计算机网络技术第二次作业
——IP 包流量分析程序

姓名：李伟　学号：2120210454　专业：计算机科学与技术

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，主要分为网络流量观察工具、网络性能测试工具、应用程序性能测量工具三类。IP 包流量是衡量网络状态的重要指标，因此编写 IP 包流量分析程序是学习和掌握网络性能分析工具的基础。

本实验基于原始套接字模式编程实现一个简单的 IP 数据包捕获与分析的程序，基于 PyQT5 实现图形界面可视化，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，深入理解 IP 协议的工作原理以及各字段的作用。

关键字：　IP 数据报　原始套接字　PyQT5　网络编程　数据报捕获　流量分析

目录

一、 作业要求

实现一个 IP 报文分组捕获以及流量分析程序，具体要求如下：

(1) windows 平台上，基于原始套接字模式，图形用户界面，编程语言不限；
(2) 输入捕获条件（IP 地址与时间段），输出 IP 分组主要字段（版本，协议，源地址和目的地址），IP 流量结果统计可视化（按应用层协议与时间段)。
(3) 撰写说明文档，包括编程环境、关键问题、程序流程、测试截图等。
(4) 提交全部程序，包括源代码、可执行程序、说明文档等。

二、 实验环境

本实验的目的是捕获本地网络中的 IP 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。**注意：本实验只对 IP 数据包进行捕获**。

本实验中基于**原始套接字模式**捕获 IP 数据包，可视化界面使用 **Python**、**PyQT5** 搭建，实验操作系统环境为 **windows 10** 专业版。实验相关代码已经托管在 Github 仓库中，`https://github.com/Jack-Lio/IPWatch`.

三、 相关背景知识

3.1 IP 数据包格式

IP 是 Internet Protocol（网际互连协议）的缩写，是 TCP/IP 体系中的网络层协议，提供最基本的数据传输服务。在 TCP/IP 协议中，使用 IP 协议传输数据的包被称为 IP 数据包，每个数据包都包含 IP 协议规定的内容，IP 协议规定的这些内容被称为 IP 数据报文（IP Datagram）或 IP 数据报。IP 分组的结构如图 1 所示。RFC791 是最早的 IP 协议的文本，它对 IP 分组结构做出了明确的规定。IP 数据报是一个与硬件无关的虚拟包，由首部和数据两部分组成。首部的前一部分是固定长度，共 20 字节，是所有 IP 数据报必须具有的。在首部的固定部分的后面是一些可选字段，其长度是可变的。首部中的源地址和目的地址都是 IP 协议地址。

图 1　IP 报文分组结构

IP 分组结构的各个字段含义如下所示：

(1) 版本： 占 4 位（bit），指 IP 协议的版本号。目前的主要版本为 IPV4，即第 4 版本号，也有一些教育网和科研机构在使用 IPV6。在进行通信时，通信双方的 IP 协议版本号必须一致，否则无法直接通信。

(2) 报头长度： 占 4 位（bit），指 IP 报文头的长度。最大的长度（即 4 个 bit 都为 1 时）为 15 个长度单位，每个长度单位为 4 字节（TCP/IP 标准，DoubleWord），所以 IP 协议报文头的最大长度为 60 个字节，最短为上图所示的 20 个字节。

(3) 服务类型： 占 8 位（bit），用来获得更好的服务。其中的前 3 位表示报文的优先级，后面的几位分别表示要求更低时延、更高的吞吐量、更高的可靠性、更低的路由代价等。对应位为 1 即有相应要求，为 0 则不要求。

(4) 总长度： 16 位（bit），指报文的总长度。注意这里的单位为字节，而不是 4 字节，所以一个 IP 报文最大长度为 216 - 1=65535 字节。

(5) 标识 (identification)： 该字段标记当前分片为第几个分片，在数据报重组时很有用。

(6) 标志（flag)： 该字段用于标记该报文是否为分片（有一些可能不需要分片，或不希望分片），后面是否还有分片（是否是最后一个分片）。

(7) 片偏移： 指当前分片在原数据报（分片前的数据报）中相对于用户数据字段的偏移量，即在原数据报中的相对位置。

(8) 生存时间： TTL（Time to Live）。该字段表明当前报文还能生存多久。每经过 1ms 或者一个网关，TTL 的值自动减 1，当生存时间为 0 时，报文将被认为目的主机不可到达而丢弃。使用过 Ping 命令的用户应该有印象，在 windows 中输入 ping 命令，在

返回的结果中即有 TTL 的数值。

(9) 协议：该字段指出在上层（网络 7 层结构或 TCP/IP 的传输层）使用的协议，可能的协议有 UDP、TCP、ICMP、IGMP、IGP 等。

(10) 首部校验和：用于检验 IP 报文头部在传播的过程中是否出错，主要校验报文头中是否有某一个或几个 bit 被污染或修改了。

(11) 源 IP 地址： 32 位（bit），4 个字节，每一个字节为 0 ~ 255 之间的整数，及我们日常见到的 IP 地址格式。

(12) 目的 IP 地址： 32 位（bit），4 个字节，每一个字节为 0 ~ 255 之间的整数，及我们日常见到的 IP 地址格式。

3.2 TCP 数据包格式

TCP（Transmission Control Protocol 传输控制协议）是一种面向连接的、可靠的、基于字节流的传输层通信协议，由 IETF 的 RFC 793 定义。TCP 封装在 IP 报文中的时候，TCP 头（其结构如图 2 所示）紧接着 IP 头，不携带选项 (option) 的 TCP 头长为 20bytes，携带选项的 TCP 头最长可到 60bytes。

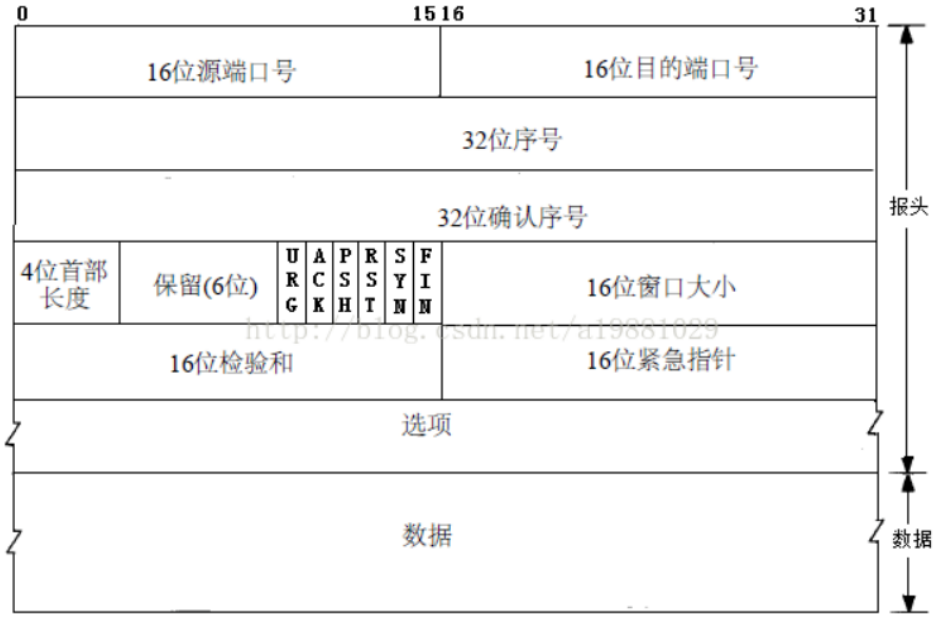

图 2　TCP 报文分组结构

(1) 源端口：16 位的源端口其中包含发送方应用程序对应的端口。源端口和源 IP 地址标示报文发送端的地址。

(2) 目的端口： 16 位的目的端口域定义传输的目的。这个端口指明报文接收计算机上的应用程序地址接口。

(3) TCP 序列号：数据序号，32bits，TCP 连接中传送的数据流中的每一个字节都编上一个序号。序号字段的值则指的是本报文段所发送的数据的第一个字节的序号。

(4) TCP 应答号：确认号，32bits，期望收到对方的下一个报文段的数据的第一个字节的序号。

(5) 头部长度： 4 位包括 TCP 头大小，指示 TCP 头的长度。

(6) 保留字： 4 位值域，这些位必须是 0。为了将来定义新的用途所保留，其中 RFC3540 将 Reserved 字段中的最后一位定义为 Nonce 标志。

(7) 标志： 8 位标志位，做 TCP 包具有特殊功能的标识。

(8) 窗口大小： 16bits，窗口字段用来控制对方发送的数据量，单位为字节。TCP 连接的一端根据设置的缓存空间大小确定自己的接收窗口大小，然后通知对方以确定对方的发送窗口的上限。

(9) 校验和： 16bits，检验和字段检验的范围包括首部和数据这两部分。在计算检验和时，要在 TCP 报文段的前面加上 12 字节的伪首部。

(10) 紧急指针： 16bits，紧急指针指出在本报文段中的紧急数据的最后一个字节的序号。

(11) 选项：长度可变。TCP 首部可以有多达 40 字节的可选信息，用于把附加信息传递给终点，或用来对齐其它选项。这部分最多包含 40 字节，因为 TCP 头部最长是 60 字节（其中还包含前面讨论的 20 字节的固定部分）有的 TCP 选项没有后面两个字段，仅包含 1 字节的 kind 字段。第二个字段 length（如果有的话）指定该选项的总长度，该长度包括 kind 字段和 length 字段占据的 2 字节。第三个字段 info（如果有的话）是选项的具体信息. kind=0 是选项表结束选项 TCP 模块通常将 MSS 设置为（MTU-40）字节（减掉的这 40 字节包括 20 字节的 TCP 头部和 20 字节的 IP 头部）。这样携带 TCP 报文段的 IP 数据报的长度就不会超过 MTU（假设 TCP 头部和 IP 头部都不包含选项字段，并且这也是一般情况），从而避免本机发生 IP 分片。对以太网而言，MSS 值是 1460（1500-40）字节。

3.3 UDP 数据包格式

UDP（User Datagram Protocol 用户数据报协议）和 TCP 都属于传输层协议。UDP 协议的主要作用是将网络数据流量压缩成数据包的形式。一个典型的数据包就是一个二进制数据的传输单位。每一个数据包（其结构如表 2.3 所示）的前 8 个字节用来包含报头信息，剩余字节则用来包含具体的传输数据。

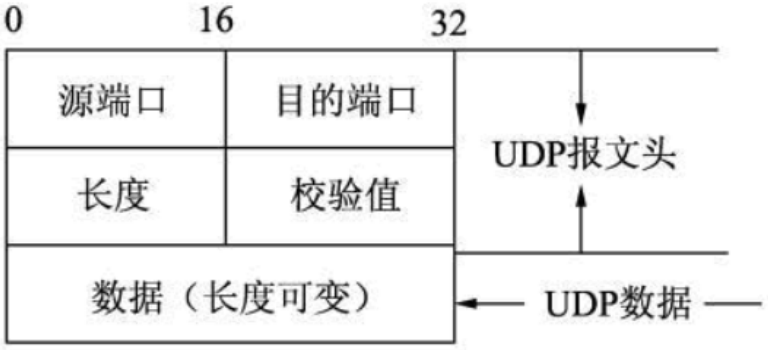

图 3　UDP 报文结构

(1) 源端口：这个字段占据 UDP 报文头的前 16 位，通常包含发送数据报的应用程序所使用的 UDP 端口。接收端的应用程序利用这个字段的值作为发送响应的目的地址。这个字段是可选的，所以发送端的应用程序不一定会把自己的端口号写入该字段中。如果不写入端口号，则把这个字段设置为 0。这样，接收端的应用程序就不能发送响应了。

(2) 目的端口： 16 位的目的端口域定义传输的目的。这个端口指明报文接收计算机上的应用程序地址接口。

(3) 封包长度：该字段占据 16 位，表示 UDP 数据报长度，包含 UDP 报文头和 UDP 数据长度。因为 UDP 报文头长度是 8 个字节，所以这个值最小为 8。

(4) 校验和：该字段占据 16 位，可以检验数据在传输过程中是否被损坏。

四、 程序设计思路和关键实现步骤说明

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，前者依靠原始套接字进行数据包捕获，后者基于 PyQT5 的可视化界面制作框架完成。使用原始套接字进行数据包捕获一般需要三个主要步骤，包括获取主机地址（设备），创建 IP 数据包捕获的原始套接字并绑定任意端口，创建数据包捕获和分析线程进行数据包的捕获和分析。

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 IP 数据报 (**默认情况下，本实验程序绑定第一个可用网卡接口，因此如果有多个网卡的情况，请保留需要捕捉 IP 报文的网卡，并禁用其他网卡！**)，然后对于每一个收到的 IP 报文，创建一个对应的数据分析线程函数，对收到的数据报进行解析和校验和的计算，收到的数据包经过分析后的结果保存在一个缓冲队列中交由前端界面进行展示，前端界面根据用户的操作和交互选择展示相应的分析结果和数据报文信息。

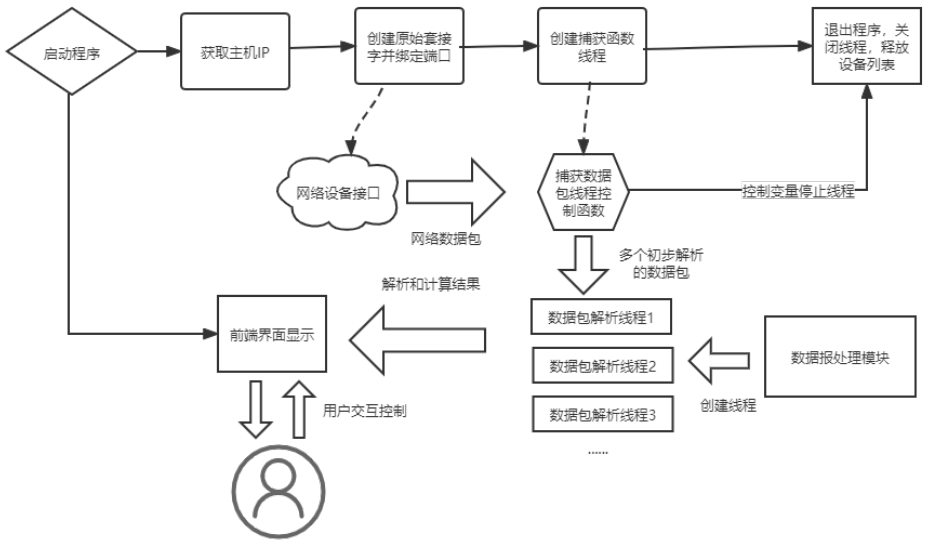

图 4　程序设计整体结构

4.1 获取主机 IP 地址

首先利用 socket 中的 gethostname() 函数获取本机的 IP 地址。然后通过 socke 的 gethostbyname() 方法获取主机的 IP 地址。然后就可以通过绑定该 IP 地址创建原始套接字进行数据包捕获了。

```

# func.py
def get_ip_address(hostname=None):
    '''
    Get the IP address of the hostname.
    hostname: the hostname of the target.
    '''
    assert(hostname is not None)
    try:
        ip_address = socket.gethostbyname(hostname)
    except socket.gaierror:
        ip_address = 'Unknown'
    return ip_address

# ui.py
def set_ip_capture_device(self):
```

```
    # get the device
    self.ip_address = func.get_ip_address(socket.gethostname())
    # get the row socket and bind it to the host address
    # self.device = socket.socket(socket.AF_INET, socket.SOCK_RAW,
        socket.IPPROTO_IP)
    self.device = socket.socket(socket.AF_INET, socket.SOCK_RAW,
        socket.IPPROTO_IP)
    # default bind to eth0 network adapter
    self.device.bind((self.ip_address,0))
    # Include IP headers
    self.device.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
    # set the socket to receive all the packages
    self.device.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)
```

4.2 打开网络接口

在获得主机 IP 地址之后，可以通过创建原始套接字进行数据包捕获，注意 socke.socket() 函数中设置参数为 socket.IPPROTO_IP 捕捉所有的 IP 报文，并通过设 socket.SIO_RCVALL 和 socket.RCVALL_ON 参数开启混合模式接受所有的数据报文。同时注意进行参数设置接受报文时包含 IP 头部。

```
# ui.py
def set_ip_capture_device(self):
    # get the device
    self.ip_address = func.get_ip_address(socket.gethostname())
    # get the row socket and bind it to the host address
    # self.device = socket.socket(socket.AF_INET, socket.SOCK_RAW,
        socket.IPPROTO_IP)
    self.device = socket.socket(socket.AF_INET, socket.SOCK_RAW,
        socket.IPPROTO_IP)
    # default bind to eth0 network adapter
    self.device.bind((self.ip_address,0))
    # Include IP headers
    self.device.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
    # set the socket to receive all the packages
    self.device.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)
```

4.3 在打开的网络接口上进行数据报捕获

本实验主要建立了一个报文捕获的主类，通过这个类完成对于报文捕获和分析的主要工作，类定义如下所示；

```
# func.py
class ThreadWorker(threading.Thread):
    '''
    define the method for multi-thread to parse the raw package.
    '''
    def __init__(self, ts, data):
        super(ThreadWorker, self).__init__()
        self.result = None
        self.ts = ts
        self.data = data

    def run(self):
        self.result = parse_raw_package(self.ts, self.data)

    def get_result(self):
        return self.result

class MainIPCaptureWorker(threading.Thread):
    def __init__(self, device, ip_address, event):
        '''
        device: the device of the target.
        ip_address: the ip address of the target.
        event: the event to be set when the task is done.
        '''
        super(MainIPCaptureWorker, self).__init__()
        self.device = device
        self.ip_address = ip_address
        self.event = event
        self.task_list = []
```

```
30
31      def run(self):
32          capture_packet(self.device, self.ip_address, self.event,
                self.task_list)
33
34      def get_task_list(self):
35          return self.task_list
36
37      def get_result(self):
38          return [worker.get_result() for worker in self.task_list]
```

在该类中，主要接受一个 device（原始套接字对象）进行数据报文的捕获，在运行过程中创建一个报文捕获主线程，同时通过 event 进行线程执行的控制，将捕获的报文保存在对应的线程对象中，然后通过 get_result() 方法获取捕获的所有报文。

数据包捕获工作线程如下所示：

```
1  # func.py
2  def capture_packet(device, ip_address, event, task_list):
3      '''
4      Capture the packet of the target.
5      device: the device of the target.
6      ip_address: the IP address of the target.
7      event: the event of the target.
8      task_list: the list of the task.
9      '''
10     assert(device is not None)
11     assert(ip_address is not None)
12     assert(event is not None)
13     assert(task_list is not None)
14     while True:
15         if event.is_set():
16             ts = time.time() # Get the timestamp of the package.
17             try:
18                 packet = device.recvfrom(65535)
19                 print(len(task_list))
20                 task = ThreadWorker(ts, packet[0])
```

```
                task_list.append(task) # add the task to the list
                task.setDaemon(True)  # if the Daemon is set to True, the
                    thread will be terminated when the main thread is
                    terminated
                task.start()          # start the thread and begin to
                    parse the raw package
            except socket.timeout:
                print(time.strftime('%Y-%m-%d %H:%M:%S',
                    (time.localtime(ts))),'No data captured')
        else:
            return
```

该线程不断进行报文捕获，将捕获的报文传送给一个数据报文分析线程进行数据分析，然后将结果保存在分析线程对象的成员变量中，所有的分析线程对象保存在 task_list 中。

分析线程的设计如下所示：

```
# func.py
def parse_raw_package(ts, data):
    '''
    Parse the raw package.
    ts: the timestamp of the package.
    data: the data of the package.
    '''
    # print(time.strftime('%Y-%m-%d %H:%M:%S',
        (time.localtime(ts))),data)
    packet = data
    ip_header = packet[0:20]
    from struct import unpack # unpack the package

    # Unpack the IP header
    iph = unpack('!BBHHHBBH4s4s', ip_header)
    version = (iph[0] & 0xF0) >> 4 # Version
    # ihl = (iph[0] & 0x0F) << 2 # IHL
    # total_length = iph[2] # Total Length
    # ttl = iph[5] # TTL
    protocol = iph[6]    # protocol type
```

```
    s_addr = socket.inet_ntoa(iph[8])
    d_addr = socket.inet_ntoa(iph[9])

    if protocol == 6:
        # Unpack the TCP header
        tcp_header = packet[20:40]
        tcph = unpack('!HHLLBBHHH', tcp_header)
        source_port = tcph[0]         # Source Port
        dest_port = tcph[1]           # Destination Port
        # sequence = tcph[2]            # Sequence Number
        # acknowledgement = tcph[3]     # Acknowledgement Number
        # doff_reserved = tcph[4]       # Data Offset, Reserved
        # tcph_length = doff_reserved >> 4 # TCP header length
        # length = total_length - 20    # Data length
    elif protocol == 17:
        # Unpack the UDP header
        udp_header = packet[20:28]
        udph = unpack('!HHHH', udp_header)
        source_port = udph[0]         # Source Port
        dest_port = udph[1]           # Destination Port
        # length = udph[2]              # Length
        # checksum = udph[3]            # Checksum
        # service determine by the port, if the src or dst port is not in
            the dictionary, then it is 'Others'

    else :
        source_port = 'Unknown'
        dest_port = 'Unknown'

    service = get_service_by_port(dest_port)
    if service == 'Others':
        service = get_service_by_port(source_port)
    return time.strftime('%Y-%m-%d %H:%M:%S', (time.localtime(ts))),
        s_addr, d_addr, str(source_port), str(dest_port)
        ,get_protocal(protocol), service, "IPv"+str(version)
```

该分析线程函数主要对 IP 头部报文进行分析，截取报文的目的地址，源地址，协议类型等信息，并根据协议类型进行进一步的分析，将 TCP 和 UDP 报文进行进一步的分析，截取目的端口和源端口信息，并根据端口号推测其服务的上层应用层协议类型，将所有的信息结果进行返回。返回的结果最终被数据包捕获主类的 get_results() 函数从 task_list 列表中提取出来，交由前端进行展示。

对于数据报文的应用层数据包统计分析，本实验根据常用的会话层协议类型和应用层协议类型进行了基本统计，通过分类和统计的方式进行了初步的统计和分析，分析统计代码如下所示：

```
# func.py
service_dictionary = {
        20: 'FTP',
        21: 'FTP',
        22: 'SSH',
        25: 'SMTP',
        53: 'DNS',
        80: 'HTTP',
        110: 'POP3',
        443: 'HTTPS',
        1900: 'SSDP',
    }

service_list = ['FTP', 'SSH', 'SMTP', 'DNS', 'HTTP', 'POP3', 'HTTPS',
    'SSDP', 'Others']

protocol_dictionary = {
        1: 'ICMP', # Internet Control Message Protocol
        2: 'IGMP', # Internet Group Management Protocol
        3: 'GGP',  # Gateway-to-Gateway Protocol
        4: 'IPv4', # IPv4 encapsulation
        5: 'ST',   # ST datagram mode
        6: 'TCP',  # Transmission Control Protocol
        8: 'EGP',  # Exterior Gateway Protocol
        9: 'IGP',  # Interior Gateway Protocol
        17: 'UDP', # UDP protocol
        41: 'IPv6', # IPv6 encapsulation
```

```
        89: 'OSPF', # Open Shortest Path First
    }

protocol_list = ['ICMP', 'IGMP', 'GGP', 'IPv4', 'ST', 'TCP', 'EGP',
    'IGP', 'UDP', 'IPv6', 'OSPF', 'Others']

def get_protocal(id):
    '''
    Get the protocal of the port.
    '''

    try:
        protocol = protocol_dictionary[id]
    except KeyError:
        protocol = 'Others'
    return protocol

def get_service_by_port(port):
    '''
    Get the service by the port.
    port: the port of the target.
    '''
    assert(port is not None)

    try:
        service = service_dictionary[port]
    except KeyError:
        service = 'Others'
    return service

def packet_count_func(results):
    '''
    Count the number of the packet.
    results: the list of the results.
    '''
```

```
    assert(results is not None)
    total_count = 0
    server_dic = {}
    protocol_dic = {}
    for result in results:
        total_count += 1
        if result[5] not in protocol_dic.keys():
            protocol_dic[result[5]] = 1
        else:
            protocol_dic[result[5]] += 1
        if result[5] not in {'TCP', 'UDP'}: # For application layer, only
            count the tcp and udp packets
            continue
        if  result[6] not in server_dic.keys():
            server_dic[result[6]] = 1
        else:
            server_dic[result[6]] += 1
    return total_count, protocol_dic, server_dic
```

在该部分，我们主要对 SMTP，TCP，UDP 等会话层常用协议和 HTTP，HTTPs，POP3 等常用应用层协议进行统计分析，根据每一类别数据报文占中报文数量的多少可以分析各类报文在网络中的使用情况，获取基本的网络报文使用信息。

4.4 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、数据包统计显示弹窗三个主要模块。具体页面效果如下：

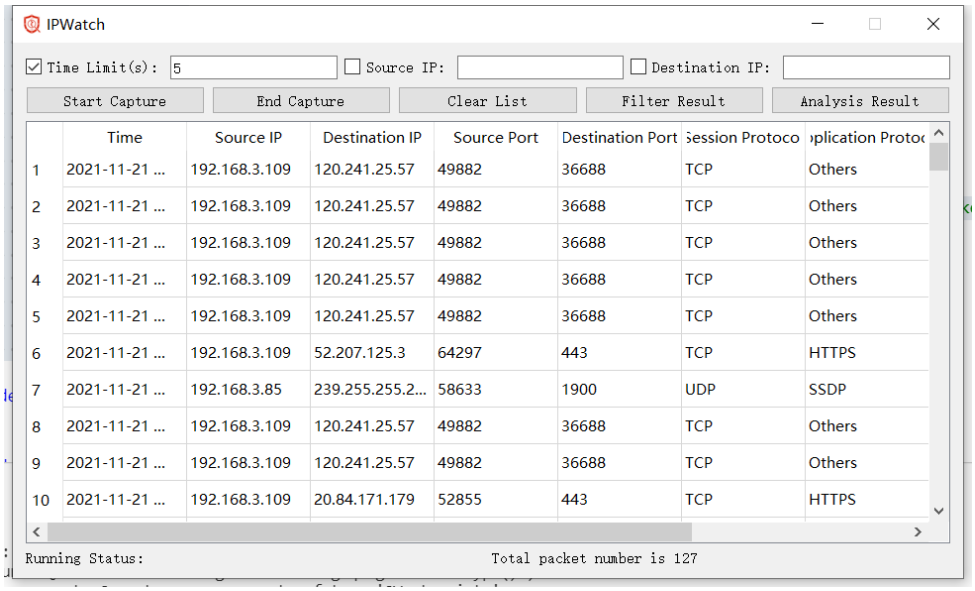

图 5　界面显示效果

当用户启动程序之后，将会扫描本机本机 IP 地址，在界面上方勾选相应的报文筛选条件并填写条件之后，点击开始捕获报文按钮将会进行报文捕获并在最下方的状态栏提示捕捉报文的持续时间。最终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报文捕获个数。点击报文分析按钮可以获取报文的统计分析结果如下。

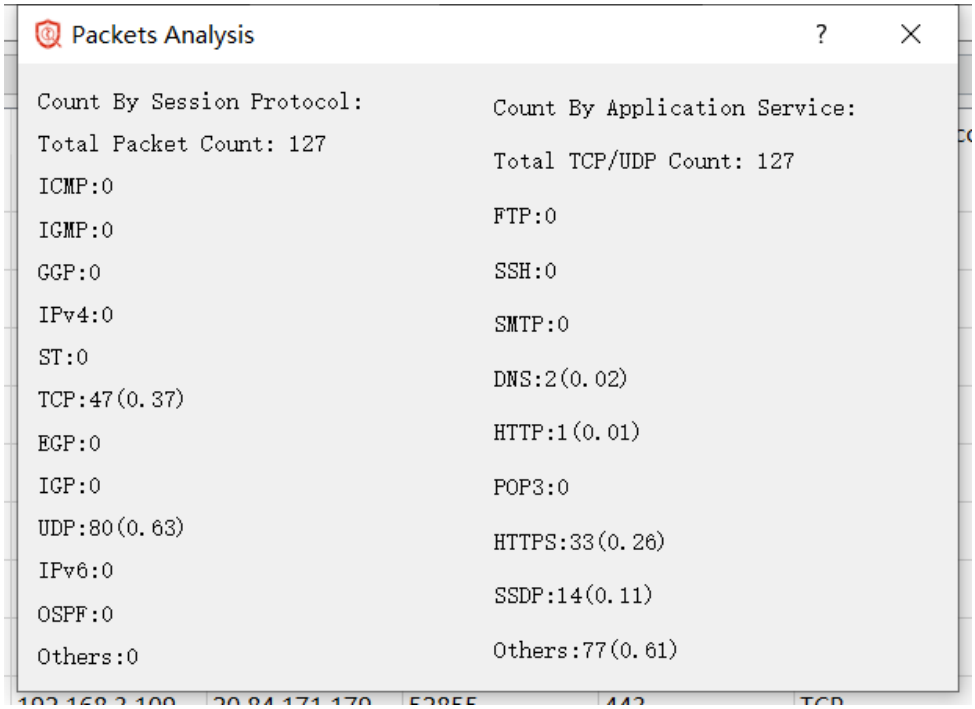

图 6　报文统计数据结果展示

五、 使用手册和结果展示

报文捕获：点击开始捕获报文进行报文数据捕获

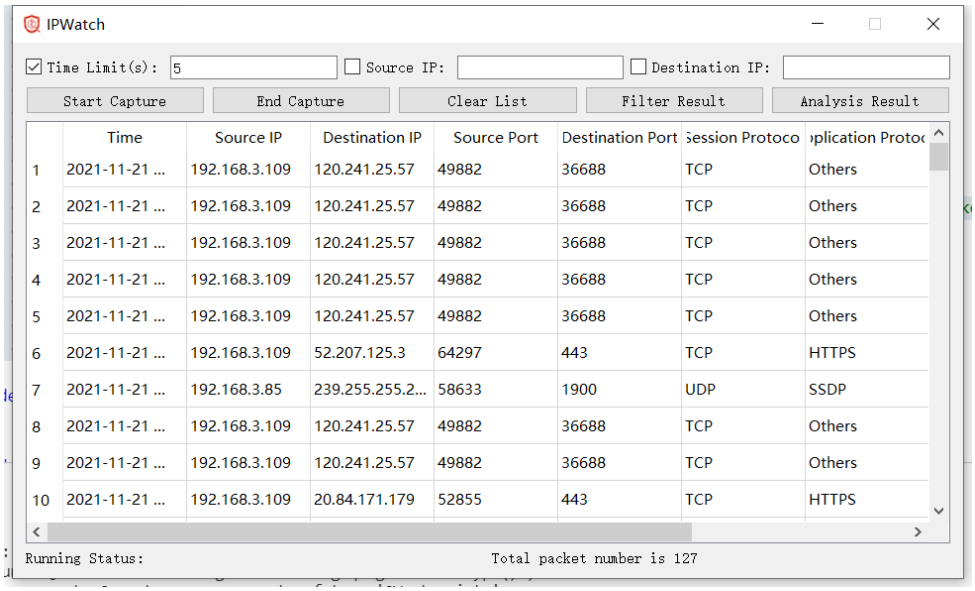

图 7　界面显示效果

报文结果统计信息展示：点击报文分析按钮进行结果展示

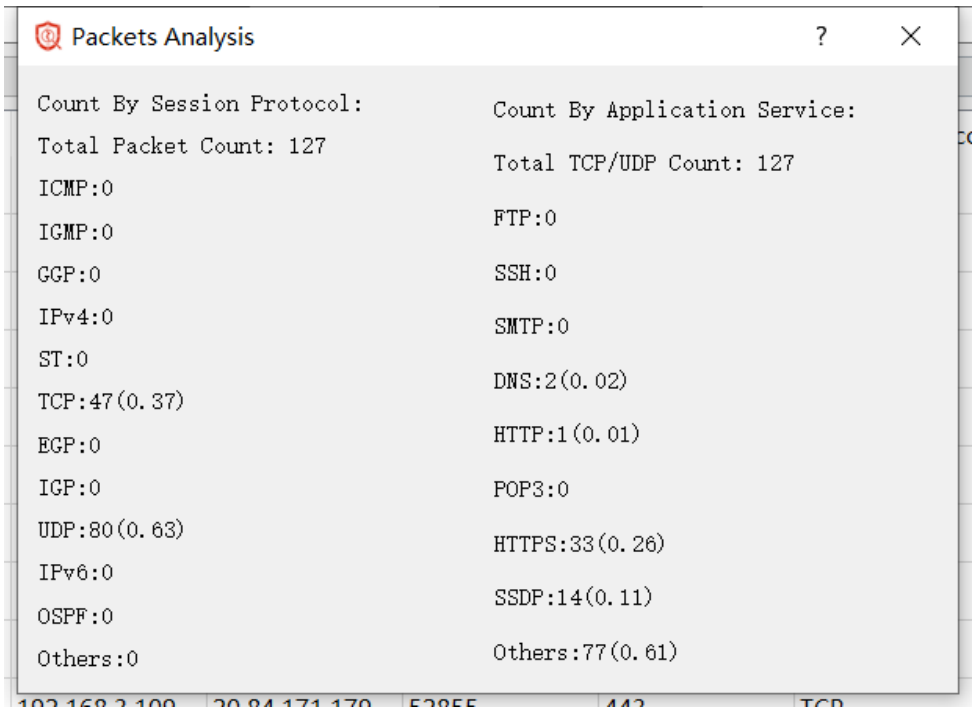

图 8　报文统计数据结果展示

报文按照各类信息排序：在表格中点击相应信息的表头根据该表头类型进行报文信息排序

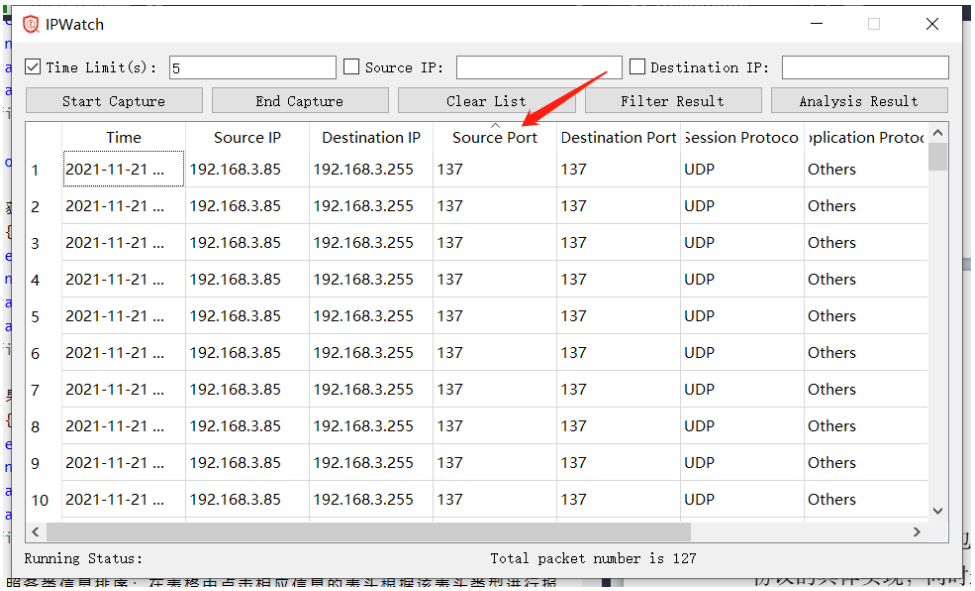

图 9　报文信息排序

报文筛选：通过勾选相应的筛选条件并填写条件，点击筛选按钮即可根据条件筛选符合条件的报文进行展示，在下方提示栏显示筛选剩余报文的总数。

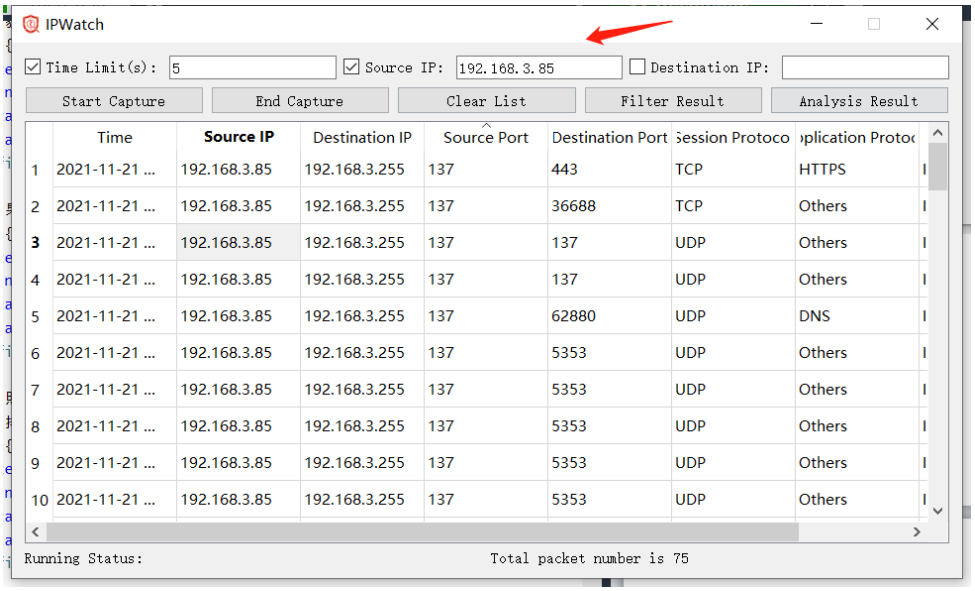

图 10　报文按条件筛选

列表清空：点击清空按钮可以清空所有报文缓存列表

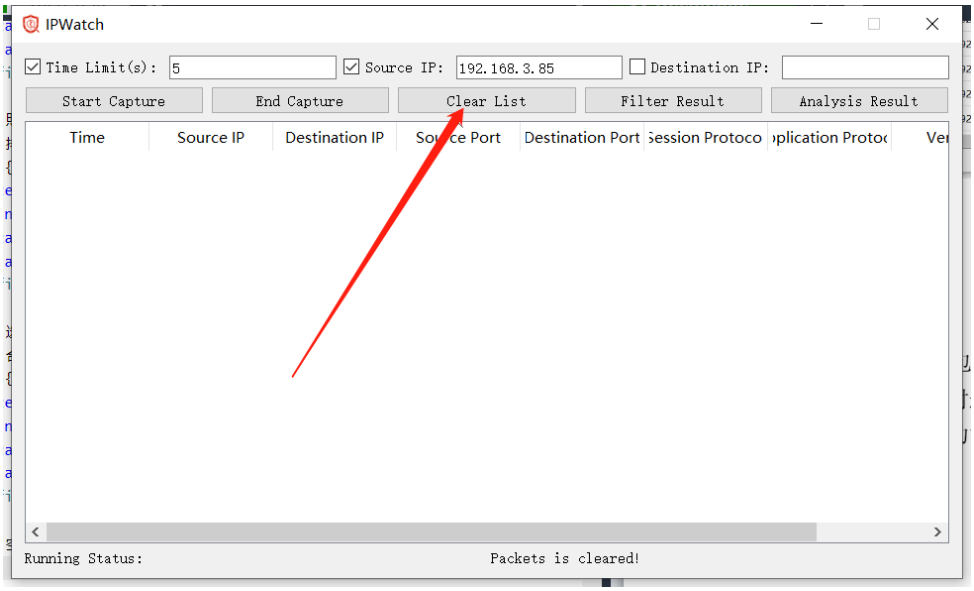

图 11　清空缓存列表

六、 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，比如在进行报文统计的时候，通过常用端口可以有效的进行基本信息的统计，简化了实验设计思路。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。