哈爾濱工業大學

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

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：

```
sudo apt install wireshark
```

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：

```
sudo wireshark &
```

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。

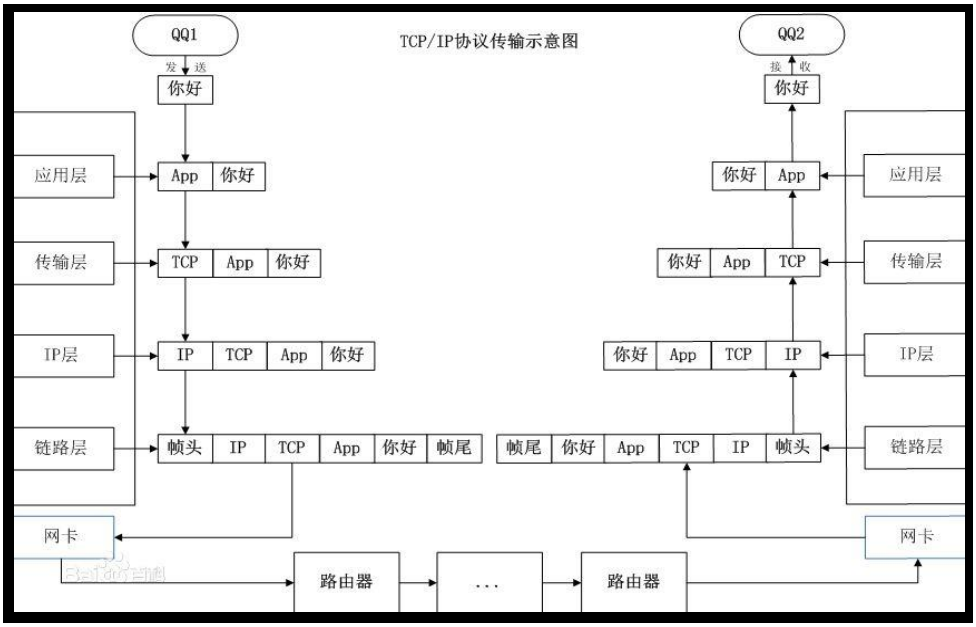

图 3-1 TCP/IP 传输协议示意图

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)

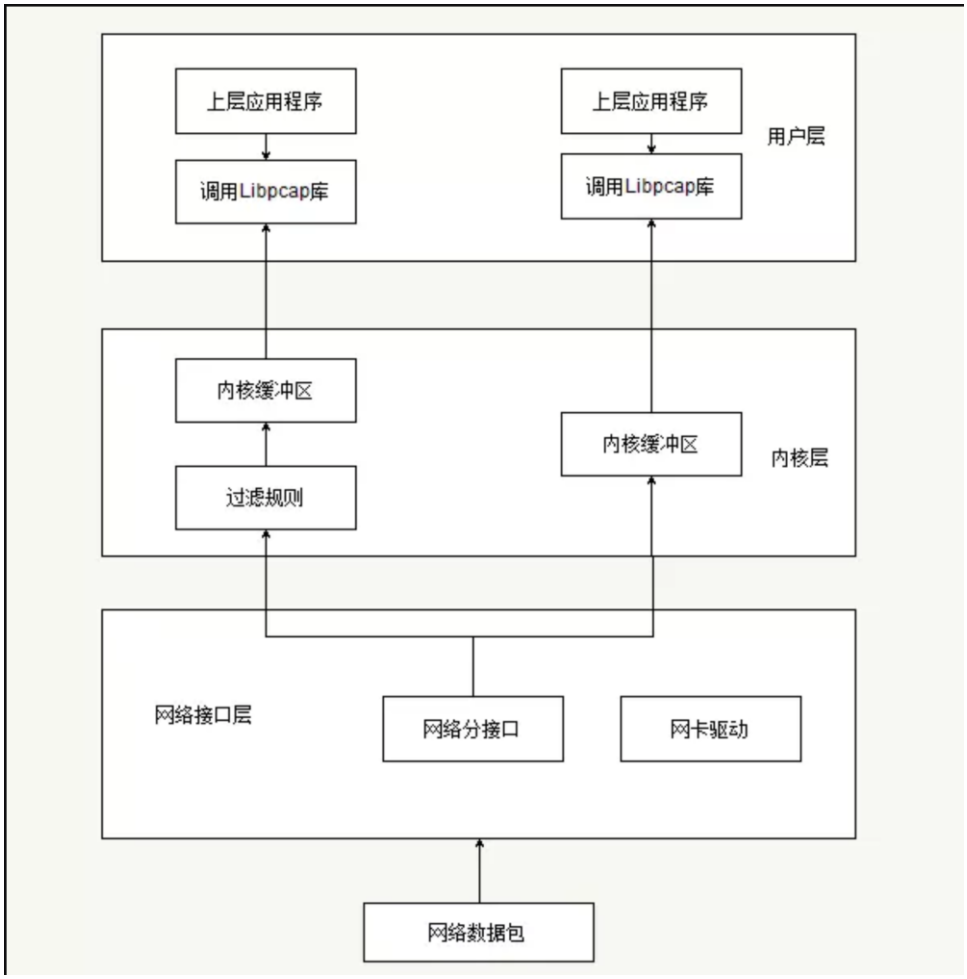

图 3-2 包捕获的旁路处理

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。

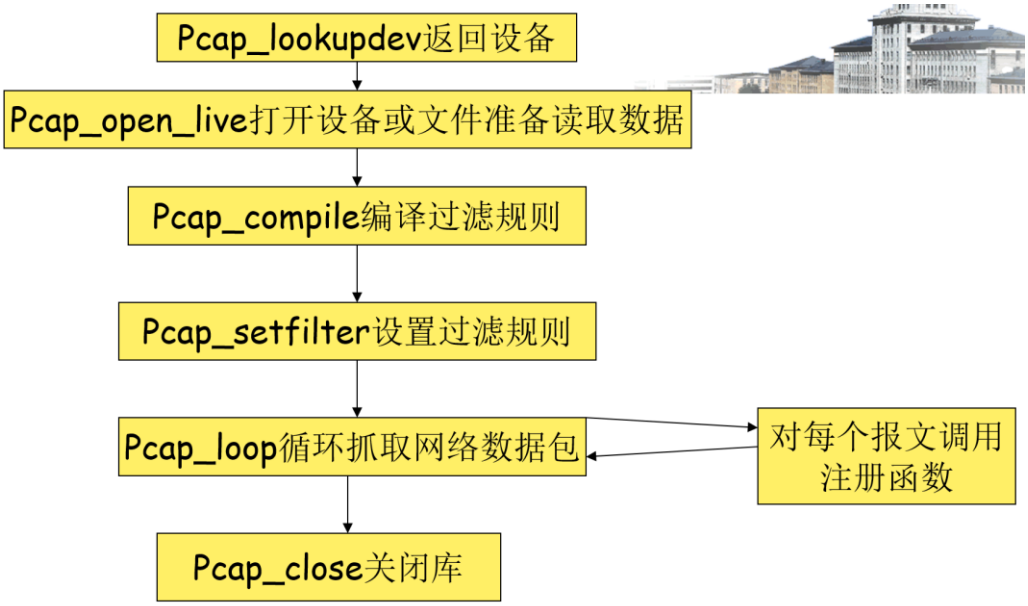

图 3-3 使用 libpcap 的流程

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？

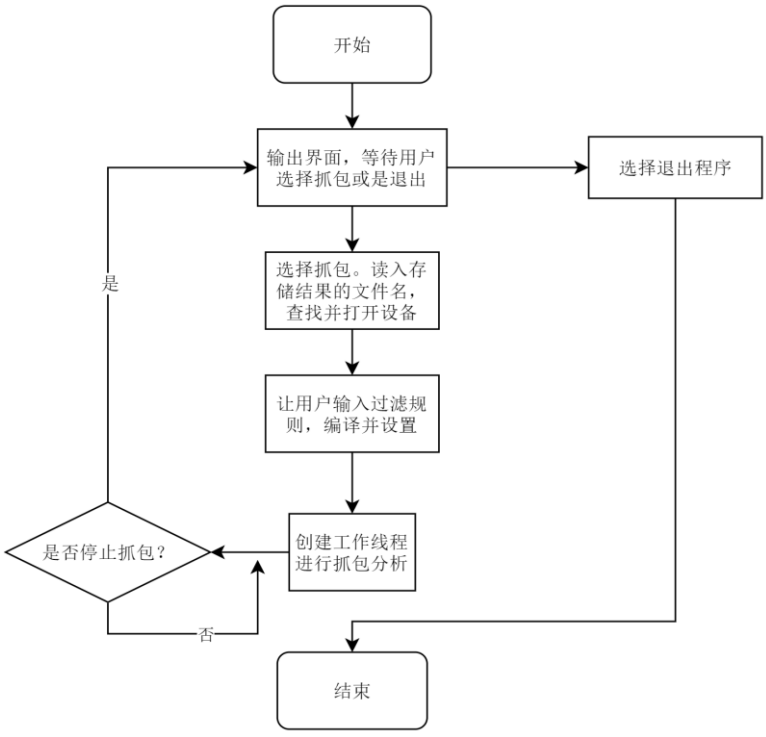

图 4-1 与用户交互的线程的流程图

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。

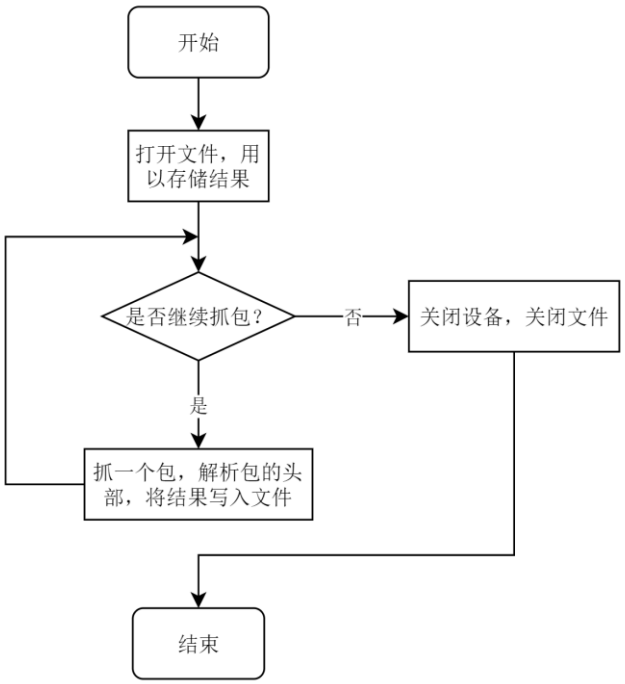

图 4-2 工作线程的执行流程图

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。

```
//  获取root权限，便于打开设备
if(setuid(0)){
    printf("setuid error: can't get root pemission!\n");
    return 0;
}

// 查找可用的设备
dev_id = pcap_lookupdev(errbuf);
if (dev_id == NULL){
    printf("Here is no available device. (%s)\n", errbuf);
    return 0;
}

//  打开查找到的设备
dev_tmp = pcap_open_live(dev_id, PACKAGE_MAX_LEN, 0, DEFAULT_TIME, errbuf);
pcap_lookupnet(dev_id, &netid, netmask, errbuf);

// 取消root权限，已不需要
if(setuid(uid)){
    printf("setuid error: can't cansel root pemission!\n");
    pcap_close(dev_tmp);
    return 0;
}
```

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。

```
//  编译过滤规则
if(pcap_compile(device, &filter_p, filter, 0, netmask)<0){
    printf("compile rule failed, check your input.\n");
    continue;
}

//  设置过滤规则
if(pcap_setfilter(device, &filter_p)<0){
    printf("set filter failed. \n");
    continue;
}
```

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

sudo wireshark &

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。

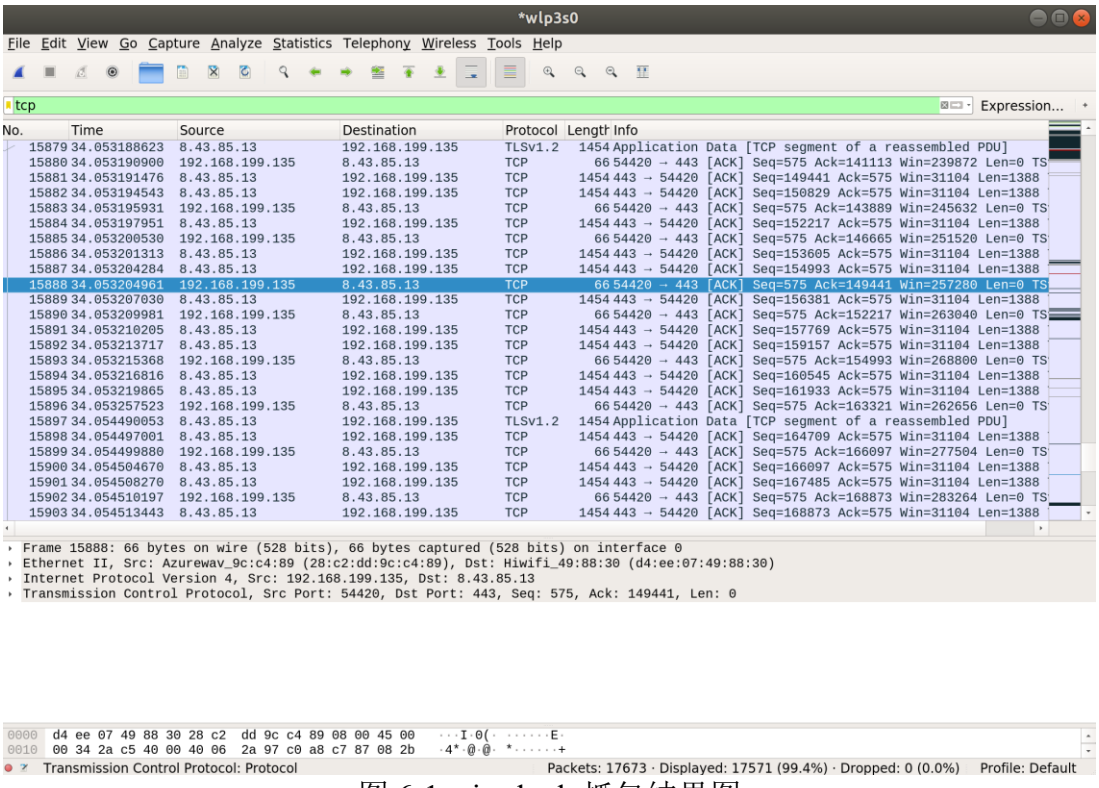

图 6-1 wireshark 抓包结果图

其中，在 IP 头部中可以找到源 IP 和目的 IP，如图 6-2 所示。

```
▸ Frame 15888: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: Azurewav_9c:c4:89 (28:c2:dd:9c:c4:89), Dst: Hiwifi_49:88:30 (d4:ee:07:49:88:30)
▾ Internet Protocol Version 4, Src: 192.168.199.135, Dst: 8.43.85.13
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 52
    Identification: 0x2ac5 (10949)
  ▸ Flags: 0x4000, Don't fragment
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0x2a97 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.199.135
    Destination: 8.43.85.13
▸ Transmission Control Protocol, Src Port: 54420, Dst Port: 443, Seq: 575, Ack: 149441, Len: 0
```

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。

```
▸ Frame 15888: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: Azurewav_9c:c4:89 (28:c2:dd:9c:c4:89), Dst: Hiwifi_49:88:30 (d4:ee:07:49:88:30)
▸ Internet Protocol Version 4, Src: 192.168.199.135, Dst: 8.43.85.13
▾ Transmission Control Protocol, Src Port: 54420, Dst Port: 443, Seq: 575, Ack: 149441, Len: 0
    Source Port: 54420
    Destination Port: 443
    [Stream index: 25]
    [TCP Segment Len: 0]
    Sequence number: 575    (relative sequence number)
    [Next sequence number: 575    (relative sequence number)]
    Acknowledgment number: 149441    (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x010 (ACK)
    Window size value: 2010
    [Calculated window size: 257280]
    [Window size scaling factor: 128]
    Checksum: 0x4edf [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
  ▸ [Timestamps]
```

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。

```
|-----------------------------------------------|
|                    menu                       |
|   1. start to capture package                 |
|   0. exit                                     |
|-----------------------------------------------|
enter your choice:
```

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。

```
zz@zz:~$ ping www.tsinghua.edu.cn
PING www.d.tsinghua.edu.cn (166.111.4.100) 56(84) bytes of data.
64 bytes from www.tsinghua.edu.cn (166.111.4.100): icmp_seq=1 ttl=47 time=47.9 ms
64 bytes from www.tsinghua.edu.cn (166.111.4.100): icmp_seq=2 ttl=47 time=42.7 ms
```

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。

```
|-------------------------------------------------|
|                    menu                         |
|   1. start to capture package                   |
|   0. exit                                       |
|-------------------------------------------------|
enter your choice: 1
Input the filename to store the results: test.txt
file already exists!
Input the filename to store the results: test2.txt
open deive wlp3s0!
Input the filt rule (input [ENTER] to stop):
net 166.111.4.100
stop?(y/n): start to capture packages...
y
capture stopped...
```

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。

```
1   (192.168.43.130, 36096, 166.111.4.100, 80)
2   (192.168.43.130, 36098, 166.111.4.100, 80)
3   (166.111.4.100, 80, 192.168.43.130, 36096)
4   (192.168.43.130, 36096, 166.111.4.100, 80)
5   (166.111.4.100, 80, 192.168.43.130, 36098)
6   (192.168.43.130, 36098, 166.111.4.100, 80)
7   (192.168.43.130, 36096, 166.111.4.100, 80)
8   (166.111.4.100, 80, 192.168.43.130, 36096)
9   (192.168.43.130, 36096, 166.111.4.100, 80)
10  (192.168.43.130, 36096, 166.111.4.100, 80)
```

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.