

**实验报告**

**实 验（二）**

题 目 捕包软件的使用与实现

专 业 计 算 机 类

学　　 号 1160300620

班　　 级 1603006

学 生 张 恒

实 验 地 点 格物214

实 验 日 期 2019.3.23

**计算机科学与技术学院**

**目 录**

[第1章 实验基本信息 - 3 -](#_Toc4398593)

[1.1 实验目的 - 3 -](#_Toc4398594)

[1.2 实验内容 - 3 -](#_Toc4398595)

[1.3 实验环境 - 3 -](#_Toc4398596)

[第2章 需求分析 - 4 -](#_Toc4398597)

[2.1 软件功能 - 4 -](#_Toc4398598)

[2.2 软件性能 - 4 -](#_Toc4398599)

[第3章 实验原理 - 5 -](#_Toc4398600)

[3.1 wireshark的安装 - 5 -](#_Toc4398601)

[3.2 抓包软件的原理 - 5 -](#_Toc4398602)

[3.2.1 系统提供的支持 - 5 -](#_Toc4398603)

[3.2.2 利用libcap抓包 - 6 -](#_Toc4398604)

[第4章 程序流程设计 - 9 -](#_Toc4398605)

[4.1 与用户交互的线程 - 9 -](#_Toc4398606)

[4.2 工作线程 - 10 -](#_Toc4398607)

[第5章 代码实现 - 11 -](#_Toc4398608)

[5.1 代码总体结构 - 11 -](#_Toc4398609)

[5.2 init\_pcap函数的实现 - 11 -](#_Toc4398610)

[5.3 filter\_pcap函数的实现 - 12 -](#_Toc4398611)

[5.4 start\_pcap函数的实现 - 12 -](#_Toc4398612)

[第6章 运行结果 - 13 -](#_Toc4398613)

[6.1 wireshark使用 - 13 -](#_Toc4398614)

[6.2 实验程序运行结果 - 14 -](#_Toc4398615)

[第7章 总结 - 16 -](#_Toc4398616)

[7.1本次实验的收获 - 16 -](#_Toc4398617)

[第8章 参考文献 - 16 -](#_Toc4398618)

# 第1章 实验基本信息

## 1.1 实验目的

（1）熟悉使用抓包软件;

（2）了解抓包的原理及流程;

（3）利用libpcap或winpcap接口进行编程；

## 1.2 实验内容

1. 熟练使用sniffer 或 wireshark 软件，对协议进行还原，要求至少能够找到访问网页的四元组；
2. 利用libpcap或winpcap进行编程，能够对本机的数据包进行捕获分析，将分析结果写入文件。

## 1.3 实验环境

本实验中使用的系统环境是Ubuntu 18.04，编程语言为C语言，使用了gcc编译器；

# 第2章 需求分析

## 2.1 软件功能

软件要实现如下的功能：

1. 用户开启软件。用户能够选择是进行抓包，还是退出软件；
2. 用户能够选择存储结果的文件；
3. 用户能够通过过滤过滤规则对抓包结果进行过滤；
4. 用户可以随时选择停止抓包；
5. 停止抓包后，用户可以选择再次抓包，或者退出程序；

## 2.2 软件性能

软件应该具备良好的性能，比如软件应该易于使用、能给出准确的结果等。这里主要关注四个方面：

1. 易用使用。每步的操作应该简单清晰，使用的步骤应该尽量少；
2. 空间性能。软件可能会一直抓包，要保证无论持续抓包的时间有多长，使用的控件都要在合理的范围内；
3. 时间性能。软件要实时给出抓包后分析的结果；
4. 稳定性。本软件要求能够稳定运行，长时间使用该软件时，或者是输入不规范时，有相应的优雅的反应，而不是软件直接崩溃。

# 第3章 实验原理

## 3.1 wireshark的安装

在Linux下，打开bash shell，输入如下的命令：

然后等待安装成功。安装成功后，就可以运行wireshark了，由于在Ubuntu下获取网络设备并监听抓包需要root权限，因此，如果是在普通的用户模型下，可以用如下的命令打开软件：

然后就可以在软件中进行抓包等等操作了。

## 3.2 抓包软件的原理

### 3.2.1 系统提供的支持

不同的操作系统实现的底层包捕获机制可能是不一样的，但从形式上看是大同小异的。如图3-1所示，数据包常规的传输路径依次为网卡、设备驱动层、数据链路层、IP层、传输层、最后到达应用程序。

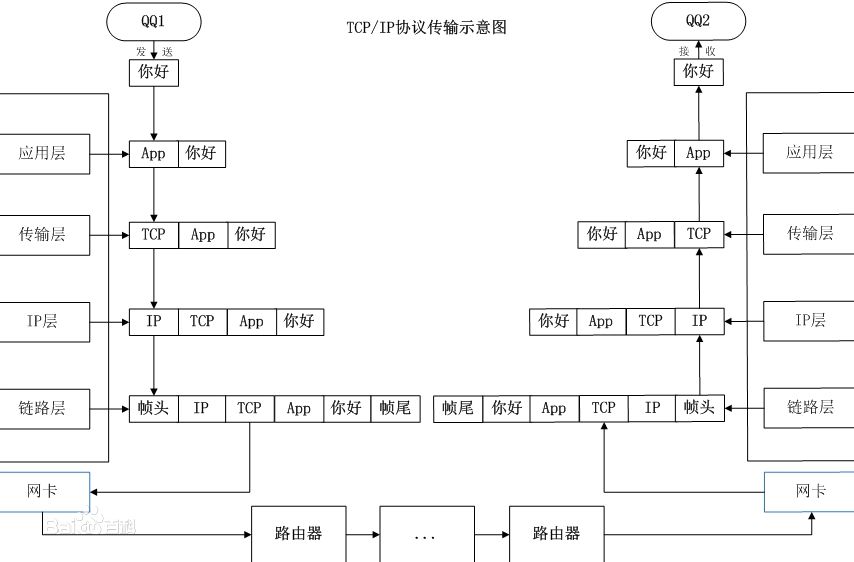


图3-1 TCP/IP传输协议示意图

包捕获机制是在数据链路层增加一个旁路处理，对发送和接收到的数据包做过滤/缓冲等相关处理，最后直接传递到应用程序。包捕获机制并不影响操作系统对数据包的网络栈处理，对用户程序而言，包捕获机制提供了一个统一的接口，使用户程序只需要简单的调用若干函数就能获得所期望的数据包。针对特定操作系统的捕获机制对用户透明，使用户程序有比较好的可移植性。包过滤机制是对所捕获到的数据包根据用户的要求进行筛选，最终只把满足过滤条件的数据包传递给用户程序。

### 3.2.2 利用libcap抓包

前面说到，包捕获机制是在数据链路层增加一个旁路处理，如图3-2所示，Libpcap在这个过程中位于应用层。

Lipcap提供了若干的API函数，让用户可以打开设备，并进行包的捕获和处理。主要有以下函数：

// 该函数用于返回网络设备名，是一个字符串

char \*pcap\_lookupdev(char \*errbuf)

// 该设备获得指定网络设备的网络号和掩码

int pcap\_lookupnet(char \*device, bpf\_u\_int32 \*netp,bpf\_u\_int32 \*maskp, char \*errbuf)

// 获得用于捕获网络数据包的数据包捕获描述字

pcap\_t \*pcap\_open\_live(char \*device, int snaplen,int promisc, int to\_ms,char \*ebuf)

// 编译过滤规则

int pcap\_compile(pcap\_t \*p, struct bpf\_program \*fp,char \*str, int optimize, bpf\_u\_int32 netmask)

// 抓包后的回调函数

void callback(u\_char \* userarg, const struct pcap\_pkthdr \* pkthdr, const u\_char \* packet)

// 获取下一个数据包

u\_char \*pcap\_next(pcap\_t \*p, struct pcap\_pkthdr \*h)

// 关闭设备，释放资源

void pcap\_close(pcap\_t \*p)

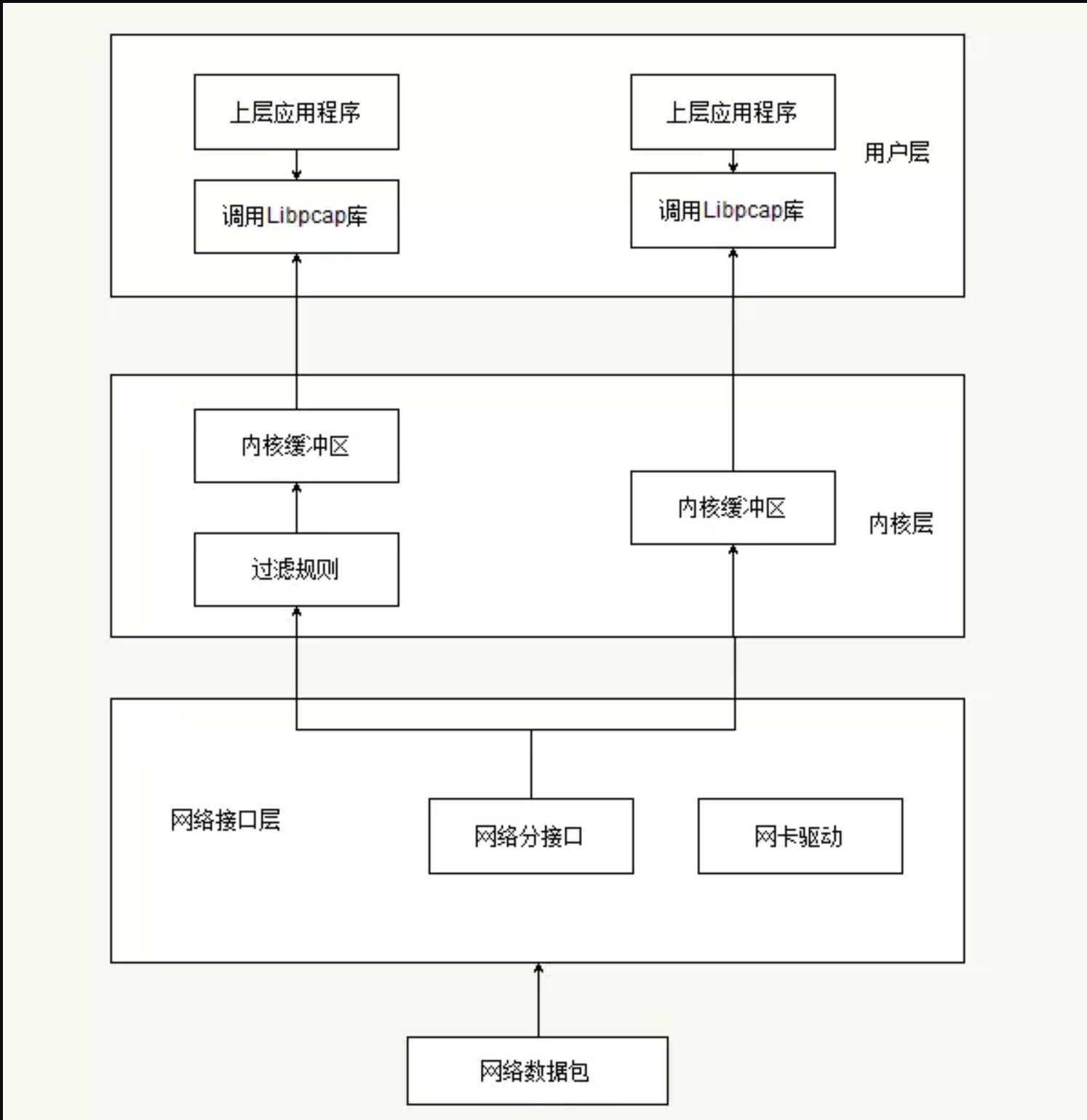


图3-2 包捕获的旁路处理

在使用libpcap进行抓包时，整体流程如图3-3所示，在每一步骤中，libpcap都提供了相应的接口函数来完成功能。

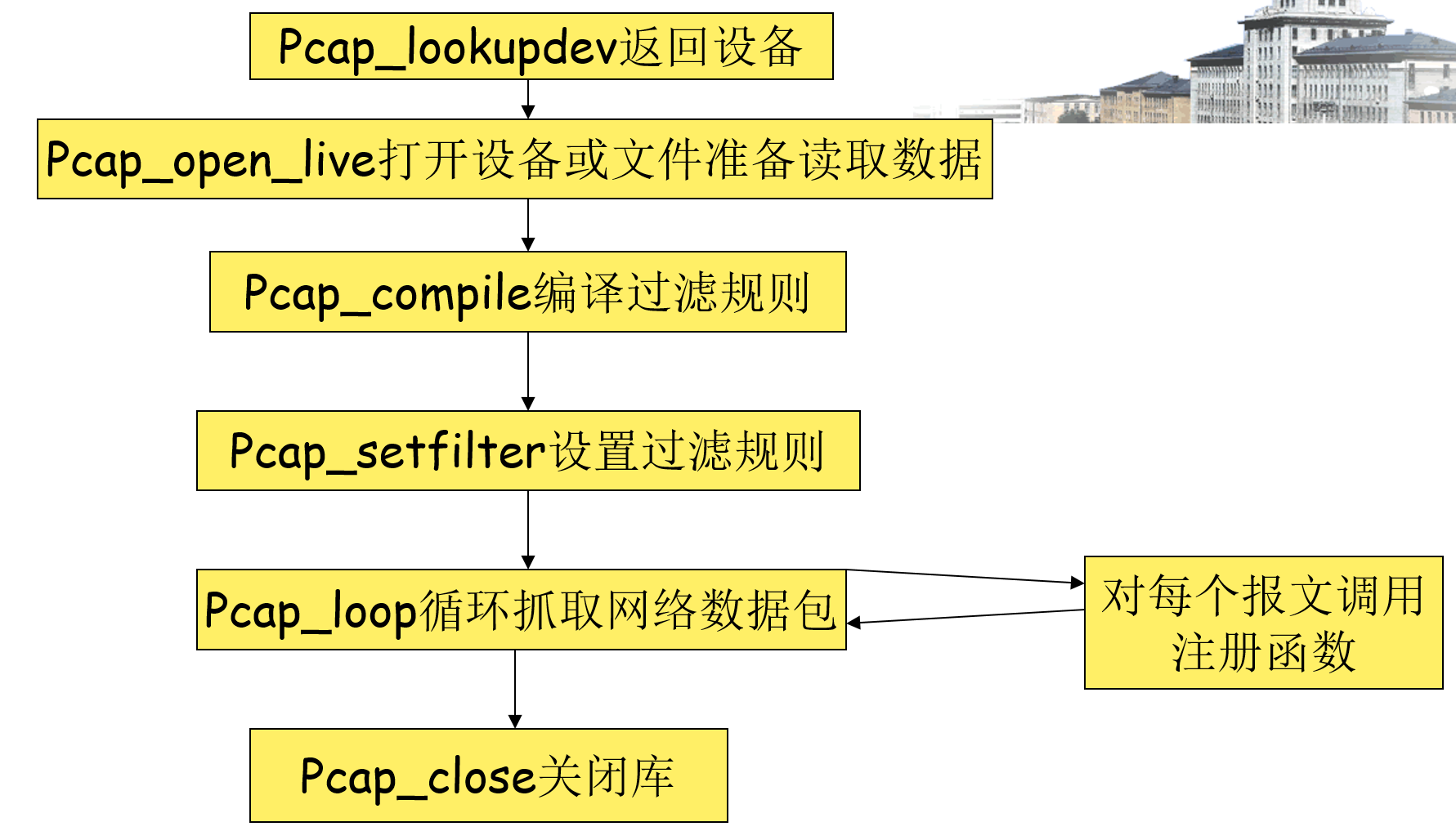


图3-3 使用libpcap的流程

# 第4章 程序流程设计

程序让用户能够随时停止转包，程序分为两个线程，一个线程与用户交互，另一个线程是工作线程，进行抓包、分析。

## 4.1 与用户交互的线程

如图4-1所示，与用户交互的线程是循环运行的，在每一次循环中，依次询问：

1. 用户是否进行抓包？
2. 抓包的分析结果存储在那个文件？
3. 抓包时的过滤规则是什么？
4. （抓包开始后），是否立马停止抓包？

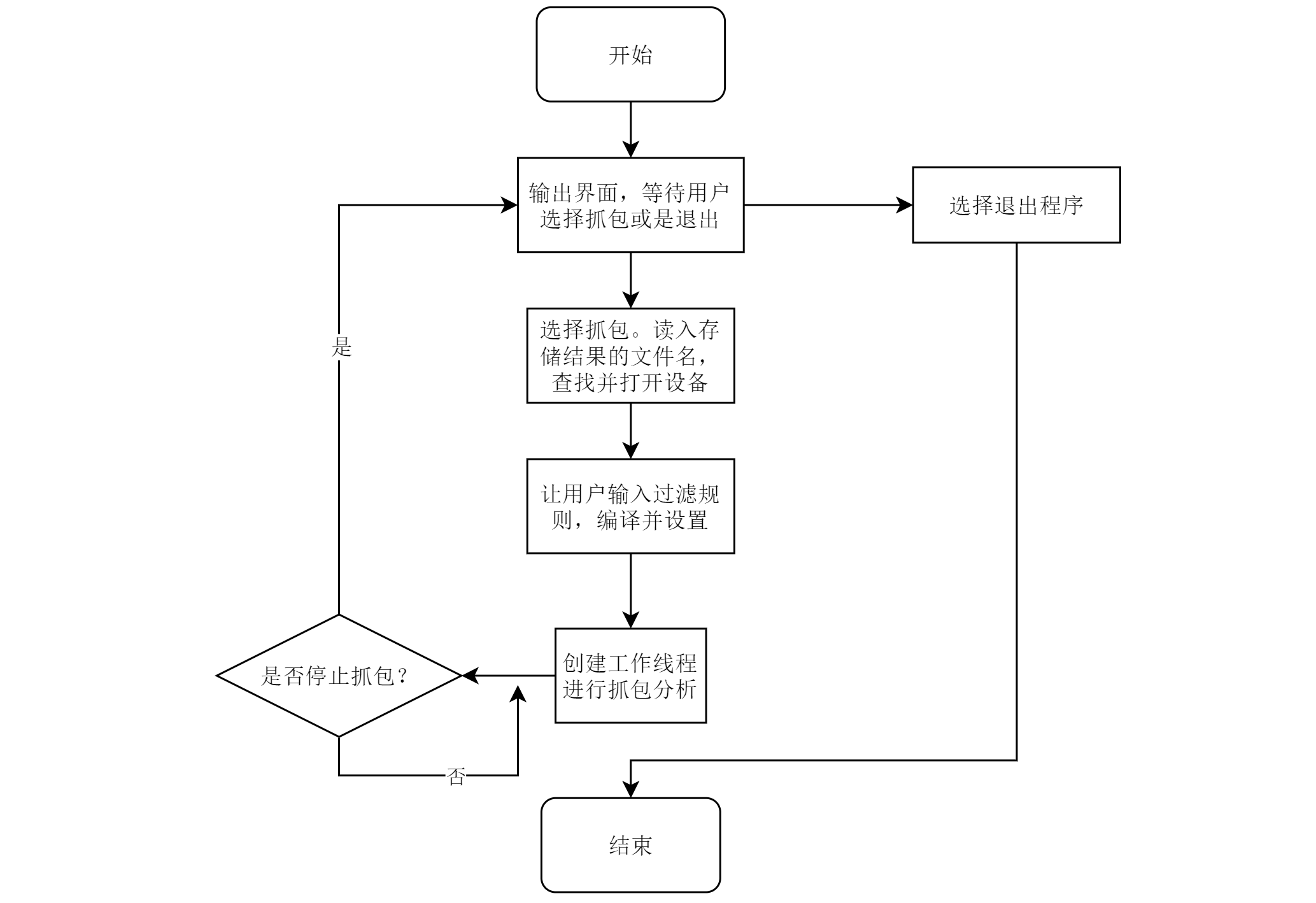


图4-1 与用户交互的线程的流程图

在这个过程中，该线程也会准备好抓包需要的环境，比如打开设备，根据用户的输入进行过滤规则的编译和设置。当然，最重要的是，在准备好后，该线程会创建一个新的线程，作为工作线程，进行抓包和分析。

## 4.2 工作线程

工作线程的主要功能就是抓包、分析，并将结构写入文件。工作线程的流程如图4-2所示。

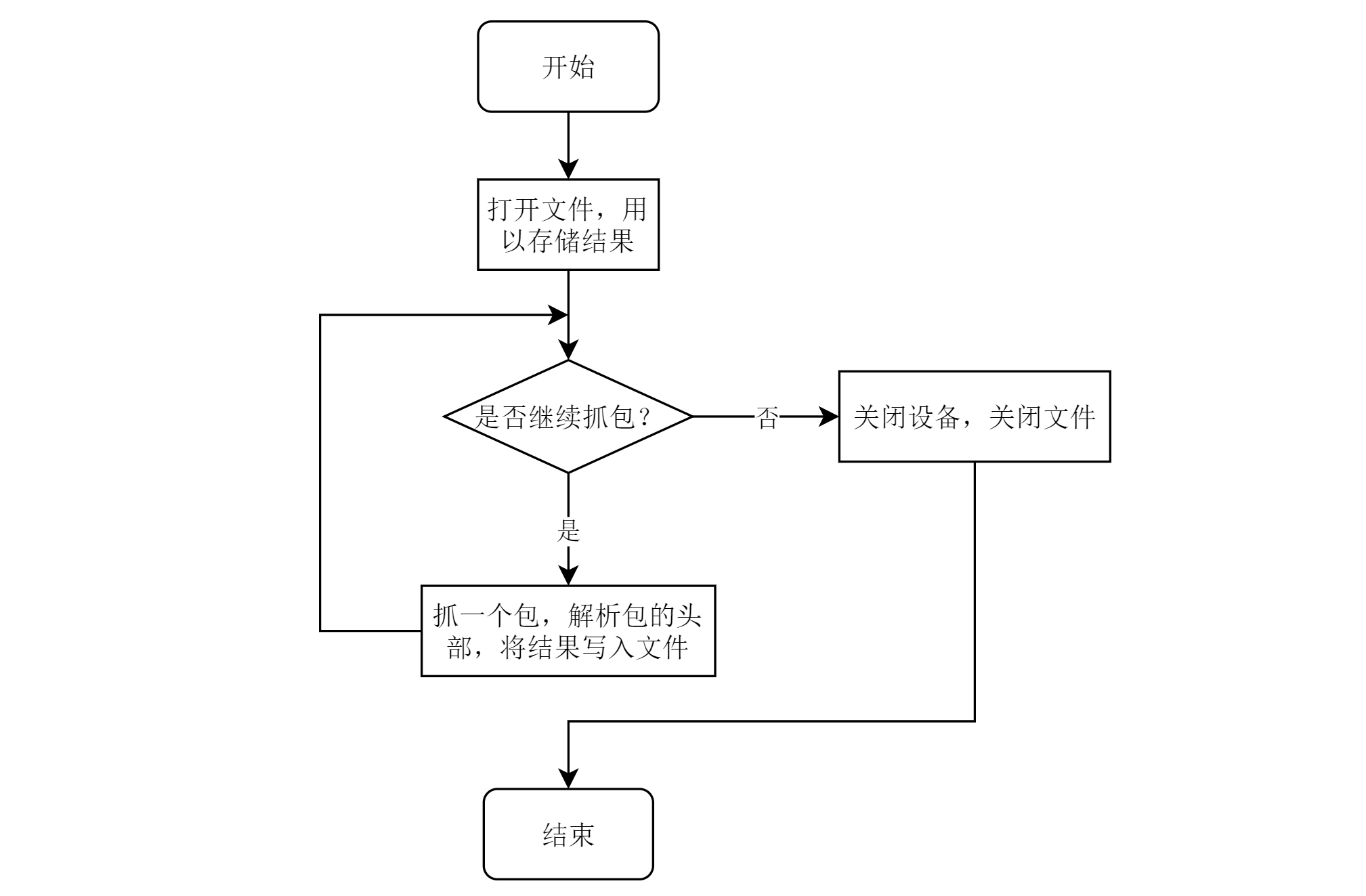


图4-2 工作线程的执行流程图

工作线程的主要工作就是循环接收包，并对包进行处理，本实验中，仅对包的头部进行处理，提取出访问网页的四元组。

# 第5章 代码实现

## 5.1 代码总体结构

整个程序分为五个部分，第一个是主函数。主函数循环执行，每一次循环就是一次抓包分析，每次循环中调用其余的函数完成具体功能。设置一个全局变量flag标识是否抓包。

## 5.2 init\_pcap函数的实现

该函数的定义如下

int init\_pcap(pcap\_t \*\*device, char \*filenamebuf, bpf\_u\_int32 \*netmask);

1. pcap\_t \*\*device：用以存储打开的设备的句柄；
2. char \*filenamebuf：用以存储结果文件名；
3. bpf\_u\_int32 \*netmask：用以存储当前设备的掩码；
4. 返回值为1表示成功，0表示失败；

该函数完成读入用户输入的用以存储抓包结果的文件名，同时查找并打开设备。这个函数有一个值得注意的地方是，查找并打开网络设备需要root权限，因此在代码中进行动态的权限的获取以及释放，相关的代码如图5-1所示。

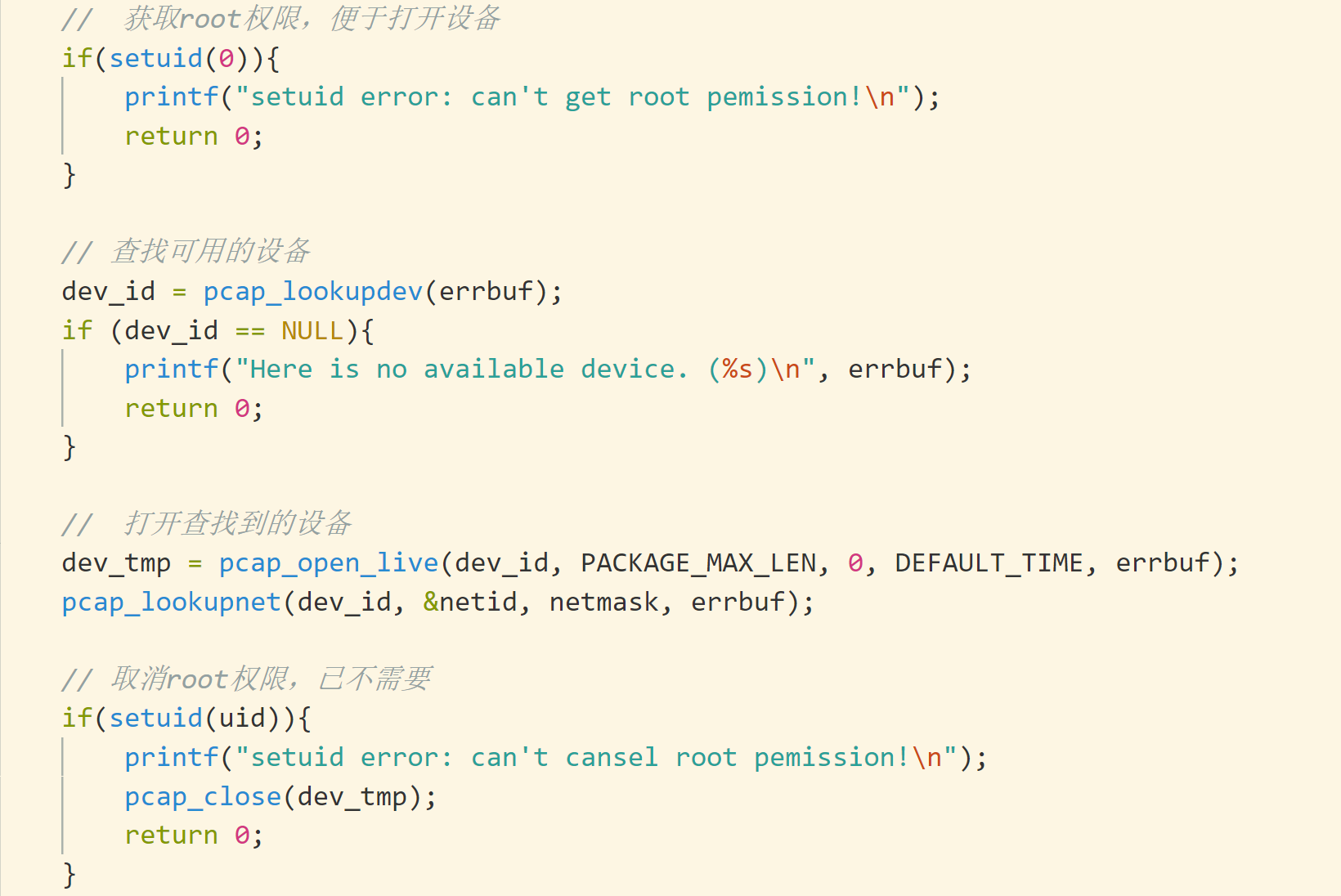


图5-1 利用libpcap提供的函数打开网络设备

## 5.3 filter\_pcap函数的实现

该函数的定义如下：

void filter\_pcap(pcap\_t \*device, bpf\_u\_int32 netmask);

1. pcap\_t \*device：用以指明抓包的设备；
2. bpf\_u\_int32 netmask：指明设备的掩码；
3. 无返回值；

该函数的功能是让用户输入过滤规则，然后编译并设置这些规则。主要的代码如图5-2所示。

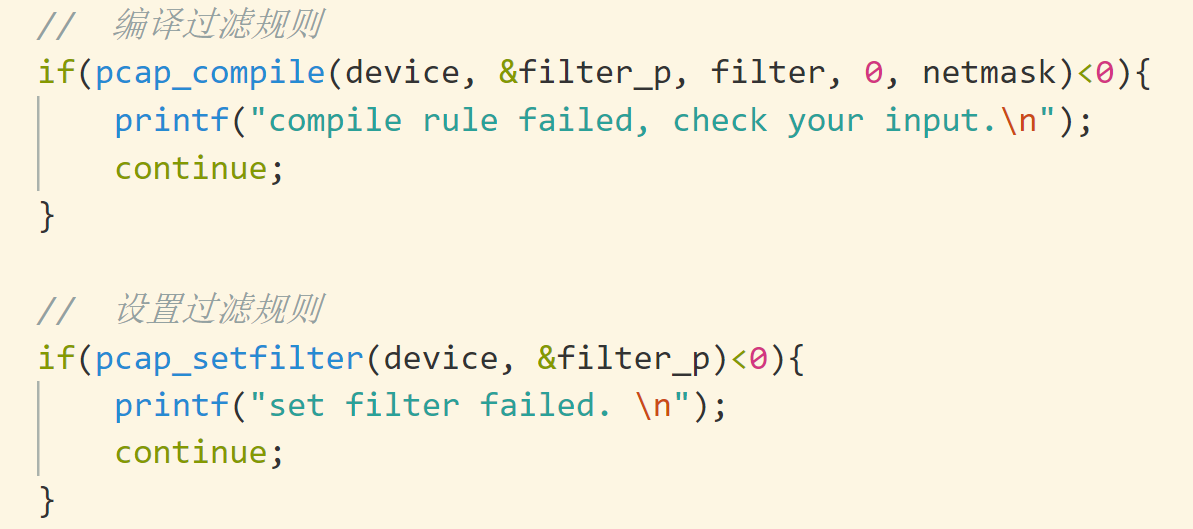


图5-2 编译并设置过滤规则

## 5.4 start\_pcap函数的实现

该函数的定义如下：

void\* start\_pcap(void \*param)

1. void \*param：一个结构体指针，该结构体为info\_pcap，定义在文件minisniffer.h中。指明结果文件的路径和抓包的设备；

该函数的功能就是抓包、分析头部。这个函数是工作线程的依托函数，也就是工作线程的回调函数。为了完成该函数解析头部的功能，在文件minisniffer.h中定义了一系列的结构体。

# 第6章 运行结果

## 6.1 wireshark使用

在bash shell中输入命令

就可以进入应用，然后开始抓包，可以得到抓包的结果，如图6-1所示。可以在上方的输入过滤条件，结果中已经列出了包头部的一些信息，比如源IP与目的IP。在真个视图的下方有关于头部的详细信息，可以看到，头部按从底层到顶层的顺序排列。以途中选中的数据包为例，依次是链路帧头部信息、IP头部信息、TCP头部信息。

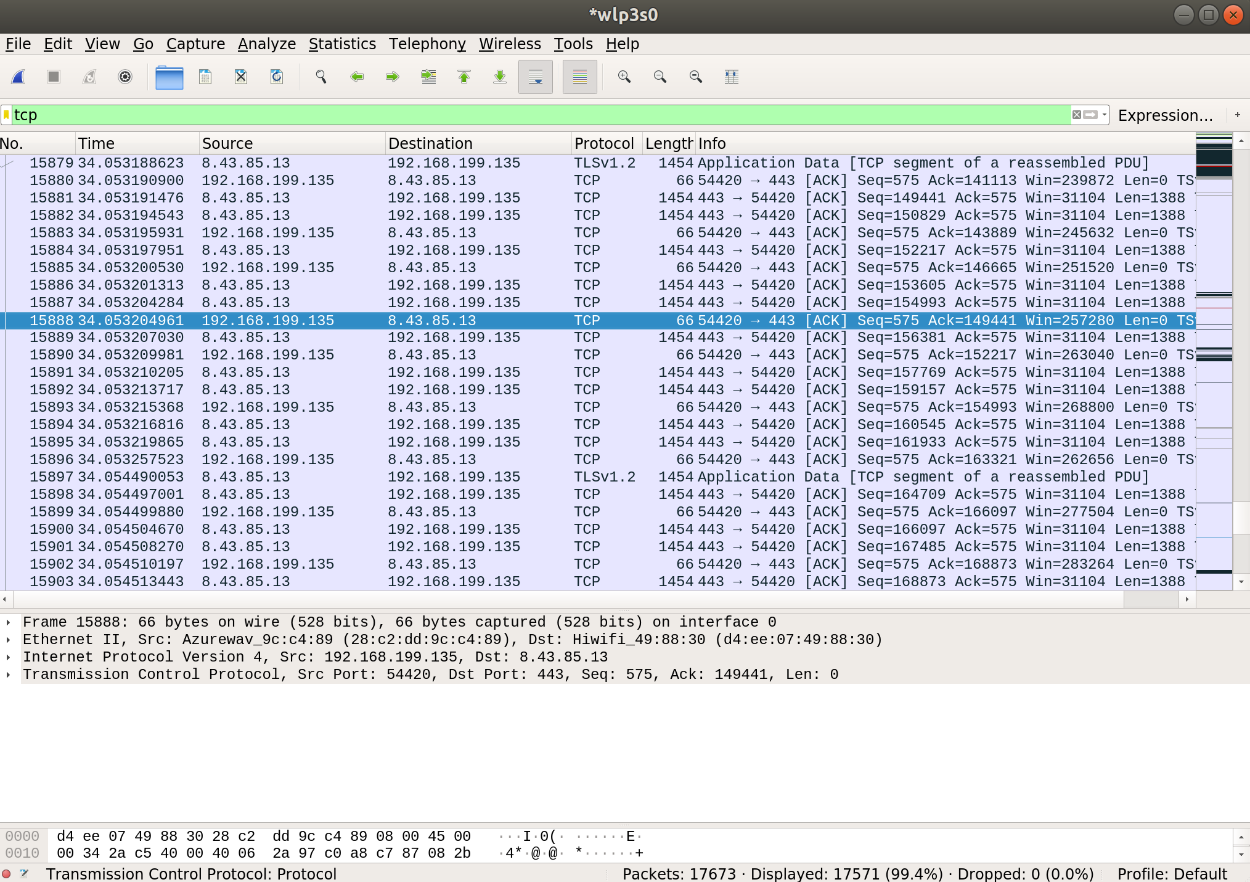


图6-1 wireshark抓包结果图

其中，在IP头部中可以找到源IP和目的IP，如图6-2所示。

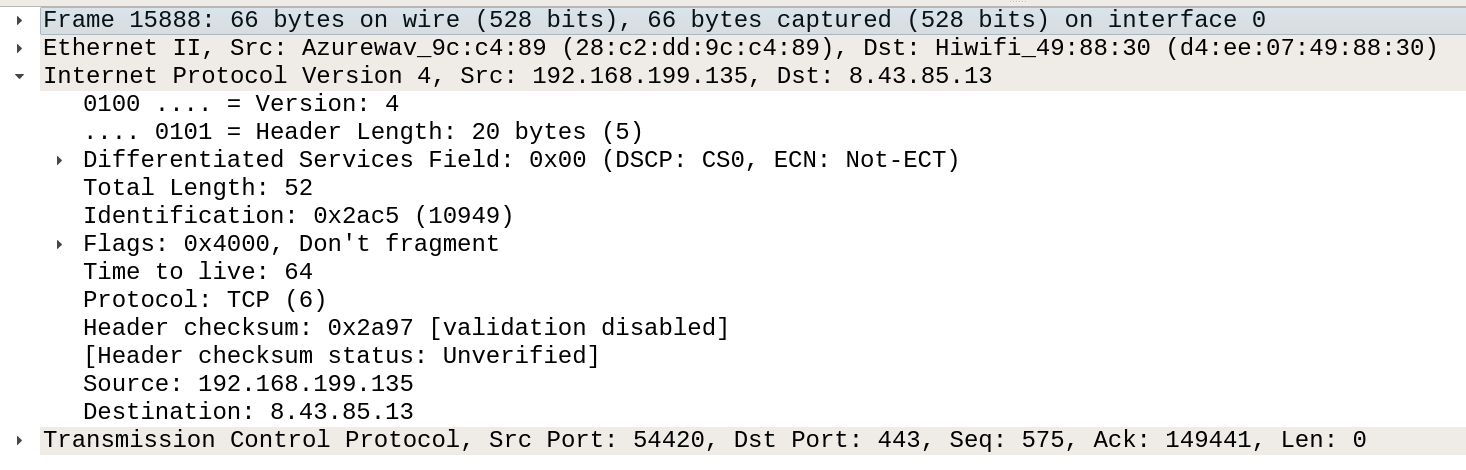


图6-2 wireshark中解析出得IP头部信息

在TCP（传输层）头部中可以找到源端口和目的端口的信息，如图6-3所示。

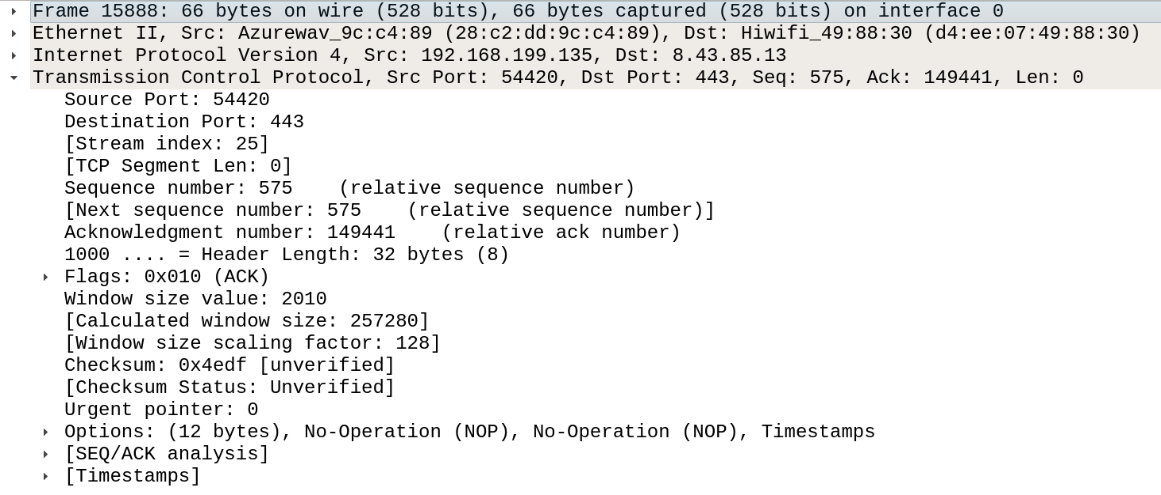


图6-3 wireshark中解析出得TCP头部信息

## 6.2 实验程序运行结果

打开程序，进入程序的主菜单页面，如图6-4所示，选择“1”，就可以开始参数的设置。

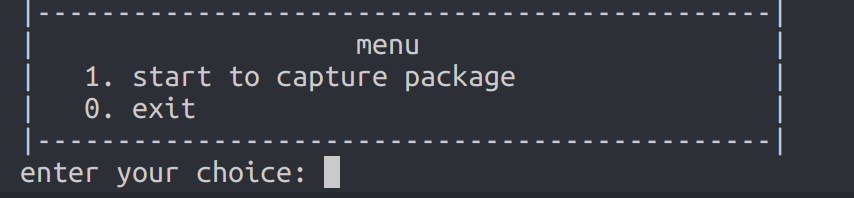


图6-4 程序的主菜单

这里为了判断结果的正确性，我们仅抓取固定IP的数据包，作为例子，我这里抓取访问清华大学官网的数据包。首先通过ping获得清华大学官网的IP，如图6-5所示。

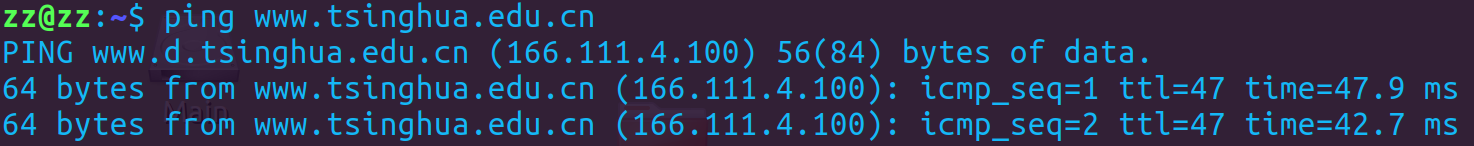


图6-5 通过ping获取清华大学官网的IP

然后完整的依次抓包的过程如图6-6所示。在抓取数据时，程序会持续抓取，直到用户输入“y”或者“Y”，程序才会停止抓包。

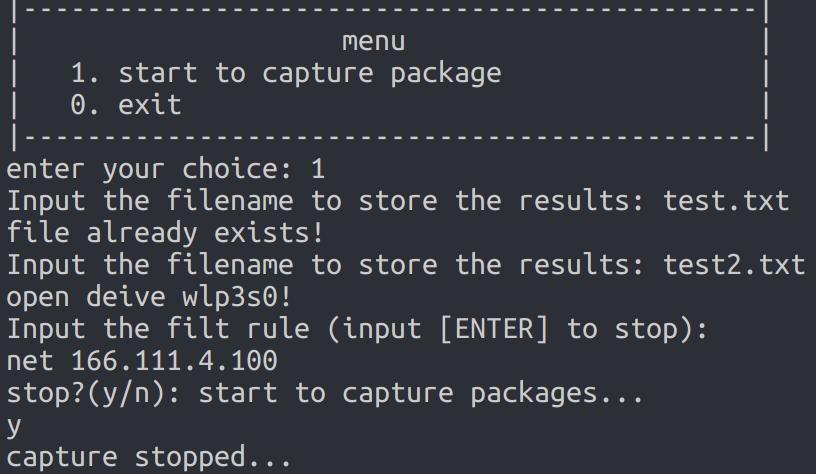


图6-6 程序完整执行依次的结果图

抓包分析的结果存在文件中，如图6-7所示。可以看到，抓包是正确的。

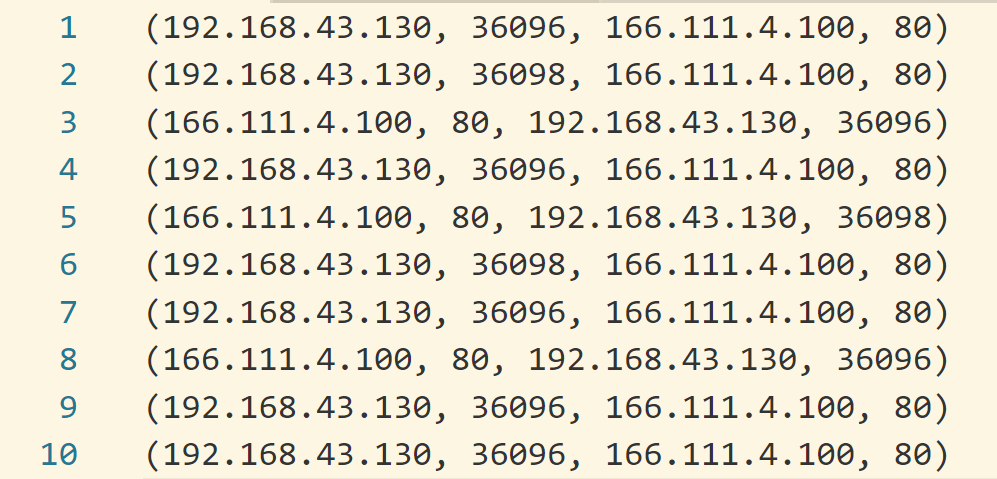


图6-7 抓取的访问网页的四元组的结果

# 第7章 总结

## 7.1本次实验的收获

（1）熟悉了抓包软件wireshark的使用；

（2）了解并掌握了使用libpcap提供的API编写抓包程序技能；

（3）理解了抓包软甲的原理；

# 第8章 参考文献

[1] <https://blog.csdn.net/xiaoxianerqq/article/details/78179751>.