《网络空间安全与风险管理》
实 验 报 告

学 生 姓 名　曹寅峰

学　　号　16020610025

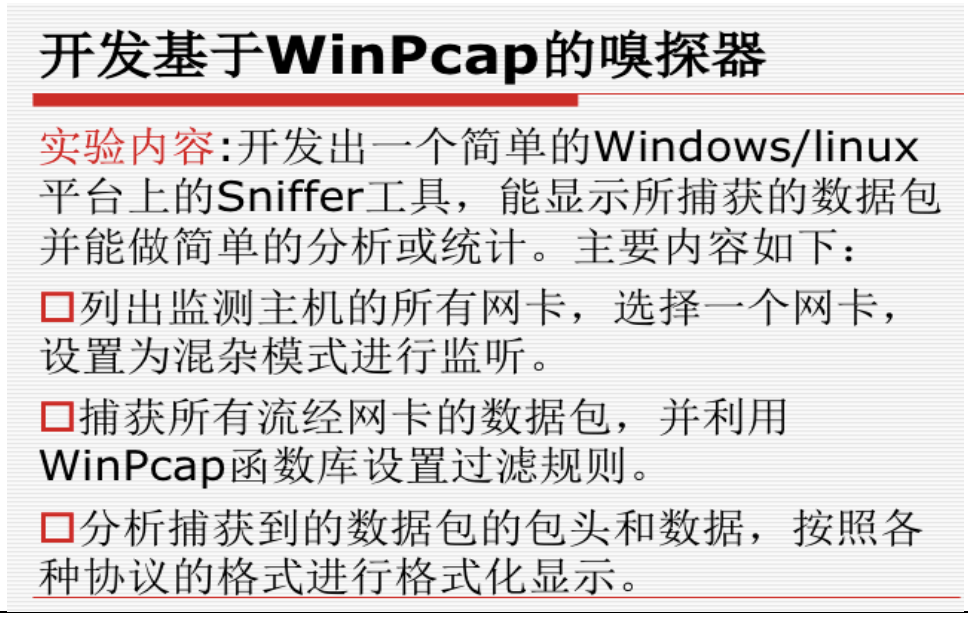
班　　级　1618019

2018 — 2019 学年　第 一 学期

《操作系统原理》实验报告

实验名称	网络嗅探器实验	实验序号	01
实验日期		实验人	曹寅峰

一、实验题目



二、相关原理与知识

WinPcap(Windows Packet Capture)是 Windows 平台下一个免费、开源、公共的网络访问系统，其目的在于为应用程序提供访问网络底层的能力。它由 UNIX 下的 Libpcap 移植而来。

WinPcap 的功能：

1. 捕获原始数据包，包括在共享网络上各主机发送/接收的以及相互之间交换的数据包。
2. 在数据包发往应用程序之前，按照自定义的规则将某些特殊的数据包过滤进行分析。
3. 在网络上发送原始的数据包。
4. 收集网络通信过程中的统计信息。

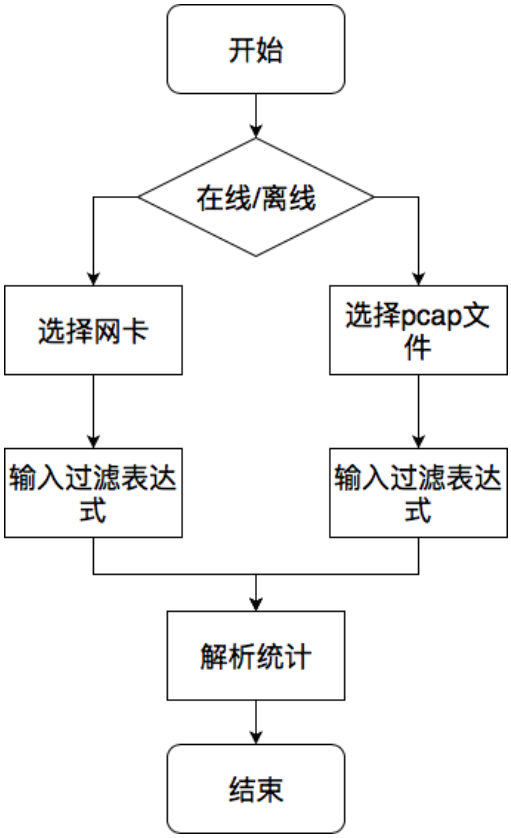
利用 WinPcap 开发嗅探程序的步骤如下：

1. 枚举所有的可用的网络设备，选择一个目标网络设备进行嗅探
2. 打开嗅探设备
3. 设置和编译过滤器
4. 开始捕获数据包
5. 关闭网络设备

在本次试验中，我们在 linux 下进行，使用类似 winpcap 的 libpcap 的 python 版本 pcapy，完成嗅探器的开发工作，实现了全部功能，可以对常见包类型进行过滤，统计，格式化输出，并且**额外拥有离线工作功能**，可以直接分析 pcap 包。

三、实验过程

嗅探器的程序流程图如下



使用 python3.6 实现，主要使用 pcapy 模块进行解析数据包，在解析时使用多线程加快处理速度，并在解析时生成统计信息。

1. 工作模式选择

```
choice=int(input("请输入 : \n1 离线工作模式\n2 在线工作模式\n"))
if choice==1:
    pcapfile=input("请输入 pcap 文件名:\n")
    cap=pcapy.open_offline(pcapfile)
if choice==2:
    devices = pcapy.findalldevs()
    print ("可用网卡:")
    for d in devices :
        print (d)
    dev = input("请输入要监听的网卡:\n")
    print ("正在监听网卡 " + dev)
    cap = pcapy.open_live(dev, 65536 , 1 , 100)
```

在这一部分中，主要完成工作的选择。

如果输入 choice 为 1，进入离线工作模式，使用 pcapy.open_offline(pcapfile)函数来打开一个 pcap 文件；

如果输入 choice 为 2，进入在线工作模式，使用 pcapy.open_live(dev, 65536 , 1 , 100)函数分析实时的 pcap 文件

2. 网卡选择/数据包选择

```
        cap=pcapy.open_offline(pcapfile)
```

如果是离线工作模式，pcapy.open_offline(pcapfile)函数可以直接打开一个本地的 pcap 文件进行分析

```
        devices = pcapy.findalldevs()
        print ("可用网卡:")
        for d in devices :
            print (d)
        dev = input("请输入要监听的网卡:\n")
        print ("正在监听网卡 " + dev)
        cap = pcapy.open_live(dev, 65536 , 1 , 100)
```

如果是在线工作模式，通过 pcapy.findalldevs()列出所有网卡后，由用户选择其中之一进行监听，并且作为 dev 参数传入 pcapy.open_live(dev, 65536 , 1 , 100)中去

3. 过滤表达式

```
    myfilter=input('请输入过滤表达式:\n')
    cap.setfilter(myfilter)
```

对于上面得到的 cap 文件，使用 setfilter(myfilter)方法来设置一个过滤表达式，结果是过滤之后的 cap 文件，内含符合过滤要求的 packet

4. 数据包解析模块

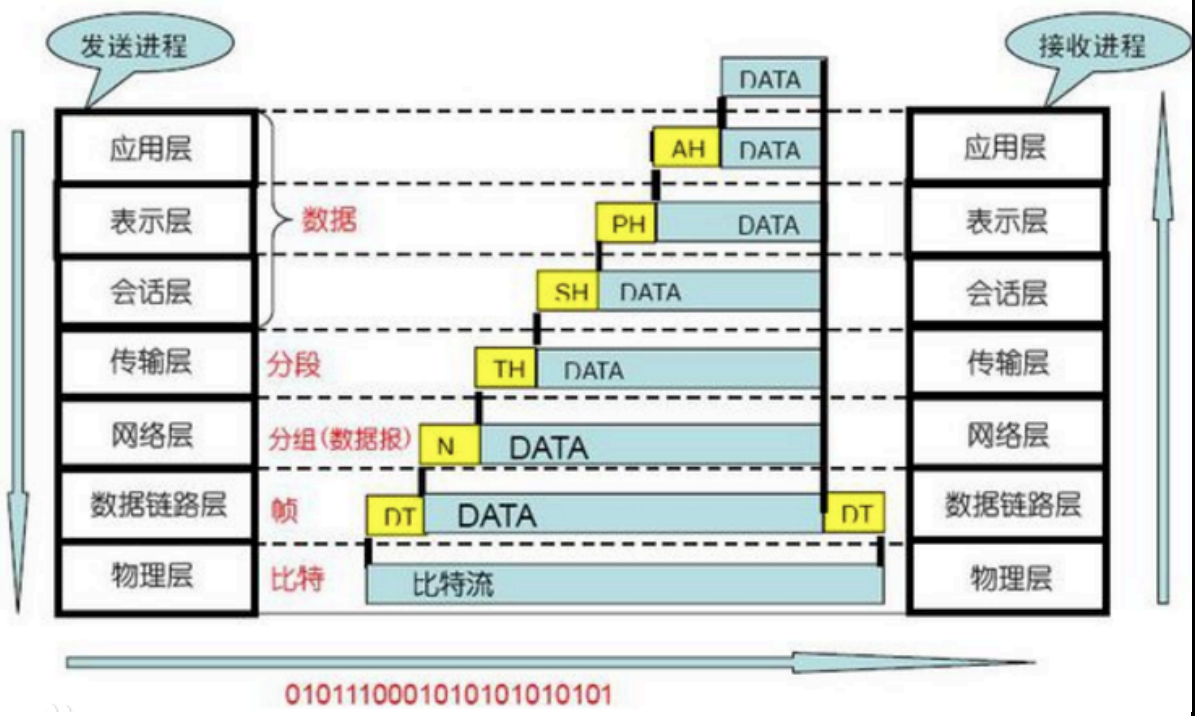
```
    t1=threading.Thread(target=loop,args=(cap,),name='LoopThread1')
    t2=threading.Thread(target=loop,args=(cap,),name='LoopThread2')
    t1.start()
    t2.start()
    t1.join()
    t2.join()
```

在过滤之后，我们设置两个线程进行解析，其中 loop 为主体函数，进行解析

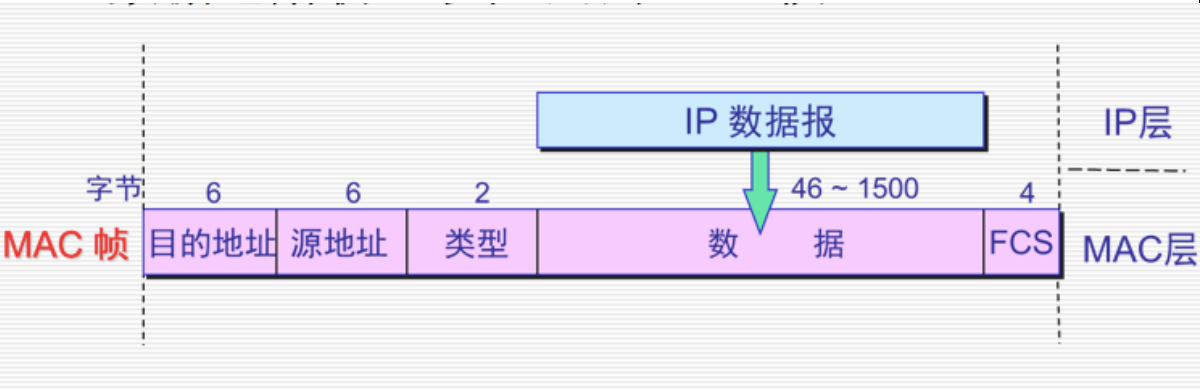
```
def loop(cap):
    print ('thread %s is running...' % threading.current_thread().name)
    packetnum=0
    while(1) :
        (header, packet) = cap.next()
        try:
            print('第 %d 个数据包'%packetnum)
            parse_packet(packet)
            packetnum=packetnum+1
```

在这个循环中，利用 cap.next()函数遍历 pcap 文件中的 packet
在 parse_packet(packet)函数中包含了多种数据包的解析方法，是扫描器的核心所在

1. 以太帧解析



根据 osi 七层模型，我们需要对数据包从下到上进行解析，首先是以太帧。

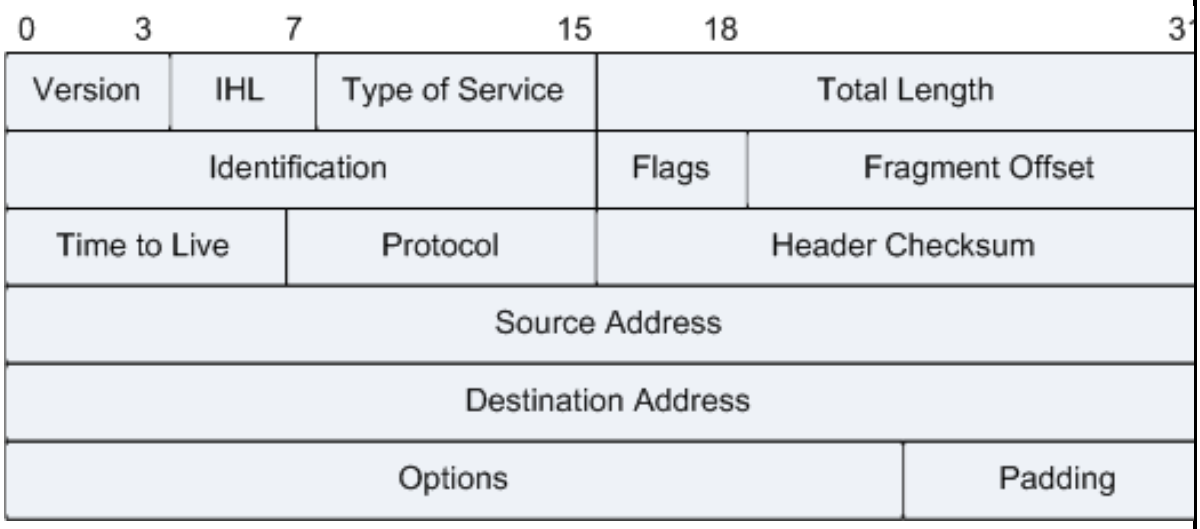


以太帧的格式如上，所以我们需要提取**前 14 个字节**的数据，并根据最后两位判断网络层的协议类型。

```
    eth_length = 14
    eth_header = packet[:eth_length]
    eth = unpack('!6s6sH' , eth_header)
    eth_protocol = socket.ntohs(eth[2])
    print ('目标MAC : ' + eth_addr(packet[0:6]) + ' 源MAC : '
    eth_addr(packet[6:12]) + ' 协议类型 : ' + str(eth_protocol))
```

代码如上，利用 python 的 struct.unpack 进行格式解析，例如!6s6sH 代表按照大段存储，6 个字节字符，6 个字节字符，无符号整型格式化解析。
最终解析出目标 mac，源 mac，协议类型

2. IP 协议解析

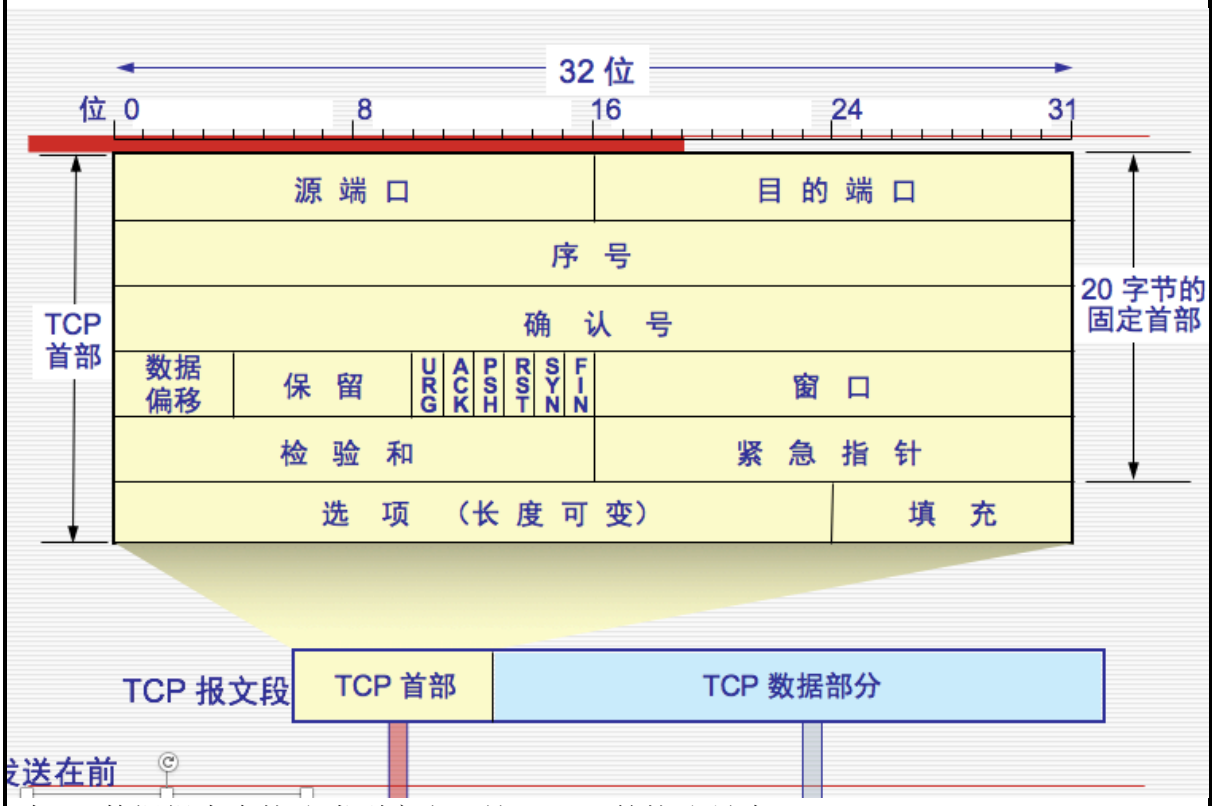


如果是以太帧，那么数据段应该是 IP 协议数据（以太协议号为 8），我们可以采取类似的方法解析 IP 数据报

```
    if eth_protocol == 8 :
        print('IP 报文')
        ip_header = packet[eth_length:20+eth_length]
        iph = unpack('!BBHHHBBH4s4s' , ip_header)
        version_ihl = iph[0]
        version = version_ihl >> 4
        ihl = version_ihl & 0xF
        iph_length = ihl * 4
        ttl = iph[5]
        protocol = iph[6]
        s_addr = socket.inet_ntoa(iph[8])
        d_addr = socket.inet_ntoa(iph[9])
        print ('版本: ' + str(version) + ' IP 首部长度 : ' + str(ihl) + '
TTL : ' + str(ttl) + ' 协议类型 : ' + str(protocol) + ' 源地址 : ' +
str(s_addr) + ' 目的地址 : ' + str(d_addr))
```

这里我们提取前 20 个字节，同样利用 unpack 方法来解析。
最终解析出版本，IP 首部长度，TTL，协议类型（传输层），源地址，目的地址。

3. TCP 报文解析



在 IP 数据报中有协议类型字段，这里 TCP 的协议号为 6

```
if protocol == 6 :
          print('TCP 报文')
          t = iph_length + eth_length
          tcp_header = packet[t:t+20]

          tcph = unpack('!HHLLBBHHH' , tcp_header)

          source_port = tcph[0]
          dest_port = tcph[1]
          sequence = tcph[2]
          acknowledgement = tcph[3]
          doff_reserved = tcph[4]
          tcph_length = doff_reserved >> 4

          print ('源端口 : ' + str(source_port) + ' 目的端口 : ' +
str(dest_port) + ' Sequence Number : ' + str(sequence) + '
Acknowledgement : ' + str(acknowledgement) + ' TCP header length : ' +
str(tcph_length))

          h_size = eth_length + iph_length + tcph_length * 4
          data_size = len(packet) - h_size
```

```
            data = packet[h_size:]

            print ('报文数据: ' + str(data)+'\n')
```

同理，我们提取前 20 字节与豹 TCP 报文数据

解析出源端口，目的端口，序列号，ACK，TCP 头部长度与具体数据

4. UDP

16 位源端口号	16 位目的端口号
16 位 UDP 长度	16 位 UDP 检验和
数据（如果有）	

同理，UDP 的格式比较简单，其协议号为 17

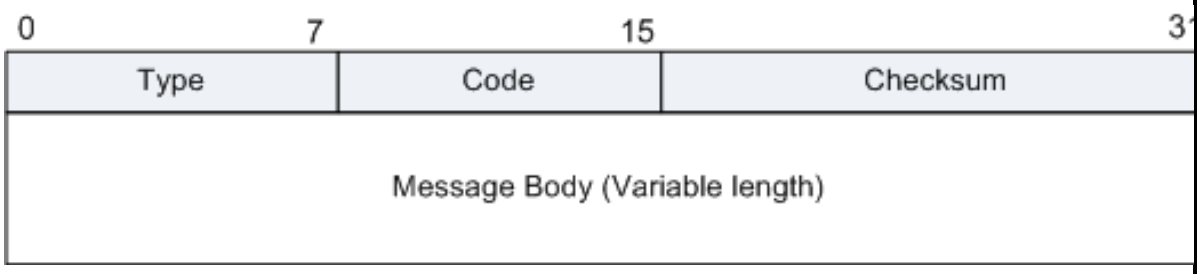
```
        elif protocol == 17 :
            print('UDP 报文')

            u = iph_length + eth_length
            udph_length = 8
            udp_header = packet[u:u+udph_length]

            udph = unpack('!HHHH' , udp_header)

            source_port = udph[0]
            dest_port = udph[1]
            length = udph[2]
            checksum = udph[3]

            print ('源端口 : ' + str(source_port) + ' 目的端口 : ' +
str(dest_port) + ' Length : ' + str(length) + ' Checksum : ' +
str(checksum))

            h_size = eth_length + iph_length + udph_length
            data_size = len(packet) - h_size

            data = packet[h_size:]
```

```
            print ('报文数据: ' + str(data)+'\n')
```

这里我们提取前 8 字节与报文数据

解析出源端口，目的端口，长度，校验值，数据

5. ICMP



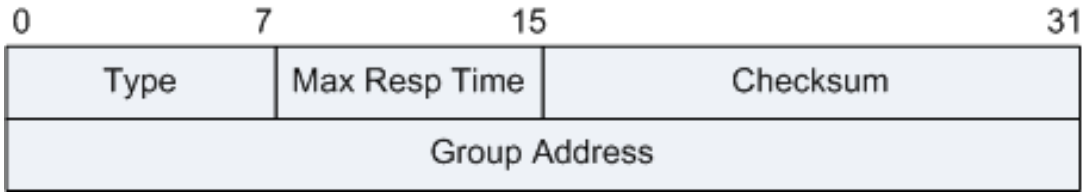
ICMP 协议是一种面向无连接的协议，用于传输出错报告控制信息。它是一个非常重要的协议，它对于网络安全具有极其重要的意义。是网络层的补充，工作在网络层与传输层之间。

```
        elif protocol == 1 :
            print('ICMP 报文')

            u = iph_length + eth_length
            icmph_length = 4
            icmp_header = packet[u:u+icmph_length]

            icmph = unpack('!BBH' , icmp_header)

            icmp_type = icmph[0]
            code = icmph[1]
            checksum = icmph[2]

            print ('类型 : ' + str(icmp_type) + ' Code : ' + str(code) + '
校验值 : ' + str(checksum))

            h_size = eth_length + iph_length + icmph_length
            data_size = len(packet) - h_size

            data = packet[h_size:]

            print ('报文数据: ' + str(data)+'\n')
```

ICMP 协议号为 1，我们提取前 4 个字节头部与报文数据

最终解析出 ICMP 类型，CODE 值，校验值，报文数据。

6. IGMP

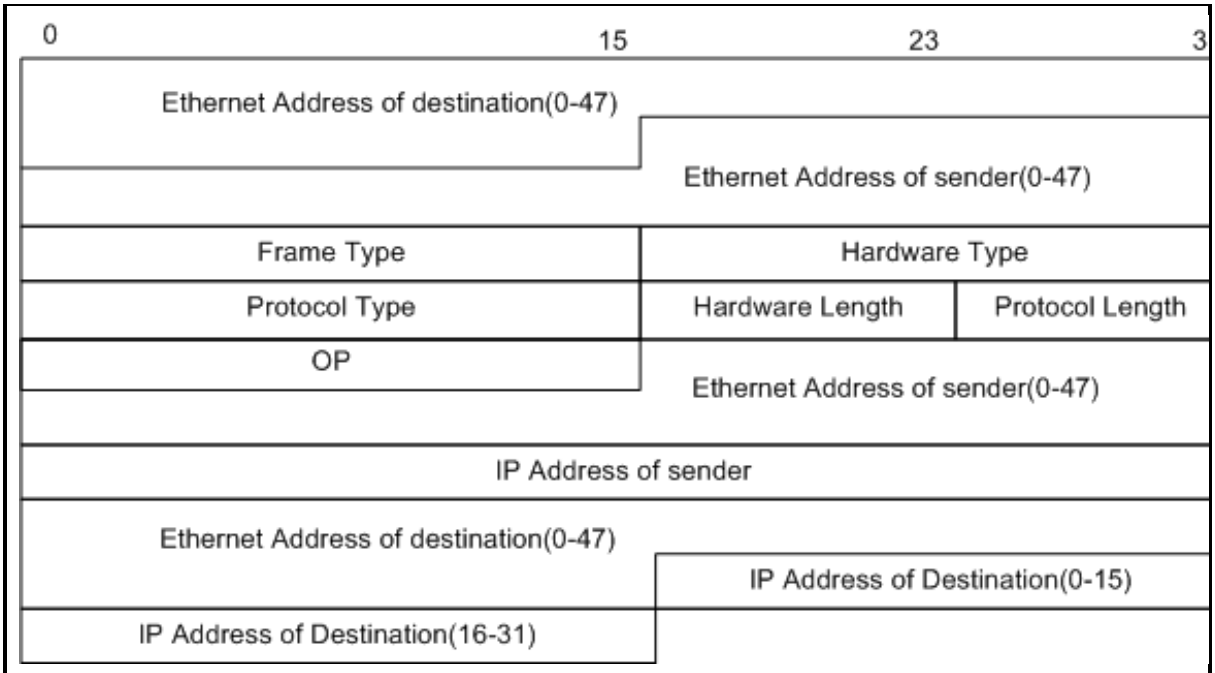



```
        elif protocol==2:

            print('IGMP 报文')
            u = iph_length + eth_length
            igmph_length = 4
            igmp_header = packet[u:u+4]
            igmph = unpack('!BBH' , igmp_header)
            igmp_type = igmph[0]
            maxresptime = igmph[1]
            checksum = igmph[2]

            print ('类型 : ' + str(igmp_type) + ' 最大响应时间 : ' +
str(maxresptime) + ' 校验值 : ' + str(checksum))

            h_size = eth_length + iph_length + igmph_length
            data_size = len(packet) - h_size

            #get data from the packet
            data = packet[h_size:]

            print ('组播地址: ' + data)
```

IGMP 的协议类型为 2，我们提取前 4 字节的首部，剩下的为组播地址
最终提取出类型，最大响应时间，校验值，组播地址

7. ARP/RARP

当以太帧内包含的是 arp 数据报的时候，格式如下




```
        print("arp 报文")
        arp_header=packet[eth_length:eth_length+28]
        arph = unpack('!HHBBH6s4s6s4s' , arp_header)
        op=arph[4]
        e_sender=arph[5]
        ip_sender=arph[6]
        e_dst=arph[7]
        ip_dst=arph[8]
        print("操作："+str(op)+' 发送方以太网地址：'+str(eth_addr(e_sender))+'
发送方的 IP 地址：'+str(socket.inet_ntoa(ip_sender))+' 接收方的以太网地址：
'+str(eth_addr(e_dst))+' 接收方的 IP 地址：
'+str(socket.inet_ntoa(ip_dst))+'\n')
```

提取前 28 字节信息，ARP 和 RARP 主要是 OP 类型不同
最终解析出操作，发送方以太网地址，发送方的 IP 地址，接收方的以太网地址，，接收方的 IP 地址。

以上就是程序的全部主体框架，可以实现设置在线/离线模式，选择特定网卡，选择 pcap 文件，多线程解析，TCP、UDP、ARP、RARP、IGMP、ICMP 的解析功能

四、实验结果与分析

首先可以选择工作模式

```
(base) cyf:风险管理/ $ python ./sniffer.py
请输入 :
1 离线工作模式
2 在线工作模式
```

在线工作模式中选择网卡

```
2
可用网卡:
en0
p2p0
awdl0
bridge0
utun0
en1
lo0
gif0
stf0
XHC20
请输入要监听的网卡:
```

输入过滤表达式

```
正在监听网卡 en0
请输入过滤表达式:
tcp
```

格式化解析 tcp 包，可以看到，首先解析以太帧，然后解析 IP 数据报，最后解析 TCP

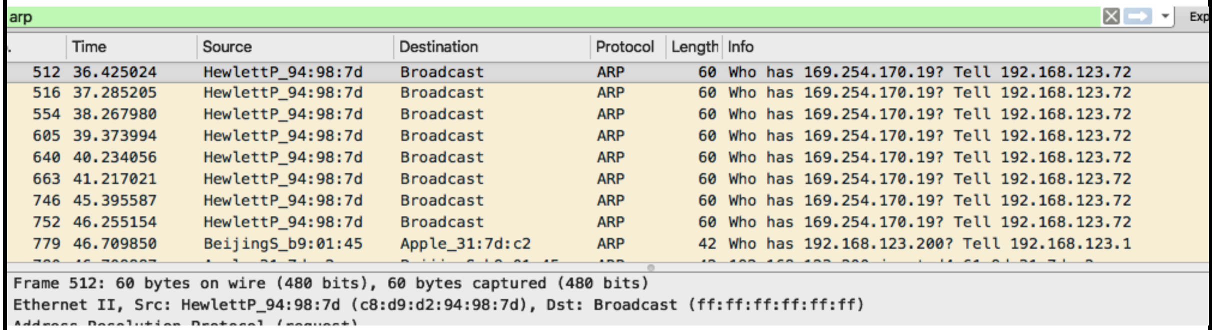
```
目标MAC : d4:61:9d:31:7d:c2 源MAC : 14:3d:f2:b9:01:45 协议类型 : 8
IP报文
版本: 4 IP 首部长度 : 5 TTL : 51 协议类型 : 6 源地址 : 123.151.137.100 目的地址 : 192.168.123.200
TCP报文
源端口 : 80 目的端口 : 57958 Sequence Number : 3102853753 Acknowledgement : 3937374356 TCP header length : 5
报文数据: b'\x00\x00\x00\xb8\x00\x00\x00\r\x01\x00\x00\x00\x00\x0e3242286322\x05w\xbc\xe3\xe8\xf2K\x98\xad{\xe2X\x8b\xf3\x0bq\xbb\xf5\x11\x8d\xa1\x19\'\x87u\xd92\xb7\xcattj\x1a[\xb1L\xab\x9e8\xb9\xd44\xa6\x01\x0c\x0egD\xdd\xac\x1e|\x1f\x1f\xf69\xab\xb2\xfb\xad_\xf4\xcb\xd1\x8e\x87\xfc~\x86A\xf7\xb2V\xfe"\xb5\xf9\xc5\xe4\xcftQ\xed~$\x19\x9e\xa1^w\xbf\x990\xe3\x865\xc1\x1bg~\xfd)b\xc1\xbf\xfe\xdb\xdf\xa8\xeb\x7f\xce\x00\xbbX\xd9\xe8rZ\xbb.\xb6\xdd\xe5\xaf#E\xd5\xc5\xe5\x9c\xf6\x15\x87:b+\x8d\x9e\xd69\x89\xf2v\xa0`k1U\xa2\x895A\x9d\xe7E\xc5\x018\x1d'
```

同样的，我们也可以使用离线工作模式，直接分析一个已经有的 pcap 包

```
1 离线工作模式
2 在线工作模式
1
请输入pcap文件名:
./test2.pcap
请输入过滤表达式:
```

之后的操作是一样的。

解析 ARP 报文



对比 wireshark 中的情况

其他情况不一一列出，经对比与 wireshark 结果一致。

五、问题总结

本次实验成功实现了一个网络嗅探器，拥有在线离线两种工作模式，可以解析常见的多种数据包。

过程中出现过一些问题，比如多线程的使用应该放在解析包时，为了避免混乱要使用线程锁。也要注意各个协议类型的所属关系，要一层一层的解析。

通过这次实验我学习到了嗅探器开发的基本知识，比如嗅探器一般的开发框架为：

1. **选择网卡并抓包** 如 pcapy.findalldevs()　cap =pcapy.open_live(dev, 65536 , 1 , 100)
2. **过滤表达式**　如 cap.setfilter(myfilter)
3. **解析数据包**　strut.unpack()

同时，对于不懂的函数，效率最高的方法就是直接查看官方文档。

六、源代码

```
import socket
from struct import *
import datetime
import pcapy
import sys
import threading
import time

def main(argv):
    choice=int(input("请输入 : \n1 离线工作模式\n2 在线工作模式\n"))
```

```
    if choice==1:
        pcapfile=input("请输入 pcap 文件名:\n")
        cap=pcapy.open_offline(pcapfile)
    if choice==2:
        devices = pcapy.findalldevs()
        print ("可用网卡:")
        for d in devices :
            print (d)
        dev = input("请输入要监听的网卡:\n")
        print ("正在监听网卡 " + dev)
        cap = pcapy.open_live(dev, 65536 , 1 , 100)
    myfilter=input('请输入过滤表达式:\n')
    cap.setfilter(myfilter)
    t1=threading.Thread(target=loop,args=(cap,),name='LoopThread1')
    t2=threading.Thread(target=loop,args=(cap,),name='LoopThread2')
    t1.start()
    t2.start()
    t1.join()
    t2.join()

def loop(cap):
    print ('thread %s is running...' % threading.current_thread().name)
    packetnum=0
    while(1) :
        (header, packet) = cap.next()

        try:
            print('第 %d 个数据包'%packetnum)
            parse_packet(packet)
            packetnum=packetnum+1


        except:
            continue

#转成冒号形式 mac
def eth_addr (a) :
    b = "%.2x:%.2x:%.2x:%.2x:%.2x:%.2x" % ((a[0]) , (a[1]) , (a[2]),
(a[3]), (a[4]) , (a[5]))
    return b

#解析数据包
def parse_packet(packet) :
```

```
    #以太帧解析
    eth_length = 14
    eth_header = packet[:eth_length]
    eth = unpack('!6s6sH' , eth_header)
    eth_protocol = socket.ntohs(eth[2])
    print ('目标MAC : ' + eth_addr(packet[0:6]) + ' 源MAC : ' +
eth_addr(packet[6:12]) + ' 协议类型 : ' + str(eth_protocol))

    #IP包
    if eth_protocol == 8 :
        print('IP报文')

        ip_header = packet[eth_length:20+eth_length]

        iph = unpack('!BBHHHBBH4s4s' , ip_header)

        version_ihl = iph[0]
        version = version_ihl >> 4
        ihl = version_ihl & 0xF

        iph_length = ihl * 4

        ttl = iph[5]
        protocol = iph[6]
        s_addr = socket.inet_ntoa(iph[8])
        d_addr = socket.inet_ntoa(iph[9])

        print ('版本: ' + str(version) + ' IP 首部长度 : ' + str(ihl) + '
TTL : ' + str(ttl) + ' 协议类型 : ' + str(protocol) + ' 源地址 : ' +
str(s_addr) + ' 目的地址 : ' + str(d_addr))

        #TCP
        if protocol == 6 :
            print('TCP报文')
            t = iph_length + eth_length
            tcp_header = packet[t:t+20]

            tcph = unpack('!HHLLBBHHH' , tcp_header)

            source_port = tcph[0]
            dest_port = tcph[1]
            sequence = tcph[2]
            acknowledgement = tcph[3]
            doff_reserved = tcph[4]
```

```
            tcph_length = doff_reserved >> 4

            print ('源端口 : ' + str(source_port) + ' 目的端口 : ' +
str(dest_port) + ' Sequence Number : ' + str(sequence) + '
Acknowledgement : ' + str(acknowledgement) + ' TCP header length : ' +
str(tcph_length))

            h_size = eth_length + iph_length + tcph_length * 4
            data_size = len(packet) - h_size

            data = packet[h_size:]

            print ('报文数据: ' + str(data)+'\n')

        #ICMP
        elif protocol == 1 :
            print('ICMP 报文')

            u = iph_length + eth_length
            icmph_length = 4
            icmp_header = packet[u:u+icmph_length]

            icmph = unpack('!BBH' , icmp_header)

            icmp_type = icmph[0]
            code = icmph[1]
            checksum = icmph[2]

            print ('类型 : ' + str(icmp_type) + ' Code : ' + str(code) + '
校验值 : ' + str(checksum))

            h_size = eth_length + iph_length + icmph_length
            data_size = len(packet) - h_size

            data = packet[h_size:]

            print ('报文数据: ' + str(data)+'\n')

        #UDP
        elif protocol == 17 :
            print('UDP 报文')

            u = iph_length + eth_length
            udph_length = 8
```

```
            udp_header = packet[u:u+udph_length]

            udph = unpack('!HHHH' , udp_header)

            source_port = udph[0]
            dest_port = udph[1]
            length = udph[2]
            checksum = udph[3]

            print ('源端口 : ' + str(source_port) + ' 目的端口 : ' +
str(dest_port) + ' Length : ' + str(length) + ' Checksum : ' +
str(checksum))

            h_size = eth_length + iph_length + udph_length
            data_size = len(packet) - h_size

            data = packet[h_size:]

            print ('报文数据: ' + str(data)+'\n')

        #IGMP
        elif protocol==2:

            print('IGMP 报文')
            u = iph_length + eth_length
            igmph_length = 4
            igmp_header = packet[u:u+4]
            igmph = unpack('!BBH' , igmp_header)
            igmp_type = igmph[0]
            maxresptime = igmph[1]
            checksum = igmph[2]

            print ('类型 : ' + str(igmp_type) + ' 最大响应时间 : ' +
str(maxresptime) + ' 校验值 : ' + str(checksum))

            h_size = eth_length + iph_length + igmph_length
            data_size = len(packet) - h_size

            #get data from the packet
            data = packet[h_size:]

            print ('组播地址: ' + data)

        else :
```

```
            print ('其他协议')
    else:#arp
        print("arp 报文")
        arp_header=packet[eth_length:eth_length+28]
        arph = unpack('!HHBBH6s4s6s4s' , arp_header)
        op=arph[4]
        e_sender=arph[5]
        ip_sender=arph[6]
        e_dst=arph[7]
        ip_dst=arph[8]
        print("操作："+str(op)+' 发送方以太网地址：'+str(eth_addr(e_sender))+'
发送方的 IP 地址：'+str(socket.inet_ntoa(ip_sender))+' 接收方的以太网地址：
'+str(eth_addr(e_dst))+' 接收方的 IP 地址：
'+str(socket.inet_ntoa(ip_dst))+'\n')


if __name__ == "__main__":
    main(sys.argv)
```