是否已重组的依据是数据报的长度和DF字段的值。对于负载部分的具体数据信息不做处理。

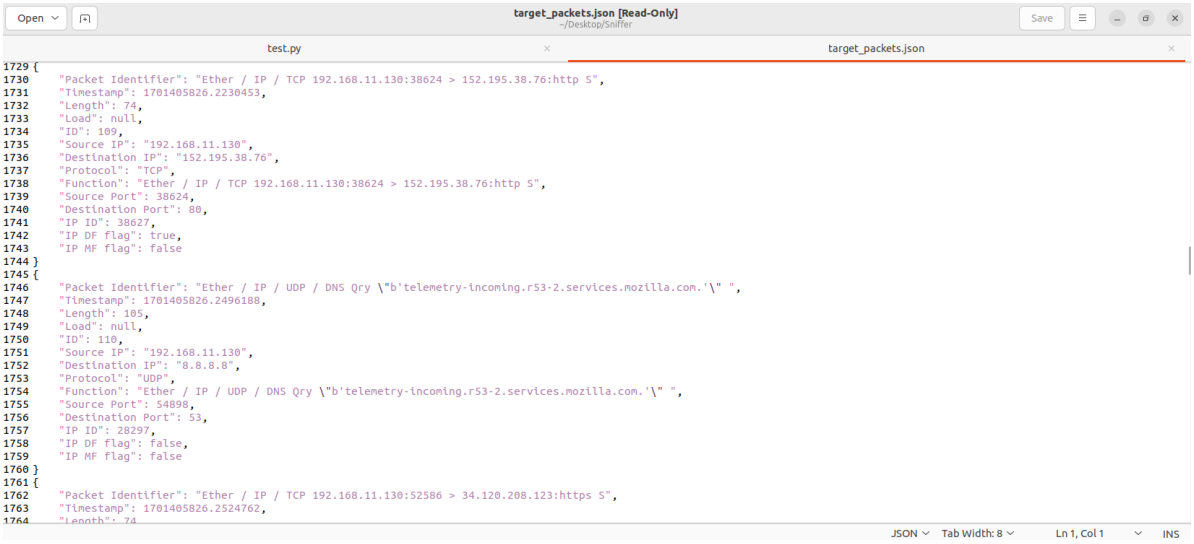
3.4 保存数据报信息

在文件名输入框内输入文件名后，点击Save按钮即可将捕获的数据报信息保存。**建议将文件保存为json格式，这样可读性更强**

```
def save_packets():
    global packets
    global filename
    if filename.get() == "":
        messagebox.showerror("Error", "Please enter a filename.")        # 文本框空异常检测
        return
    with open(filename.get(), "w") as log_file:
        for packet in packets:
            json.dump(packet, log_file, indent=4)                        # 格式化存储保证可读性
            log_file.write("\n")
    packets = []
```








```
{
    "Packet Identifier": "Ether / IP / TCP 192.168.11.130:38624 > 152.195.38.76:http S",
    "Timestamp": 1701405826.2230453,
    "Length": 74,
    "Load": null,
    "ID": 109,
    "Source IP": "192.168.11.130",
    "Destination IP": "152.195.38.76",
    "Protocol": "TCP",
    "Function": "Ether / IP / TCP 192.168.11.130:38624 > 152.195.38.76:http S",
    "Source Port": 38624,
    "Destination Port": 80,
    "IP ID": 38627,
    "IP DF flag": true,
    "IP MF flag": false
}
{
    "Packet Identifier": "Ether / IP / UDP / DNS Qry \"b'telemetry-incoming.r53-2.services.mozilla.com.'\" ",
    "Timestamp": 1701405826.2496188,
    "Length": 105,
    "Load": null,
    "ID": 110,
    "Source IP": "192.168.11.130",
    "Destination IP": "8.8.8.8",
    "Protocol": "UDP",
    "Function": "Ether / IP / UDP / DNS Qry \"b'telemetry-incoming.r53-2.services.mozilla.com.'\" ",
    "Source Port": 54898,
    "Destination Port": 53,
    "IP ID": 28297,
    "IP DF flag": false,
    "IP MF flag": false
}
{
    "Packet Identifier": "Ether / IP / TCP 192.168.11.130:52586 > 34.120.208.123:https S",
    "Timestamp": 1701405826.2524762,
    "Length": 74
```

4. 实例演示

4.1 进行ping请求

```
ping www.bilibili.com
```

1. 向DNS服务器发送域名解析请求

1	2023-12-01 02:07:39	192.168.11.130	8.8.8.8	UDP	Ether / IP / UDP / DNS Qry "b'www.baidu.com.'"

```
{
    "Packet Identifier": "Ether / IP / UDP / DNS Qry \"b'www.baidu.com.'\" ",
    "Timestamp": "2023-12-01 01:55:36",
    "Length": 84,
    "Load": null,
    "ID": 2,
    "Source IP": "192.168.11.130",
    "Destination IP": "8.8.8.8",
    "Protocol": "UDP",
    "Function": "Ether / IP / UDP / DNS Qry \"b'www.baidu.com.'\" ",
    "Source Port": 39193,
    "Destination Port": 53,
    "IP ID": 42768,
    "IP DF flag": false,
    "IP MF flag": false
}
```

2. DNS服务器回复请求

2	1701414459.6469376	8.8.8.8	192.168.11.130	UDP	Ether / IP / UDP / DNS Ans "b'www.a.shifen.com.'"

```
{
    "Packet Identifier": "Ether / IP / UDP / DNS Ans \"b'www.a.shifen.com.'\" ",
    "Timestamp": "2023-12-01 02:01:46",
    "Length": 167,
    "Load": null,
    "ID": 2,
    "Source IP": "8.8.8.8",
    "Destination IP": "192.168.11.130",
    "Protocol": "UDP",
    "Function": "Ether / IP / UDP / DNS Ans \"b'www.a.shifen.com.'\" ",
    "Source Port": 53,
    "Destination Port": 50278,
    "IP ID": 15262,
    "IP DF flag": false,
    "IP MF flag": false
}
```

3. 向解析出的IP发送ICMP报文

3	2023-12-01 02:01:46	192.168.11.130	153.3.238.102	ICMP	Ether / IP / ICMP 192.168.11.130 > 153.3.238.102 echo-request 0 / Raw

```
{
    "Packet Identifier": "Ether / IP / ICMP 192.168.11.130 > 153.3.238.102 echo-request 0 / Raw",
    "Timestamp": "2023-12-01 02:01:46",
    "Length": 98,
    "Load": "da8469650000000073d30e0000000001011121314151617181 91a1b1c1d1e1f2021222324252627282 92a2b2c2d2e2f3031323334353637",
    "ID": 3,
    "Source IP": "192.168.11.130",
    "Destination IP": "153.3.238.102",
    "Protocol": "ICMP",
    "Function": "Ether / IP / ICMP 192.168.11.130 > 153.3.238.102 echo-request 0 / Raw",
    "Source Port": null,
    "Destination Port": null,
    "IP ID": 49916,
    "IP DF flag": true,
    "IP MF flag": false
}
```


4. 收到ICMP回复报文

4	1701414459.6804123	220.181.38.149	192.168.11.130	ICMP	Ether / IP / ICMP 220.181.38.149 > 192.168.11.130 echo-reply 0 / Raw


```
{
    "Packet Identifier": "Ether / IP / ICMP 220.181.38.149 > 192.168.11.130 echo-reply 0 / Raw",
    "Timestamp": "2023-12-01 02:07:39",
    "Length": 98,
    "Load": "3b866965000000005aec0900000000001011121314151617181 91a1b1c1d1e1f2021222324252627282 92a2b2c2d2e2f3031323334353637",
    "ID": 4,
    "Source IP": "220.181.38.149",
    "Destination IP": "192.168.11.130",
    "Protocol": "ICMP",
    "Function": "Ether / IP / ICMP 220.181.38.149 > 192.168.11.130 echo-reply 0 / Raw",
    "Source Port": null,
    "Destination Port": null,
    "IP ID": 15294,
    "IP DF flag": false,
    "IP MF flag": false
}
```


4.2 访问网站（https://pypi.tuna.tsinghua.edu.cn/simple/）

1. 解析DNS

50	1701415029.808506	192.168.11.130	8.8.8.8	UDP	Ether / IP / UDP / DNS Qry "b'pypi.tuna.tsinghua.edu.cn.'"


```
{
    "Packet Identifier": "Ether / IP / UDP / DNS Qry \"b'pypi.tuna.tsinghua.edu.cn.'\" ",
    "Timestamp": "2023-12-01 02:17:09",
    "Length": 85,
    "Load": null,
    "ID": 50,
    "Source IP": "192.168.11.130",
    "Destination IP": "8.8.8.8",
    "Protocol": "UDP",
    "Function": "Ether / IP / UDP / DNS Qry \"b'pypi.tuna.tsinghua.edu.cn.'\" ",
    "Source Port": 59965,
    "Destination Port": 53,
    "IP ID": 37013,
    "IP DF flag": false,
    "IP MF flag": false
}
```


51	1701415029.8433263	8.8.8.8	192.168.11.130	UDP	Ether / IP / UDP / DNS Ans "b'mirrors.tuna.tsinghua.edu.cn.'"
52	1701415029.846605	192.168.11.130	101.6.15.130	TCP	Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https S
53	1701415029.8773563	101.6.15.130	192.168.11.130	TCP	Ether / IP / TCP 101.6.15.130:https > 192.168.11.130:58328 SA / Padding
54	2023-12-01 02:17:09	192.168.11.130	101.6.15.130	TCP	Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https A
55	1701415029.878948	192.168.11.130	101.6.15.130	TCP	Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https PA / Raw


```
{
    "Packet Identifier": "Ether / IP / UDP / DNS Ans \"b'mirrors.tuna.tsinghua.edu.cn.'\" ",
    "Timestamp": "2023-12-01 02:17:09",
    "Length": 162,
    "Load": null,
    "ID": 51,
    "Source IP": "8.8.8.8",
    "Destination IP": "192.168.11.130",
    "Protocol": "UDP",
    "Function": "Ether / IP / UDP / DNS Ans \"b'mirrors.tuna.tsinghua.edu.cn.'\" ",
    "Source Port": 53,
    "Destination Port": 56029,
    "IP ID": 28784,
    "IP DF flag": false,
    "IP MF flag": false
}
```


2. 进行三次握手

（1）客户端发送连接申请

52	1701415029.846605	192.168.11.130	101.6.15.130	TCP	Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https S
53	1701415029.8773563	101.6.15.130	192.168.11.130	TCP	Ether / IP / TCP 101.6.15.130:https > 192.168.11.130:58328 SA / Padding
54	2023-12-01 02:17:09	192.168.11.130	101.6.15.130	TCP	Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https A
55	1701415029.878948	192.168.11.130	101.6.15.130	TCP	Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https PA / Raw


```
{
    "Packet Identifier": "Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https S",
    "Timestamp": "2023-12-01 02:17:09",
    "Length": 74,
    "Load": null,
    "ID": 52,
    "Source IP": "192.168.11.130",
    "Destination IP": "101.6.15.130",
    "Protocol": "TCP",
    "Function": "Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https S",
    "Source Port": 58328,
    "Destination Port": 443,
    "IP ID": 32053,
    "IP DF flag": true,
    "IP MF flag": false
}
```


（2）服务器同意连接

53	1701415029.8773563	101.6.15.130	192.168.11.130	TCP	Ether / IP / TCP 101.6.15.130:https > 192.168.11.130:58328 SA / Padding
54	2023-12-01 02:17:09	192.168.11.130	101.6.15.130	TCP	Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https A
55	1701415029.878948	192.168.11.130	101.6.15.130	TCP	Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https PA / Raw


```
{
    "Packet Identifier": "Ether / IP / TCP 101.6.15.130:https > 192.168.11.130:58328 SA / Padding",
    "Timestamp": "2023-12-01 02:17:09",
    "Length": 60,
    "Load": null,
    "ID": 53,
    "Source IP": "101.6.15.130",
    "Destination IP": "192.168.11.130",
    "Protocol": "TCP",
    "Function": "Ether / IP / TCP 101.6.15.130:https > 192.168.11.130:58328 SA / Padding",
    "Source Port": 443,
    "Destination Port": 58328,
    "IP ID": 28786,
    "IP DF flag": false,
    "IP MF flag": false
}
```


（3）客户端发送确认

54	2023-12-01 02:17:09	192.168.11.130	101.6.15.130	TCP	Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https A
55	1701415029.878948	192.168.11.130	101.6.15.130	TCP	Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https PA / Raw


```
{
    "Packet Identifier": "Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https A",
    "Timestamp": "2023-12-01 02:17:09",
    "Length": 54,
    "Load": null,
    "ID": 54,
    "Source IP": "192.168.11.130",
    "Destination IP": "101.6.15.130",
    "Protocol": "TCP",
    "Function": "Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https A",
    "Source Port": 58328,
    "Destination Port": 443,
    "IP ID": 32054,
    "IP DF flag": true,
    "IP MF flag": false
}
```


3. 传输一些数据

55	1701415029.878948	192.168.11.130	101.6.15.130	TCP	Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https PA / Raw
56	2023-12-01 02:17:09	101.6.15.130	192.168.11.130	TCP	Ether / IP / TCP 101.6.15.130:https > 192.168.11.130:58328 A / Padding
57	1701415029.9108853	101.6.15.130	192.168.11.130	TCP	Ether / IP / TCP 101.6.15.130:https > 192.168.11.130:58328 PA / Raw


```
a32753d95a6dc458d2f66d3fc7854eae7c881111aafab83c7315c6d5e491179863cbbfd992408186126a181ab63eebba7387770308d22cc6999e82f130b7ff9639f5df9706d19026d99d18
e4fda43b3ab395df86f053a239f79f4e4bf09f2ca8d222610666c69c4f273deceff5d1271066427a5059f22181054dcc6adf315cefbfdb3c38ec7759edf6f0f7ad9a0e111264e3c9581477
de20aa17c6df4be7c7f1277ee33ed291c433cfbb2d8c77e7efebe666ab157a350bda2b3ad082ee80d8f1d7a9f98ee3a78aa7a6173cfdc396f328df5d0be5a8066150522d50c80bc3d37a9d
5a66e726e047ccbcaa8b6e19453816253b814eae30029013b01060100a6f14c088375382a34c25cd8445ae770c5451ed8d4baf30cfb6558376de20cf830331f90d27b7e70bc060e36ce25c
d2c6cf1800b1ca12d12ead5d563526899ab19c7be7ea26a0c739893397ad3f92ea009028eaa854ffd86a173d332f84c863f49d9a915a7baab5d84804487bae81c484573ea33aec1530dfdf
0862835629736112980e9de0c5a8297ce96083dcc7efb8d1c568be72c90fb93055eae60a867107ad670490bd5e8e6cb5b76288957abc49958a7ac4203b052afccbaf28dc78ebc1e385ce47
d0707f4f37bdb0e49a3e3271127e290fad6c66687d42a317415fd2e1dbf6efa63d623b364525b88422e046bbc28959fea78b25b4a636b1a155712b806c76ec200313072d564a187d8abad7
53d1cbb1ca9d02f45309b42d5d69f93d3909176eae3d7731119e7ab94ca3ea909d3d47777725d9b",
    "ID": 55,
    "Source IP": "192.168.11.130",
    "Destination IP": "101.6.15.130",
    "Protocol": "TCP",
    "Function": "Ether / IP / TCP 192.168.11.130:58328 > 101.6.15.130:https PA / Raw",
    "Source Port": 58328,
    "Destination Port": 443,
    "IP ID": 32055,
    "IP DF flag": true,
    "IP MF flag": false
}
```

57	1701415029.9108853	101.6.15.130	192.168.11.130	TCP	Ether / IP / TCP 101.6.15.130:https > 192.168.11.130:58328 PA / Raw


```
{
    "Packet Identifier": "Ether / IP / TCP 101.6.15.130:https > 192.168.11.130:58328 PA / Raw",
    "Timestamp": "2023-12-01 02:17:09",
    "Length": 308,
    "Load": "160303008002000007c0303222c1679ba77f5b7e7ff4d6b5ccb07dc094cfcd6f5afba0251cbe6ae43515511206e0f4e53de8ed4a0682bf89ffb807d89a7ee26e27dc1c1d90
e86cade1265c5851302000034002b0002030400330024001d00209956f0dc3df37f3001c8e62193401d5a6d8e777ab7d0852d7f767d845ed8f0740029000200001403030001011703030002
44c41046522dc6b74756849ec199db0281d471cf9fee0ee5061447fc261c2c622d845bb0517030300454162563794588ce565795a348d10da1d049291094358173cb862fd793fe6ed322bc
1e79db574091b5a9a8721d6f59aa872025822e49c8812a4e408fc029fa357427fccf253",
    "ID": 57,
    "Source IP": "101.6.15.130",
    "Destination IP": "192.168.11.130",
    "Protocol": "TCP",
    "Function": "Ether / IP / TCP 101.6.15.130:https > 192.168.11.130:58328 PA / Raw",
    "Source Port": 443,
    "Destination Port": 58328,
    "IP ID": 28788,
    "IP DF flag": false,
    "IP MF flag": false
}
```


4.3 播放视频（www.bilibili.com）

1. 解析DNS

120	1701419205.6236968	192.168.11.130	8.8.8.8	UDP	Ether / IP / UDP / DNS Qry "b'cn-sh-fx-01-05.bilivideo.com.'"
121	1701419205.6249027	192.168.11.130	8.8.8.8	UDP	Ether / IP / UDP / DNS Qry "b'cn-sh-fx-01-03.bilivideo.com.'"
122	1701419205.6568964	8.8.8.8	192.168.11.130	UDP	Ether / IP / UDP / DNS Ans "202.120.193.6"
123	1701419205.6583712	8.8.8.8	192.168.11.130	UDP	Ether / IP / UDP / DNS Ans "2405:d900:e000:8::70"
124	1701419205.6583765	8.8.8.8	192.168.11.130	UDP	Ether / IP / UDP / DNS Ans "202.120.193.4"

这里解析了多个域名，可能是因为在播放视频时涉及不止一个服务器提供服务

2. 第一次握手

1701419205.6601243	192.168.11.130	202.120.193.6	TCP	Ether / IP / TCP 192.168.11.130:44674 > 202.120.193.6:https S
1701419205.6604266	192.168.11.130	202.120.193.6	TCP	Ether / IP / TCP 192.168.11.130:44684 > 202.120.193.6:https S
1701419205.662499	8.8.8.8	192.168.11.130	UDP	Ether / IP / UDP / DNS Ans "2405:d900:e000:8::68"
1701419205.6636417	192.168.11.130	202.120.193.4	TCP	Ether / IP / TCP 192.168.11.130:56480 > 202.120.193.4:https S
1701419205.6638618	192.168.11.130	202.120.193.4	TCP	Ether / IP / TCP 192.168.11.130:56494 > 202.120.193.4:https S

3. 第二次握手

130	1701419205.6640434	202.120.193.6	192.168.11.130	TCP	Ether / IP / TCP 202.120.193.6:https > 192.168.11.130:44684 SA / Padding
131	1701419205.6640484	202.120.193.6	192.168.11.130	TCP	Ether / IP / TCP 202.120.193.6:https > 192.168.11.130:44674 SA / Padding

4. 第三次握手

132	1701419205.6640918	192.168.11.130	202.120.193.6	TCP	Ether / IP / TCP 192.168.11.130:44684 > 202.120.193.6:https A
133	1701419205.6641433	192.168.11.130	202.120.193.6	TCP	Ether / IP / TCP 192.168.11.130:44674 > 202.120.193.6:https A

5. 连接建立后开始传输数据

1701419205.665278	192.168.11.130	202.120.193.6	TCP	Ether / IP / TCP 192.168.11.130:44684 > 202.120.193.6:https PA / Raw
1701419205.6654441	202.120.193.6	192.168.11.130	TCP	Ether / IP / TCP 202.120.193.6:https > 192.168.11.130:44684 A / Padding
1701419205.6661944	192.168.11.130	202.120.193.6	TCP	Ether / IP / TCP 192.168.11.130:44674 > 202.120.193.6:https PA / Raw