课程编号：A0802050020

《网络与系统安全》实验报告

姓　　名	
班　　级	
实 验 名 称	
开 设 学 期	
开 设 时 间	
报 告 日 期	
评 定 成 绩	

东北大学软件学院

一、 实验目的

1. 利用数据链路访问接口来进行网络数据包监控。
2. 加深对 TCP/IP 协议族中常见协议的理解。
3. 掌握如何利用数据包监控程序来了解当前网络状况。
4. 编写过滤条件，对网络数据包进行过滤，提取出所关心的网络数据。

二、 实验环境

Linux Ubuntu 18.04.6
Linux CentOS release 7.0(Final)
Visual studio code C/C++ May 2022 (version 1.68)

三、 实验流程

1. 底层模块(数据包捕获)

在 Linux 下使用 <sys/socket.h> 头文件中 socket() 函数来创建套接字，原型为：

int socket(int af, int type, int protocol);

af 为地址族（Address Family），也就是 IP 地址类型，常用的有 AF_INET 和 AF_INET6。AF 是“Address Family”的简写，INET 是“Inetnet”的简写。AF_INET 表示 IPv4 地址，例如 127.0.0.1；AF_INET6 表示 IPv6 地址，例如 1030::C9B4:FF12:48AA:1A2B。PF 和 AF 是一样的，PF 是“Protocol Family”的简写。例如，PF_INET 等价于 AF_INET，PF_INET6 等价于 AF_INET6。

type 为数据传输方式/套接字类型，常用的有 SOCK_STREAM（流格式套接字/面向连接的套接字）和 SOCK_DGRAM（数据报套接字/无连接的套接字）。

protocol 表示传输协议，常用的有 IPPROTO_TCP 和 IPPTOTO_UDP，分别表示 TCP 传输协议和 UDP 传输协议。

在本实验中的底层模块中，首先声明一个套接字 s, 并实例化该套接字。

```
int s;
if((s=socket(PF_PACKET,SOCK_PACKET,htons(ETH_P_ALL)))<0){
    perror(" socket");
    exit(1);
}
```

在 linux 环境中要从链路层（MAC）直接收发数据帧，可以通过 libpcap 与 libnet 两个动态库来分别完成收与发的工作。虽然它已被广泛使用，但在要求进行跨平台移植的软件中使用仍然有很多弊端。这里介绍一种更为直接地、无须安装其它库的从 MAC 层收发数据帧的方式，即通过定义链路层的套接字来完成，简单的格式为：

packet_socket=socket(PF_PACKET,int type,int protocol);

指定地址族 af 为 PF_PACKET 即可实现从 MAC 直接收发数据帧，SOCK_PACKET 需要配合 PF_PACKET 一起使用，指定 protocol 为 htons(ETH_P_ALL) 时表示收发所有的协议。

在循环捕获分析数据包中：

```
while(true){
    if((len=read(s,buff,MAXSIZE))<0){
        perror(" read");
        exit(1);
    }
```

每次循环，从套接字 s 中读取 MAXSIZE 大小的数据到缓存区 buff 中。Buff 存放的就是数据包。至此，我们使用很简单的方法便实现了底层模块（捕获数据包）的全部功能。

2. 中层模块(MAC 层处理模块,IP 层处理模块,TCP 层处理模块,UDP 层处理模块,ICMP 层处理模块)

(1) 模块结构

函数	说明
void print_ethernet(struct ether_header *eth)	显示以太网帧头部结构信息
void print_arp(struct ether_arp *arp)	显示 arp 报头结构信息
void print_ip(struct ip *ip)	显示 ip 报头结构信息
void print_icmp(struct icmp *icmp)	显示 icmp 报头结构信息
void print_tcp(struct tcphdr *tcp)	显示 tcp 报头结构信息
void print_udp(struct udphdr *udp)	显示 udp 报头结构信息
void dump_packet(unsigned char *buff,int len)	将从 Ethernet 报头的初始地址到 FCS 之前的值使用十六进制整数和 ASCII 码来表示
char* mac_ntoa(u_char *d)	将 MAC 地址变换为字符串
char* ip_ttoa(int flag)	将 IP 报头中的标志变换为 ASCII 码的辅助函数
char* ip_ftoa(int flag)	将 IP 报头标志变换为 ASCII 码的辅助函数
char* tcp_ftoa(int flag)	TCP 报头中的标志变换为 ASCII 码的辅助函数
int ip_atou(char *ipa,unsigned int *ip32)	设置指定 IP 地址并检查 IP 的值是否合法

(2) 实现方法

通过底层模块所建立的网络数据包捕获接口捕获流经本网卡的所有原始数据包。并循环处理捕获的数据包.首先，开始处理 Ethernet 的报头。检查 Ethernet 类型之后，分析进行 ARP 协议、IP 协议、其他协议的处理。如果为 IP 协议，则进一步地进行 IP 报头的处理，然后，分别进行下面的 TCP 协议、UDP 协议、ICMP 协议、其他协议等的任一处理。如果判明了协议类型，则按照命令行可选域的指示，显示相关的报头。报头的显示在传输层一级上知道了包的种类之后进行。并且，按照 Ethernet 报头的顺序进行显示。

3. 上层统计处理模块(数据包统计模块，数据包协议统计模块，网络元发现模块，数据包构造模块，数据包过滤模块)

(1) 相关数据结构

1) 声明网元模块结构体

```
struct ne{
    u_char a[6];
    struct ne *next;
};
```

2) 初始化网元结构体

```
struct ne nenode={0,0,0,0,0,0,NULL};
```

3) 声明统计模块结构体

```
struct count{
    //main
    time_t st;
    int mac_s;
    int mac_l;
    int macbyte;
    int mac;
    //p_count
    int macbroad;

    int ipbroad;
    int ipbyte;
    int ip;
    int tcp;
    int udp;

    int icmp;
    int icmp_r;
    int icmp_d;
};
```

4) 初始化统计模块结构体

```
struct count ct={0,0,0,0,0, 0,0,0,0,0, 0,0,0};
```

5) 声明总的命令行标志位结构体

```
struct cmd_flags{
    bool a;//arp和ip,其他
    bool e;//显示Ethernet报头
    bool d;//包的内容是以16进制整数和ASCII码来显示
    bool i;//选择可指定的网口
    char ifname[IFRLEN];//网口名称
    bool p;//协议开关
    bool f;//过滤器开关
};
```

此结构体根据命令行中的输入参数设置对应的标志位，不同的标志位决定着不同的功能和输出结果。

6) 初始化命令行标志位结构体

```
struct cmd_flags f={false,false,false,false,false,false};
```

7) 声明协议标志位结构体(只在命令行标志位结构体指定了-p 时生效)

```
struct print_out{//只在指定了-p时有用
    bool arp;
    bool ip;
    bool icmp;
    bool tcp;
    bool udp;
};
```

8) 初始化协议标志位结构体

```
struct print_out p={false,false,false,false,false};
```

9) 声明过滤器结构体(只在命令行标志位结构体指定了-f 时生效)

```
struct filter{
    bool i;
    bool p;
    unsigned int ip;
    int port;
};
```

10) 初始化过滤器结构体

```
struct filter pf={false,false,0,0};
```

(2) 相关函数

函数	说明
int find_ne(u_char *d)	寻找网元
void p_ne(struct ne *neptr)	展示链表里的网元
void free_ne(struct ne *neptr)	释放网元链表
void p_table()	展示统计信息
void p_count(struct ether_header *eth)	计算统计信息
int getif1(char *ifname,int i)	得到第一个网络接口名
int p_filter(struct ether_header *eth)	过滤出需要的数据包
void endfun()	数据包的统计模块、数据包协议统计模块

(3) 主要功能

1) 网元发现：发现网络上的主机。

2) 数据包的统计模块、数据包协议统计模块：本次实验中统计模块包含的统计信息有：

开始时间，结束时间，运行时间，捕获的所有 MAC 数据帧，网络超短帧，网络超长帧，数据帧大小，数据包数量，捕获数据帧速率，数据包捕获速度，IP 数据包，UDP 数据包，TCP 数据包，ICMP 数据包。

3）设计过滤规则：在完成实验的基本过滤功能后，还完善了新的过滤功能。在本实验中，支持用户指定网口抓包，支持对不同协议数据包进行过滤，支持用户对指定 IP 地址的指定端口的数据包进行过滤。

4. 系统字符命令接口

关于执行 ipdump 时的语法格式，如下所示：

./ipdump [-aedh] [-i ifrname] [-p protocol] [-f filters]

因为使用[]括起来的部分是一个可选域，所以即使不写也没有关系。

当表示全部包信息时才指定-a。在不指定-a 时，Ethernet 类型不显示除了 ARP 或 IP 以外的协议。在指定-a 时，则以收到的所有包为显示对象。

如果指定-e，则显示 Ethernet 报头。但是，在指定-a 时，Ethernet 类型不显示除了 ARP 或 IP 以外的协议。

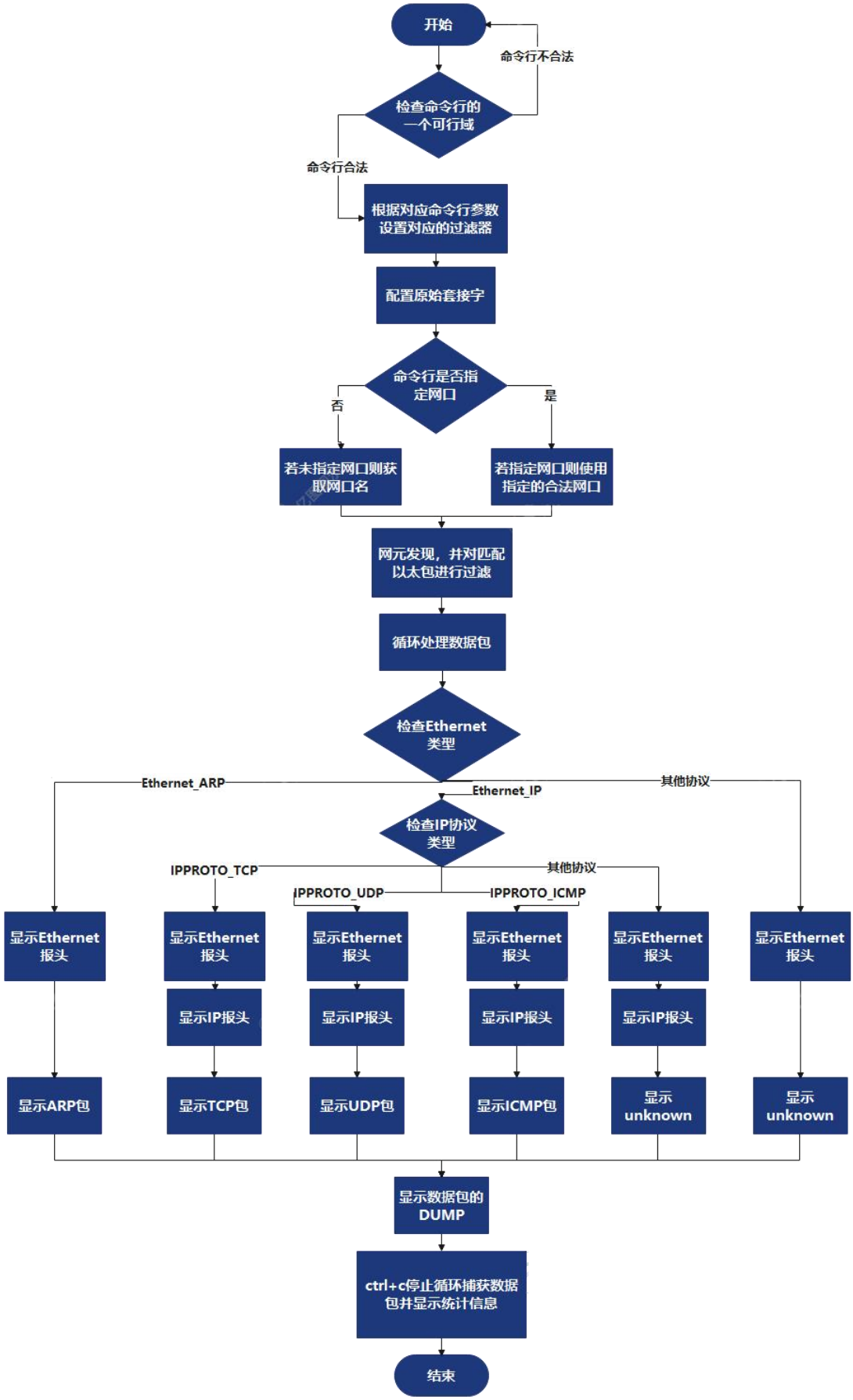
-d 表示包的内容是以 16 进制整数和 ASCII 码来显示的。-h 表示 help，用来简单地显示使用方法。

在[-i ifname]中，指定读取包的接口名称。在 Linux 操作系统中，如果输入[-i eth0]，则表示从 eth0 接口传输包。在不指定时，通过循环测试接口 Lo 表示通信包。

在[-p protocol]中，指定要显示的包的类型，可以指定的包的类型有：arp、ip、icmp、tcp 和 udp。一次可以指定多个协议，例如，在要显示 TCP 端和 IP 报头时，可以指定[-p tcp ip]。

在 [-f filters]中，指定需要读取包的 IP 地址和端口号。基本格式为[-f ip xxx.xxx.xxx.xxx port port_number]。xxx.xxx.xxx.xxx 代表合法的 IP 地址。

5. 实验流程图


开始
命令行不合法
检查命令行的一个可行域
命令行合法
根据对应命令行参数设置对应的过滤器
配置原始套接字
命令行是否指定网口
否
是
若未指定网口则获取网口名
若指定网口则使用指定的合法网口
网元发现，并对匹配以太包进行过滤
循环处理数据包
检查Ethernet类型
Ethernet_ARP
Ethernet_IP
其他协议
检查IP协议类型
IPPROTO_TCP
IPPROTO_UDP
IPPROTO_ICMP
其他协议
显示Ethernet报头
显示Ethernet报头
显示Ethernet报头
显示Ethernet报头
显示Ethernet报头
显示Ethernet报头
显示IP报头
显示IP报头
显示IP报头
显示IP报头
显示ARP包
显示TCP包
显示UDP包
显示ICMP包
显示unknown
显示unknown
显示数据包的DUMP
ctrl+c停止循环捕获数据包并显示统计信息
结束

四、 关键代码及说明

1. ipdump.c 源文件部分

(1) void endfun()

```
void endfun(){
    //统计
    p_table();
    //网元
    p_ne(&nenode);
    free_ne(nenode.next);
    exit(0);
}
```

此函数的生效位置是在 main 函数中建立套接字读取数据包前：

```
    //信号
    signal(SIGINT,endfun);
    //原始套接字
    int s;
    if((s=socket(PF_PACKET,SOCK_PACKET,htons(ETH_P_ALL)))<0){
        perror(" socket");
        exit(1);
    }
```

这里给函数 endfun()设置了一个标准信号量 SIGINT，该信号量在命令行 Ctrl+C 后生效。p_table()函数以表格的形式打印出统计信息。p_ne()函数则根据传入的参数&nenode 网元链表打印出网络上发现的所有网元。free_ne()则释放掉对应的网元链表。

(2) void help()

```
void help(){
    printf("usage: ./ipdump [-aedht] [-p protocols] [-i ifrname] [-f filters]\n");
    printf("protocols: arp ip icmp tcp udp \n");//other??
    printf("filters: ip <IP address> port <PORT number>\n");
    printf("default: ./ipdump -p arp ip icmp tcp udp\n");
}
```

打印出帮助信息，这里给出了程序的使用规则和默认功能。新增功能为对指定 IP 和端口号进行抓包处理。

(3) int p_filter(struct ether_header *eth)

```
int p_filter(struct ether_header *eth){//0 则进一部处理，1 则不处理
    char *ptr=(char *)eth;
    ptr=ptr+sizeof(struct ether_header);
    struct ip *ip;
    struct tcphdr *tcp;
    struct udphdr *udp;
    if(ntohs(eth->ether_type)==ETHERTYPE_IP){
```

```
        ip=(struct ip *)ptr;
        if(pf.i==true){
            if(pf.ip!=ntohl(ip->ip_src.s_addr)&&pf.ip!=ntohl(ip->ip_dst.s_addr)){
                return 1;
            }else if(pf.port==false){
                return 0;
            }
        }
        ptr=ptr+((int)(ip->ip_hl)<<2);
        switch(ip->ip_p){
            case IPPROTO_TCP://TCP 匹配
                tcp=(struct tcphdr *)ptr;
                if(ntohs(tcp->th_sport)==pf.port||ntohs(tcp->th_dport)==pf.port)
                    return 0;
                break;
            case IPPROTO_UDP://UDP 匹配
                udp=(struct udphdr *)ptr;
                if(ntohs(udp->uh_sport)==pf.port||ntohs(udp->uh_dport)==pf.port)
                    return 0;
                break;
        }
    }
    return 1;
}
```

此函数和过滤指定 IP 地址和端口号相关。传入 Ethernet 数据包头部，检查对应的端口号选项是否打开，若未打开返回 false；若打开，检查数据包的源端口号和目的端口号是否等于指定的端口号，若不同则返回 false；其他情况则返回 true，表示数据包过滤完毕。

(4) int find_ne(u_char *d)

```
int find_ne(u_char *d){
    struct ne* neptr=&nenode;
    while(neptr->next!=NULL){
        if(neptr->a[0]==d[0]&&neptr->a[1]==d[1]&&neptr->a[2]==d[2]&&neptr->a[3]==d[3]&&neptr->
a[4]==d[4]&&neptr->a[5]==d[5])
            return 1;
        else
            neptr=neptr->next;
    }
    for(int i=0;i<6;i++){
        neptr->a[i]=d[i];
    }
    neptr->next=(struct ne*)malloc(sizeof(struct ne));
    neptr=neptr->next;
```

```
    neptr->next=NULL;
    return 0;
}
```

此函数的功能是发现网元，传入参数是字符串地址。

(5) void p_count(struct ether_header *eth)

```
void p_count(struct ether_header *eth){
    int i;
    for(i=0;i<6;i++){
        if(eth->ether_dhost[0]!=0xff)
            break;
    }
    if(i==6){
        ct.macbroad++;
    }
    char *ptr=(char *)eth;
    ptr=ptr+sizeof(struct ether_header);
    struct ip *ip;
    struct icmp *icmp;
    struct udphdr *udp;
    if(ntohs(eth->ether_type)==ETHERTYPE_IP){
        ct.ip++;
        ip=(struct ip *)ptr;
        ct.ipbyte+=ntohs(ip->ip_len)-ip->ip_hl*4;
        if((ntohl(ip->ip_dst.s_addr)|submask)==0xFFFFFFFF)//目的地址后面全 1
            ct.ipbroad++;
        switch(ip->ip_p){
            case IPPROTO_TCP://TCP 匹配
                ct.tcp++;
                break;
            case IPPROTO_UDP://UDP 匹配
                ct.udp++;
                break;
            case IPPROTO_ICMP://ICMP 匹配
                ct.icmp++;
                icmp=(struct icmp *)ptr;
                if(icmp->icmp_type==3)
                    ct.icmp_d++;
                if(icmp->icmp_type==5)
                    ct.icmp_r++;
                break;
        }
    }
}
```

此函数完成统计模块的统计功能。

(6) int getif1(char *ifname,int i)

```
int getif1(char *ifname,int i) {
    char bad_if[6][6]= {"lo:","lo","stf","gif","dummy","vmnet"};
    struct if_nameindex* ifn=if_nameindex();
    if(ifn == NULL) {
        return -1;
    }
    for(int j=0;j<6;j++){
        if (strcmp(ifn[i].if_name,bad_if[j]) == 0){
            i++;
            if(ifn[i].if_index==0){
                if_freenameindex(ifn);
                return -1;
            }
            j=0;
        }
    }
    strcpy(ifname,ifn[i].if_name);
    if_freenameindex(ifn);
    return i;
}
```

此函数主要功能是获取网络接口名称。

(7) void make_packet(const char *src_ip, int src_port, const char *dst_ip, int dst_port, const char *data)

```
void make_packet(const char *src_ip, int src_port, const char *dst_ip, int dst_port, const char
*data){
    struct iphdr* ip_header;
    struct tcphdr* tcp_header;
    struct sockaddr_in dst_addr;
    socklen_t sock_addrlen=sizeof(struct sockaddr_in);

    int data_len=strlen(data);
    int ip_packet_len=sizeof(struct iphdr)+sizeof(struct tcphdr)+data_len;
    char buf[ip_packet_len];
    int on=1;
    bzero(&dst_addr,sock_addrlen);
    dst_addr.sin_family=PF_INET;
    dst_addr.sin_addr.s_addr=inet_addr(dst_ip);
    dst_addr.sin_port=htons(dst_port);
```

```
    int sockfd=socket(PF_INET,SOCK_RAW,IPPROTO_TCP);
    if(sockfd<=0)
    {
        perror("socket create error:");
        exit(1);
    }
    if(setsockopt(sockfd,IPPROTO_IP,IP_HDRINCL,&on,sizeof(on))==-1)
    {
        perror("socket config error:");
        exit(1);
    }

    ip_header=fill_iphead(src_ip,dst_ip,ip_packet_len);
    tcp_header = fill_tcphead(src_port, dst_port);
    bzero(buf,ip_packet_len);
    memcpy(buf,ip_header,sizeof(struct iphdr));
    memcpy(buf+sizeof(struct iphdr),tcp_header,sizeof(struct tcphdr));
    memcpy(buf+sizeof(struct iphdr)+sizeof(struct tcphdr),data,data_len);
    int ret=sendto(sockfd,buf,ip_packet_len,0,(struct sockaddr*)&dst_addr,sock_addrlen);
    if(ret<0)
        perror("sendto()");
    close(sockfd);
    free(ip_header);
        free(tcp_header);
}
```

此函数完成对 TCP 畸形数据包的构造。协助该函数完成的函数还有：

```
struct iphdr* fill_iphead(const char* src_ip,const char* dst_ip,int ip_packet_len);
struct tcphdr* fill_tcphead(int src_port,int dst_port);
```

它们分别完成对 IP 头部和 TCP 头部的填充。

(8) main()函数

首先是处理命令行的各个参数，并将命令行的参数逐个分离出来。并根据对应的命令行的参数设置对应的结构体标志位。

```
int opt;
    bool loop=true;
    while((opt=getopt(argc,argv,"aedhmp:i:f:"))!=-1){
        switch(opt){
            case 'm':
                loop=false;
                //make_packet("100.101.102.103",1234,"192.168.0.106",7788,"hahahaha");
                make_packet(argv[optind],atoi(argv[optind+1]),argv[optind+2],atoi(argv[optind+3
),argv[optind+4]);
                printf("send successfully!\n");
                break;
```

```
            case 'a':
                f.a=true;//opt[ALL]=ON;
                break;
            case 'e':
                f.e=true;//opt[ETHER]=ON;
                break;
            case 'd':
                f.d=true;//opt[DUMP]=ON;
                break;
            case 'h':
                help();
                exit(0);
            case 'i':
                f.i=true;
                if(strlen(optarg)<IFRLEN){
                    strcpy(f.ifname,optarg);
                }else{
                    printf("the size of the ifname is too big\n");
                    exit(1);
                }
                break;
            case 'p':
                f.p=true;//opt[ARP]=OFF;opt[IP]=OFF;opt[TCP]=OFF;opt[UDP]=OFF;opt[ICMP]=OFF;
                optind--;
                while(argv[optind]!=NULL&&argv[optind][0]!='-'){
                    if(strcmp(argv[optind],"arp")==0)
                        p.arp=true;//opt[ARP]=ON;
                    else if(strcmp(argv[optind],"ip")==0)
                        p.ip=true;//opt[IP]=ON;
                    else if(strcmp(argv[optind],"icmp")==0)
                        p.icmp=true;//opt[ICMP]=ON;
                    else if(strcmp(argv[optind],"tcp")==0)
                        p.tcp=true;//opt[TCP]=ON;
                    else if(strcmp(argv[optind],"udp")==0)
                        p.udp=true;//opt[UDP]=ON;
                    else{
                        printf("unknown parameter: %s",argv[optind]);
                        exit(1);
                    }
                    print(1);

                    optind++;
                }
                break;
```

```
            case 'f':
                f.f=true;//设定为过滤出需要的包
                optind--;
                while(argv[optind]!=NULL&&argv[optind][0]!='-'){
                    if(strcmp(argv[optind],"ip")==0){
                        pf.i=true;
                        optind++;
                    if(argv[optind]==NULL){
                        printf("input the ip address\n");
                        exit(1);
                    }
                    if(ip_atou(argv[optind],&pf.ip)==1){
                        printf("bad parameter of ip address:%s\n",argv[optind]);
                        exit(1);
                    }
                    //printf("pf.ip:%u\n",pf.ip);
                    }else if(strcmp(argv[optind],"port")==0){
                        pf.p=true;
                        optind++;
                        if(argv[optind]==NULL){
                            printf("input the port number\n");
                            exit(1);
                        }
                        pf.port=atoi(argv[optind]);
                        if(pf.port<=0){
                            printf("bad parameter of port:%s\n",argv[optind]);
                            exit(1);
                        }
                    }else{
                        printf("unknown parameter:%s",argv[optind]);
                        exit(1);
                    }
                    optind++;
                }
                break;
            case ':':
                printf("option need a value\n");
                break;
            case '?':
                printf("unknown option:%c\n",optopt);
                exit(1);
            default:
                printf("unknown error");
                exit(1);
```

```
        }
    }
    if(optind<argc){
        for(;optind<argc;optind++){
            printf("unknown:%s\n",argv[optind]);
        }
        printf(" \n");
        help();
        exit(1);
    }
```

如果没有指定网口名，则获取网口名：

```
    struct ifreq ifr_mask;
    char ifname[20];
    unsigned int* intptr;
    int ret=0;
    if(f.i==false){//get first ifname that has a submask//统计
        while(1){
            if((ret=getif1(ifname,ret))==-1){
                printf("can\'t get a network interface name that has a submask");
                exit(1);
            }
            memset(&ifr_mask, 0, sizeof(ifr_mask));
            strcpy(ifr_mask.ifr_name,ifname);
            if(ioctl(s,SIOCGIFNETMASK,&ifr_mask)< 0){
                ret++;
                continue;
            }else{
                printf("Listen to the network interface:%s\n",ifname);
                intptr=(unsigned int*)&(ifr_mask.ifr_netmask);//int 32bit
                submask=intptr[1];
                break;
            }
        }

    }
```

若指定了网口名：

```
    struct ifreq interface;
    if(f.i==true){
        memset(&interface, 0, sizeof(interface));
        strncpy(interface.ifr_ifrn.ifrn_name,f.ifname,strlen(f.ifname));
        if(setsockopt(s,SOL_SOCKET,SO_BINDTODEVICE,(char *)&interface,sizeof(interface))<0){
            printf("network interface %s bind failed\n",f.ifname);
            exit(1);
```

```
        }
        if(ioctl(s,SIOCGIFNETMASK,&interface)< 0){
            printf("get submask failed\n");
            exit(1);
        }
        printf("Listen to the network interface:%s\n",f.ifname);
        intptr=(unsigned int*)&(interface.ifr_netmask);//int 32bit
        submask=intptr[1];
    }
```

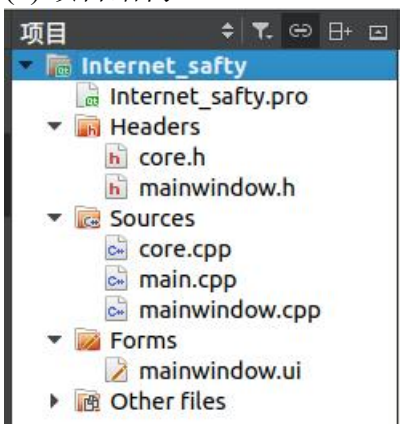
进入 while 循环抓包：

```
    while(true){
        if((len=read(s,buff,MAXSIZE))<0){
            perror(" read");
            exit(1);
        }
```

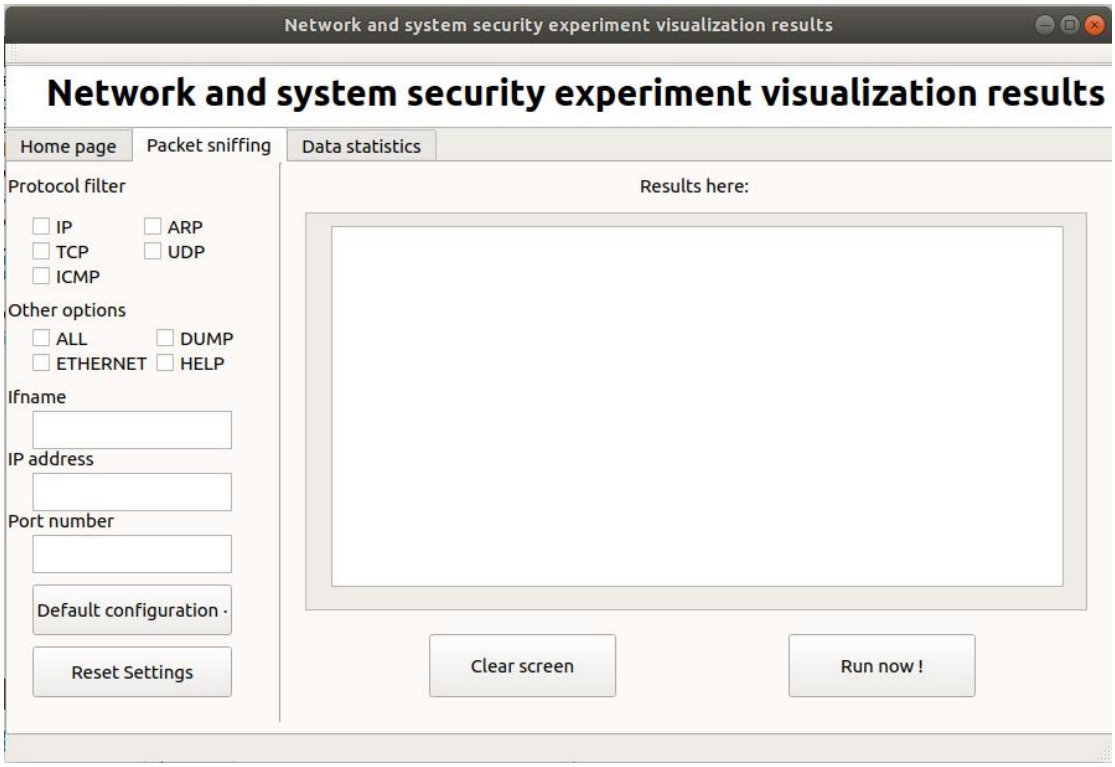
最后就是根据对应的数据包协议类型来打印输出语句即可。

2. QT 可视化开发部分

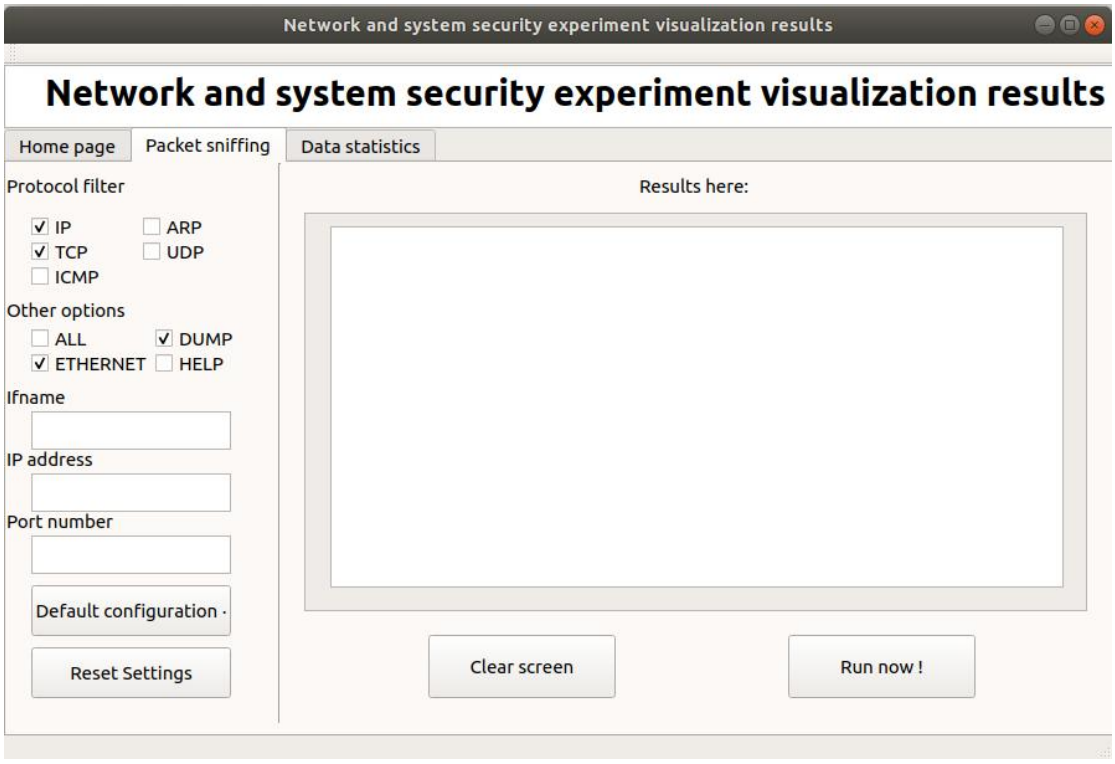
(1) 项目结构



(2) 项目主页面一



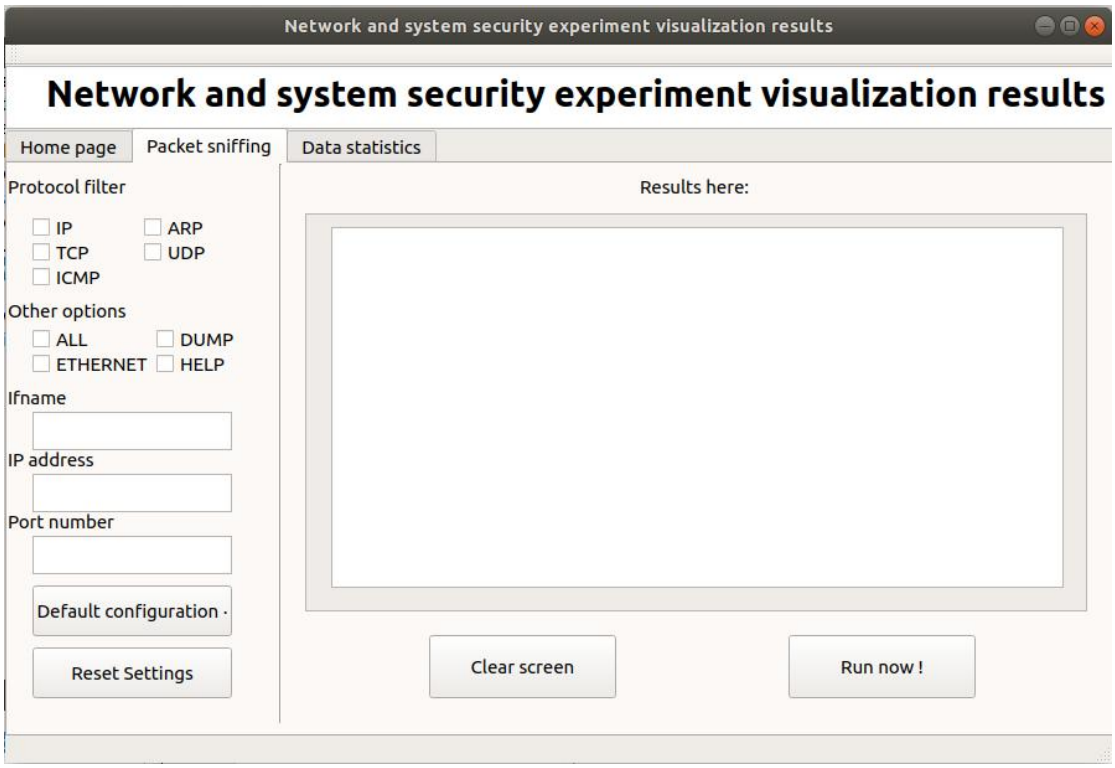
点击 default configuration 按钮实现数据包嗅探的默认配置：



此功能依赖按钮的信号槽函数：

```
void MainWindow::on_pb_default_clicked()
{
    ui->cb_help->setChecked(false);
    ui->cb_ip->setChecked(true);
    ui->cb_arp->setChecked(false);
    ui->cb_tcp->setChecked(true);
    ui->cb_udp->setChecked(false);
    ui->cb_icmp->setChecked(false);
    ui->cb_all->setChecked(false);
    ui->cb_dump->setChecked(true);
    ui->cb_eth->setChecked(true);
}
```

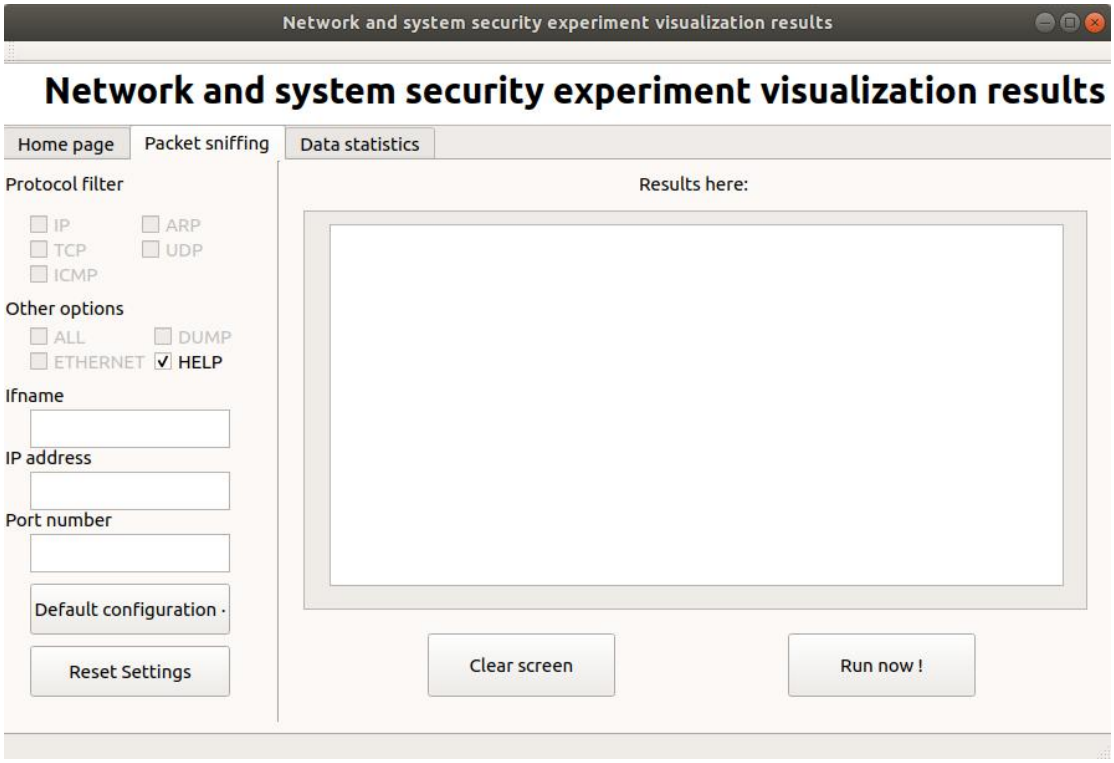
点击 reset settings 按钮实现设置的重置：



此功能依赖按钮的信号槽函数：

```
void MainWindow::on_pb_reset_clicked()
{
    ui->cb_help->setChecked(false);
    ui->cb_ip->setChecked(false);
    ui->cb_arp->setChecked(false);
    ui->cb_tcp->setChecked(false);
    ui->cb_udp->setChecked(false);
    ui->cb_icmp->setChecked(false);
    ui->cb_all->setChecked(false);
    ui->cb_dump->setChecked(false);
    ui->cb_eth->setChecked(false);
}
```

若 HELP 选项处于激活状态会让其他选项变得不可选中：



此操作依赖对于按钮 HELP 状态的监听：

```
void MainWindow::on_cb_help_stateChanged(int arg1)
{
    if (arg1 == Qt::Checked) // "选中"
    {
        ui->cb_ip->setChecked(false);
        ui->cb_arp->setChecked(false);
        ui->cb_tcp->setChecked(false);
        ui->cb_udp->setChecked(false);
        ui->cb_icmp->setChecked(false);
        ui->cb_all->setChecked(false);
        ui->cb_dump->setChecked(false);
        ui->cb_eth->setChecked(false);

        ui->cb_ip->setEnabled(false);
        ui->cb_arp->setEnabled(false);
        ui->cb_tcp->setEnabled(false);
        ui->cb_udp->setEnabled(false);
        ui->cb_icmp->setEnabled(false);
        ui->cb_all->setEnabled(false);
        ui->cb_dump->setEnabled(false);
        ui->cb_eth->setEnabled(false);
        //选中执行函数
    }
    else                    // 未选中 - Qt::Unchecked
    {
        ui->cb_ip->setEnabled(true);
        ui->cb_arp->setEnabled(true);
        ui->cb_tcp->setEnabled(true);
        ui->cb_udp->setEnabled(true);
        ui->cb_icmp->setEnabled(true);
        ui->cb_all->setEnabled(true);
        ui->cb_dump->setEnabled(true);
        ui->cb_eth->setEnabled(true);
       //未选中执行函数
    }

}
```

点击 clear screen 会清除结果，此操作依赖按钮的信号槽函数：

```
void MainWindow::on_pb_clear_clicked()
{
    ui->tb_res->setText("");
}
```

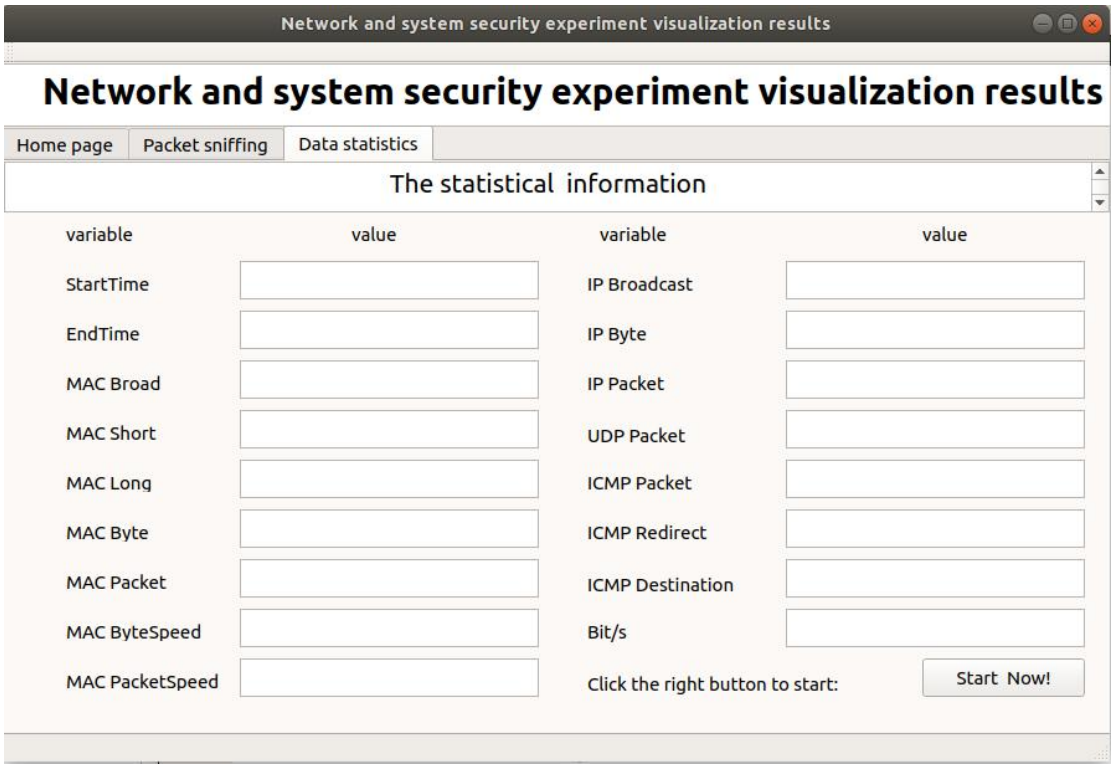
另外，由于抓包是一个死循环过程，在程序编写过程中会出现程序卡死的情况。这里采用子进程来解决这个问题：

点击 run now 按钮，开始循环抓包：

```
//Core_cpp core;//core.
pid_t child_pid;
child_pid = fork();

if(child_pid==0&&(!(*((bool*)shm))));
```

(3) 项目主页面二



点击 start now 按钮需要强行停止抓包过程。但是 fork()创建的子进程和父进程的参数是互不影响的，因此这里需要使用共享内存来解决这个问题。

```
void MainWindow::on_pushButton_clicked()
{
    *((bool*)shm)=true;
```

点击 start now 按钮后，对共享内存的布尔值进行修改。

首先声明子进程和父进程需要访问变量的公共 key：

```
#define KEY 100
```

调用 shmget 和 shmat 函数将原布尔值 stop 存入 void*变量 shm 中。

```
int shmid=shmget(KEY,sizeof(Core::stop),0666|IPC_CREAT);
shm=shmat(shmid,0,0);
*((bool*)shm)=false;
```

最后通过强制类型转换父子进程即可同时访问该值。

五、 实验结果

1. 命令行实验结果

(1) 编译源文件

```
neuljh@neuljh-virtual-machine:~/c/linux1/InternetAndSafty$ gcc -o testip2 test.c
neuljh@neuljh-virtual-machine:~/c/linux1/InternetAndSafty$
```

(2) 查看帮助信息

```
neuljh@neuljh-virtual-machine:~/c/linux1/InternetAndSafty$ sudo ./testip2 -h
[sudo] neuljh 的密码:
usage: ./ipdump [-aedht] [-p protocols] [-i ifrname] [-f filters]
protocols: arp ip icmp tcp udp
filters: ip <IP address> port <PORT number>
default: ./ipdump -p arp ip icmp tcp udp
```

(3) 默认规则捕获数据包

```
neuljh@neuljh-virtual-machine:~/c/linux1/InternetAndSafty$ sudo ./testip
Listen to the network interface:ens33


Packet Number:1
Protocal:IP
+--------+--------+--------+--------+
|IV:4|HL:05|T:00000000|T-Length:1242|
+--------+--------+--------+--------+
|Identifier: 45753|FF:0D0|FO:      0|
+--------+--------+--------+--------+
|TTL:  64|Pro:   6|Checksum:   35568|
+--------+--------+--------+--------+
|Source IP Address:  192.168.176.132|
+--------+--------+--------+--------+
|Dest   IP Address:  211.162.179.164|
+--------+--------+--------+--------+
Protocol:TCP
+--------+--------+--------+--------+
|Source Port:48228|Dest Port:    443|
+--------+--------+--------+--------+
|sequnce Number:          2737108929|
+--------+--------+--------+--------+
|Acknowlegement Number:    190378062|
+--------+--------+--------+--------+
|Do  5|RR|F:0AP000|Window Size:62780|
+--------+--------+--------+--------+
|Cheksum:    48510|Urgent-P:       0|
+--------+--------+--------+--------+
```

由于默认数据包大多数都是 IP 和 TCP 数据包，因此后面不对 IP 和 TCP 进行专门的过滤实验。

(4) 捕获全部的数据包

```
neuljh@neuljh-virtual-machine:~/c/linux1/InternetAndSafty$ sudo ./testip -a
Listen to the network interface:ens33


 Packet Number:1
 Protocal:IP
 +--------+--------+--------+--------+
 |IV:4|HL:05|T:00000000|T-Length: 455|
 +--------+--------+--------+--------+
 |Identifier: 46067|FF:0D0|FO:      0|
 +--------+--------+--------+--------+
 |TTL:  64|Pro:   6|Checksum:   36041|
 +--------+--------+--------+--------+
 |Source IP Address:  192.168.176.132|
 +--------+--------+--------+--------+
 |Dest   IP Address:  211.162.179.164|
 +--------+--------+--------+--------+
 Protocol:TCP
 +--------+--------+--------+--------+
 |Source Port:48228|Dest Port:    443|
 +--------+--------+--------+--------+
 |sequnce Number:          2737216581|
 +--------+--------+--------+--------+
 |Acknowlegement Number:    190436344|
 +--------+--------+--------+--------+
 |Do  5|RR|F:0AP000|Window Size:65535|
 +--------+--------+--------+--------+
 |Cheksum:    49246|Urgent-P:       0|
 +--------+--------+--------+--------+
```

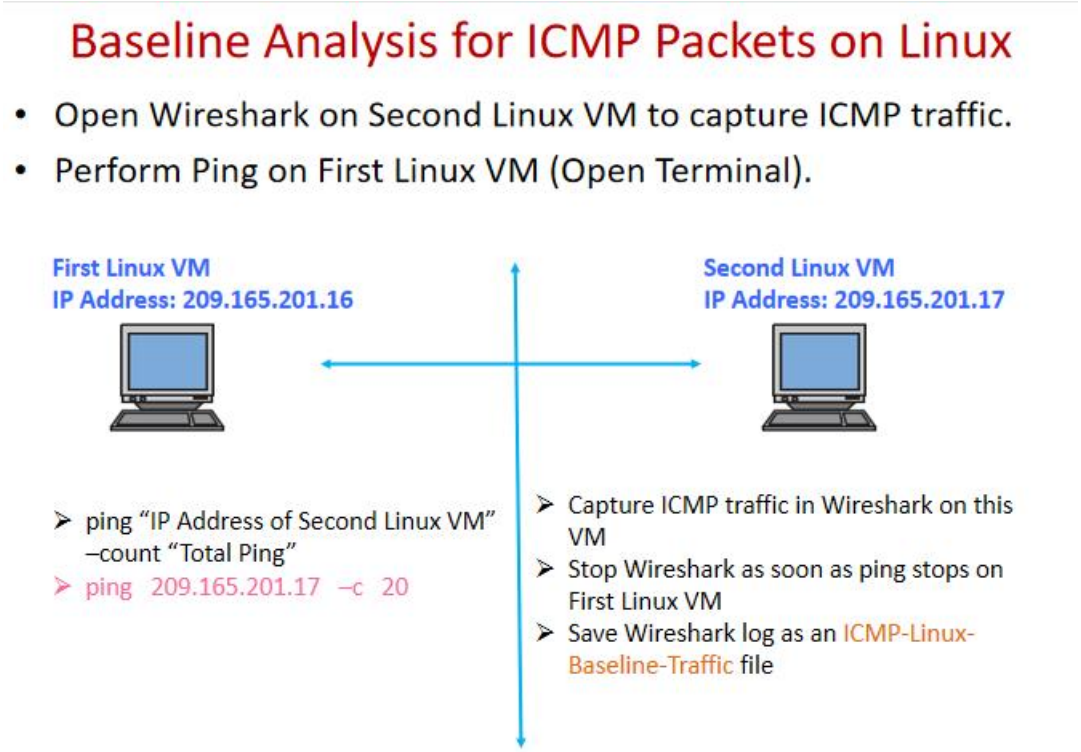
(5) 捕获 ARP 数据包并显示 Ethernet 头部和 DUMP

```
neuljh@neuljh-virtual-machine:~/c/linux1/InternetAndSafty$ sudo ./testip -e -d -p arp
Listen to the network interface:ens33


 Packet Number:1
 Ethernet Frame:
 +---------------+---------------+---------------+
 |Destination MAC Adress:          ff:ff:ff:ff:ff:ff|
 +---------------+---------------+---------------+
 |Source MAC Adress:               00:50:56:c0:00:08|
 +---------------+---------------+---------------+
 |E-Type:   0x0806|
 +---------------+
 Protocal:ARP
 +--------+--------+--------+--------+
 |H-Ty: 1(Ether)   |P:0x0800(IP)     |
 +--------+--------+--------+--------+
 |H-len: 6|P-Len: 4|op:1(ARP Request)|
 +--------+--------+--------+--------+--------+--------+
 |Source MAC Adress:                 00:50:56:c0:00:08|
 +--------+--------+--------+--------+--------+--------+
 |Source IP Address:    192.168.176.1|
 +--------+--------+--------+--------+--------+--------+
 |Destination MAC Adress:            00:00:00:00:00:00|
 +--------+--------+--------+--------+--------+--------+
 |Dest   IP Address:    192.168.176.2|
 +--------+--------+--------+--------+
 Frame Dump:
ffff ffff ffff 0050 56c0 0008 0806 0001     .......PV.......
0800 0604 0001 0050 56c0 0008 c0a8 b001     .......PV.......
0000 0000 0000 c0a8 b002 0000 0000 0000     ................
0000 0000 0000 0000 0000 0000               ............
```

(6) 捕获 ICMP 数据包

在实验测试中，我发现捕获的数据包基本上没有 ICMP 和 UDP 包，因此这里我们采用另一台主机和本主机通信的方式来捕获对应的 ICMP 数据包。参考下图逻辑：



首先服务端(本主机)输入以下命令，循环捕获 ICMP 数据包：

```
neuljh@neuljh-virtual-machine:~/c/linux1/InternetAndSafty$ sudo ./testip -p icmp
Listen to the network interface:ens33
```

客户端主机打开终端，向服务端主机(IP 地址为 192.168.176.132)发送 20 个 ICMP 数据包：

```
[neu_ljh@localhost ~]$ ping 192.168.176.132 -c 20
PING 192.168.176.132 (192.168.176.132) 56(84) bytes of data.
64 bytes from 192.168.176.132: icmp_seq=1 ttl=64 time=0.325 ms
64 bytes from 192.168.176.132: icmp_seq=2 ttl=64 time=0.532 ms
64 bytes from 192.168.176.132: icmp_seq=3 ttl=64 time=0.435 ms
64 bytes from 192.168.176.132: icmp_seq=4 ttl=64 time=0.312 ms
64 bytes from 192.168.176.132: icmp_seq=5 ttl=64 time=0.433 ms
64 bytes from 192.168.176.132: icmp_seq=6 ttl=64 time=0.491 ms
64 bytes from 192.168.176.132: icmp_seq=7 ttl=64 time=0.629 ms
64 bytes from 192.168.176.132: icmp_seq=8 ttl=64 time=0.604 ms
64 bytes from 192.168.176.132: icmp_seq=9 ttl=64 time=0.357 ms
64 bytes from 192.168.176.132: icmp_seq=10 ttl=64 time=1.40 ms
64 bytes from 192.168.176.132: icmp_seq=11 ttl=64 time=0.536 ms
64 bytes from 192.168.176.132: icmp_seq=12 ttl=64 time=0.388 ms
64 bytes from 192.168.176.132: icmp_seq=13 ttl=64 time=0.553 ms
64 bytes from 192.168.176.132: icmp_seq=14 ttl=64 time=0.251 ms
64 bytes from 192.168.176.132: icmp_seq=15 ttl=64 time=0.417 ms
64 bytes from 192.168.176.132: icmp_seq=16 ttl=64 time=0.279 ms
64 bytes from 192.168.176.132: icmp_seq=17 ttl=64 time=0.267 ms
64 bytes from 192.168.176.132: icmp_seq=18 ttl=64 time=0.261 ms
64 bytes from 192.168.176.132: icmp_seq=19 ttl=64 time=0.251 ms
64 bytes from 192.168.176.132: icmp_seq=20 ttl=64 time=0.468 ms

--- 192.168.176.132 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19010ms
rtt min/avg/max/mdev = 0.251/0.459/1.406/0.248 ms
```

展示部分捕获数据包：

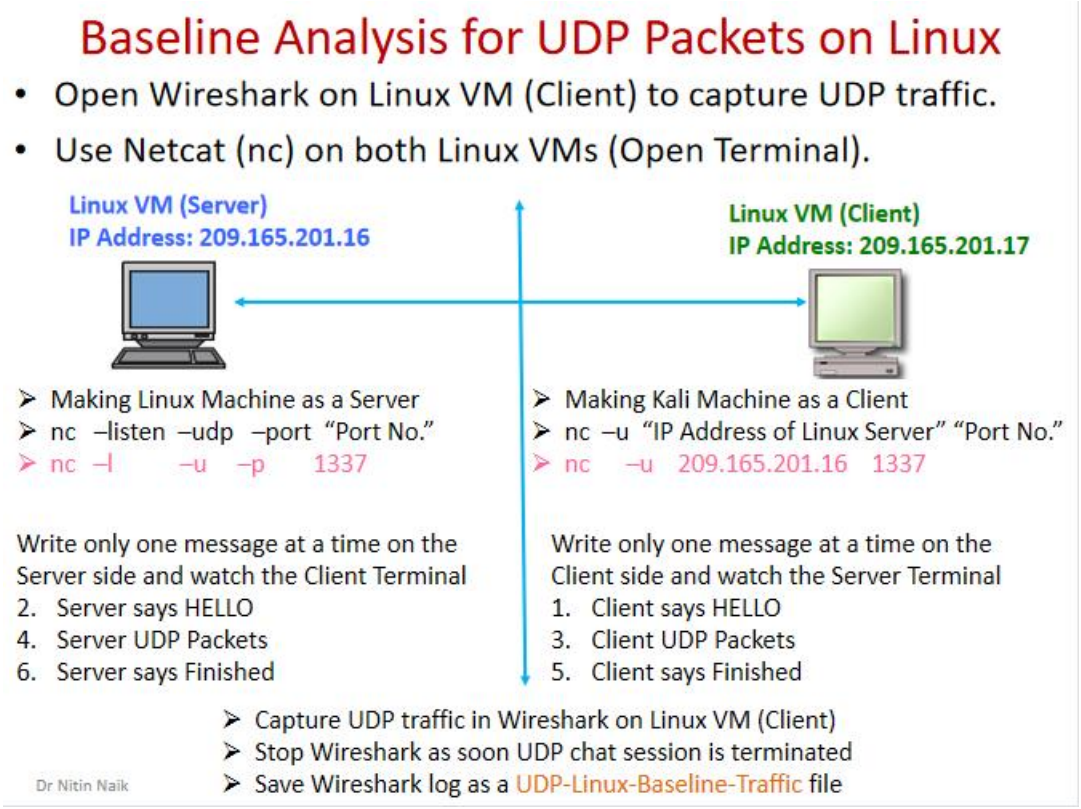
```
Packet Number:449
Protocal:ICMP(Destination Unreachable)
+--------+--------+--------+--------+
|Type:  3|Code:  3|Checksum:   57162|
+--------+--------+--------+--------+
|Unsed:         0|
+--------+--------+--------+--------+
Protocal:IP
+--------+--------+--------+--------+
|IV:4|HL:05|T:00000000|T-Length: 120|
+--------+--------+--------+--------+
|Identifier: 29218|FF:000|FO:      0|
+--------+--------+--------+--------+
|TTL: 128|Pro:  17|Checksum:   59002|
+--------+--------+--------+--------+
|Source IP Address:    192.168.176.2|
+--------+--------+--------+--------+
|Dest   IP Address:  192.168.176.132|
+--------+--------+--------+--------+
```

```
Packet Number:45
Protocal:ICMP(Echo Request)
+--------+--------+--------+--------+
|Type:  8|Code:  0|Checksum:   58107|
+--------+--------+--------+--------+
|Identifi:   23009|Seq  Num:       7|
+--------+--------+--------+--------+


Packet Number:46
Protocal:ICMP(Echo Reply)
+--------+--------+--------+--------+
|Type:  0|Code:  0|Checksum:   60155|
+--------+--------+--------+--------+
|Identifi:   23009|Seq  Num:       7|
+--------+--------+--------+--------+
```

(7) 捕获 UDP 数据包

在实验测试中，我发现捕获的数据包基本上没有 ICMP 和 UDP 包，因此这里我们采用另一台主机和本主机通信的方式来捕获对应的 UDP 数据包。参考下图逻辑：



首先建立两个主机的 UDP 通信。

服务端(IP 地址为 192.168.176.132)输入以下命令：

```
neuljh@neuljh-virtual-machine:~$ su root
密码:
root@neuljh-virtual-machine:/home/neuljh# nc -l -u -p 80
```

客户端输入以下命令：

```
[neu_ljh@localhost ~]$ nc -u 192.168.176.132 80
```

服务端循环监听捕获 UDP 数据包：

```
neuljh@neuljh-virtual-machine:~/c/linux1/InternetAndSafty$ sudo ./testip -p udp
```

服务端发送消息 1，客户端回复消息 2：

```
root@neuljh-virtual-machine:/home/neuljh# nc -l -u -p 80
1
2
```

展示部分捕获 UDP 数据包：

```
Packet Number:17
Protocol:UDP
+--------+--------+--------+--------+
|Source Port:32790|Dest Port:     53|
+--------+--------+--------+--------+
|Length:        66|Checksum:   65161|
+--------+--------+--------+--------+


Packet Number:18
Protocol:UDP
+--------+--------+--------+--------+
|Source Port:32790|Dest Port:     53|
+--------+--------+--------+--------+
|Length:        66|Checksum:   65161|
+--------+--------+--------+--------+
```

```
Packet Number:45
Protocol:UDP
+--------+--------+--------+--------+
|Source Port:33612|Dest Port:     80|
+--------+--------+--------+--------+
|Length:        10|Checksum:   26588|
+--------+--------+--------+--------+


Packet Number:46
Protocol:UDP
+--------+--------+--------+--------+
|Source Port:   80|Dest Port:  33612|
+--------+--------+--------+--------+
|Length:        10|Checksum:   26844|
+--------+--------+--------+--------+
```

(8) 显示统计信息

为了方便,这里直接查看(7)中的数据包统计信息：

```
The statistical  information
varibale         values
StartTime        Thu Dec 22 17:20:00 2022
EndTime          Thu Dec 22 17:20:21 2022
MAC Broad        0
MAC Short        34
MAC Long         0
MAC Byte         20685
MAC Packet       74
Bit/s            7880
MAC ByteSpeed    985
MAC PacketSpeed  3
IP Broadcast     0
IP Byte          18005
IP Packet        70
UDP Packet       14
ICMP Packet      0
ICMP Redirect    0
ICMP Destination 0

Find these mac address:
00:50:56:e7:ee:6b
00:0c:29:fc:1b:76
00:00:00:00:00:00
00:0c:29:5d:c8:32
```

(9)选择指定 IP 地址和端口号

```
neuljh@neuljh-virtual-machine:~/c/linux1/InternetAndSafty$ sudo ./testip -f ip  192.168.176.132 port 80
Listen to the network interface:ens33


 Packet Number:1
 Protocal:IP
 +--------+--------+--------+--------+
 |IV:4|HL:05|T:00000000|T-Length:  60|
 +--------+--------+--------+--------+
 |Identifier: 19882|FF:0D0|FO:      0|
 +--------+--------+--------+--------+
 |TTL:  64|Pro:   6|Checksum:    1078|
 +--------+--------+--------+--------+
 |Source IP Address:  192.168.176.132|
 +--------+--------+--------+--------+
 |Dest   IP Address:   185.125.190.49|
 +--------+--------+--------+--------+
 Protocol:TCP
 +--------+--------+--------+--------+
 |Source Port:53040|Dest Port:     80|
 +--------+--------+--------+--------+
 |sequnce Number:          2976666506|
 +--------+--------+--------+--------+
 |Acknowlegement Number:            0|
 +--------+--------+--------+--------+
 |Do 10|RR|F:0000S0|Window Size:64240|
 +--------+--------+--------+--------+
 |Cheksum:    38807|Urgent-P:       0|
 +--------+--------+--------+--------+
```

(10)发送畸形 TCP 数据包

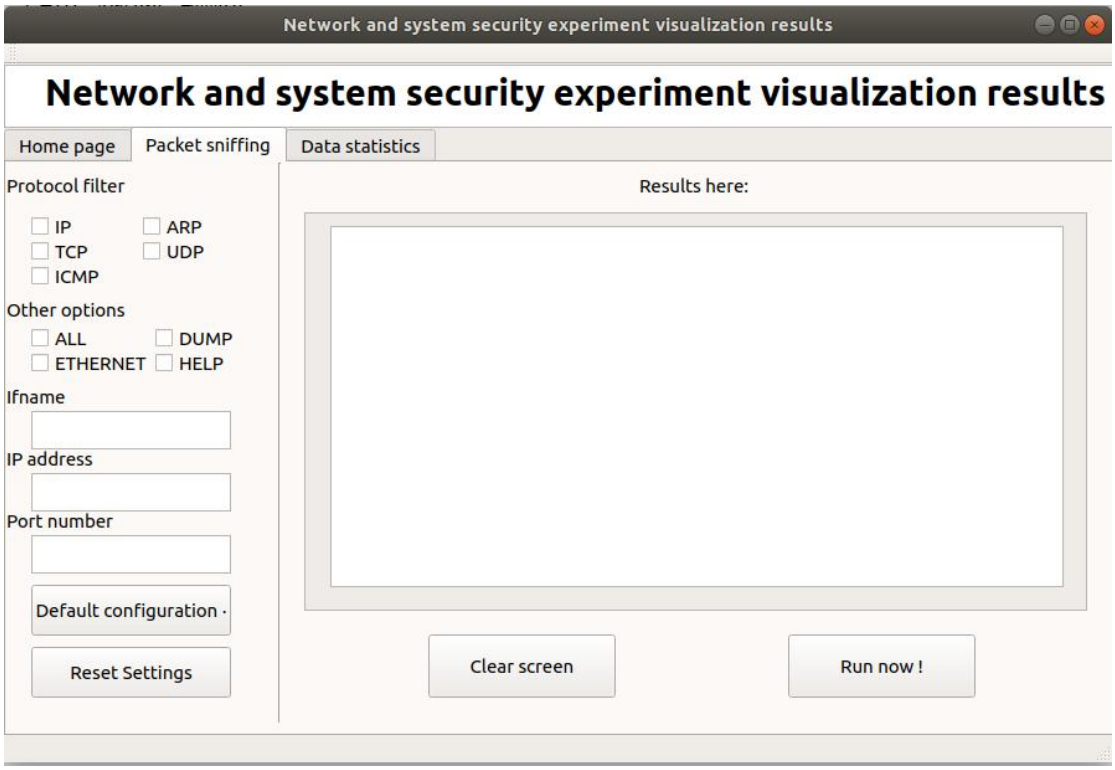
在命令行指定该数据包的源 IP 地址、源端口号、目的 IP 地址和目的端口号。

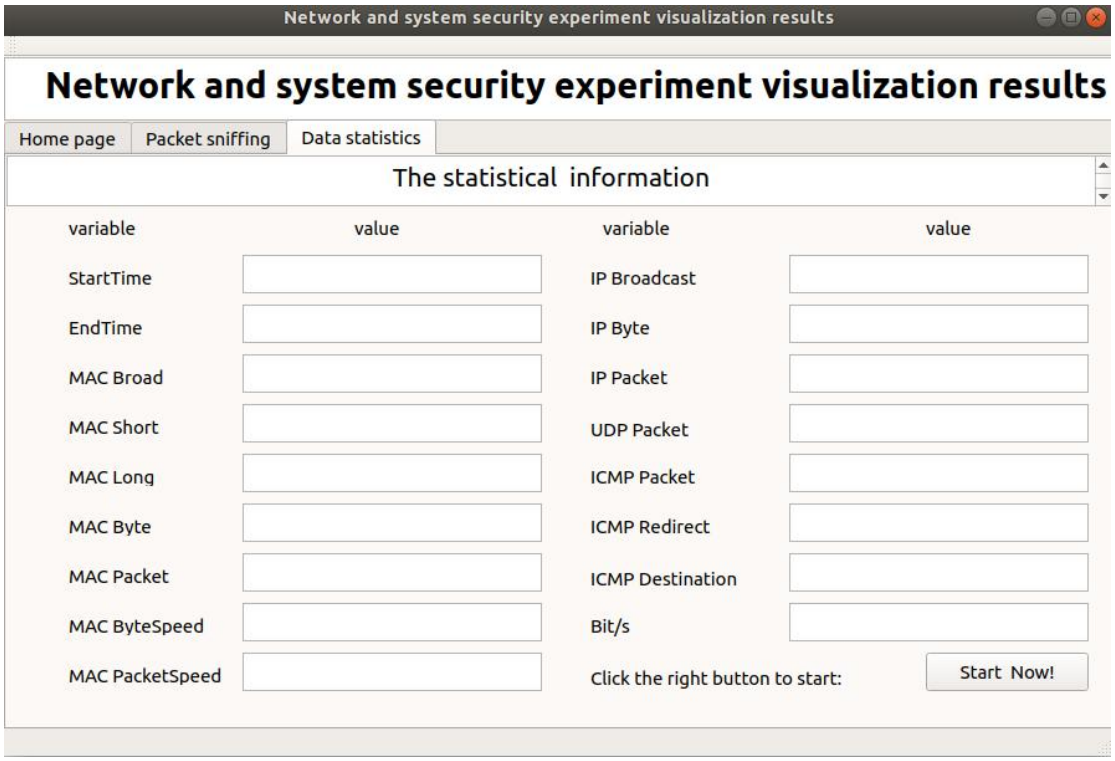
```
root@neuljh-virtual-machine:/home/neuljh/c/linux1/InternetAndSafty# ./testip2 -m 100.101.102.103 1234 127.0.0.1 7788 hello
send successfully!
```

畸形 TCP 数据包发送成功。

2. QT 可视化结果

(1) 主页面展示

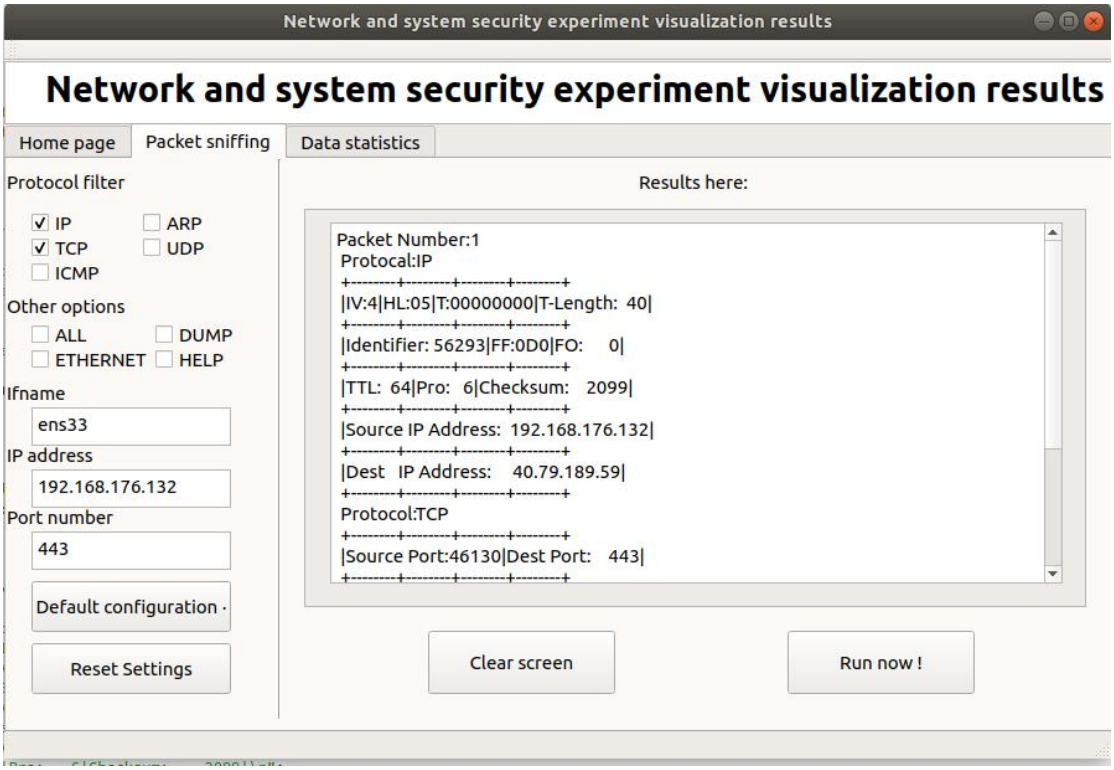

Network and system security experiment visualization results
Network and system security experiment visualization results
Home page
Packet sniffing
Data statistics
Protocol filter
IP
ARP
TCP
UDP
ICMP
Other options
ALL
DUMP
ETHERNET
HELP
Ifname
IP address
Port number
Default configuration
Reset Settings
Results here:
Clear screen
Run now !


Network and system security experiment visualization results
Network and system security experiment visualization results
Home page
Packet sniffing
Data statistics
The statistical information
variable
value
variable
value
StartTime
EndTime
MAC Broad
MAC Short
MAC Long
MAC Byte
MAC Packet
MAC ByteSpeed
MAC PacketSpeed
IP Broadcast
IP Byte
IP Packet
UDP Packet
ICMP Packet
ICMP Redirect
ICMP Destination
Bit/s
Click the right button to start:
Start Now!

(2) 主要功能

支持用户自定义的网络数据包嗅探。运行结果：

首先在命令行进入超级管理员模式，并允许程序：

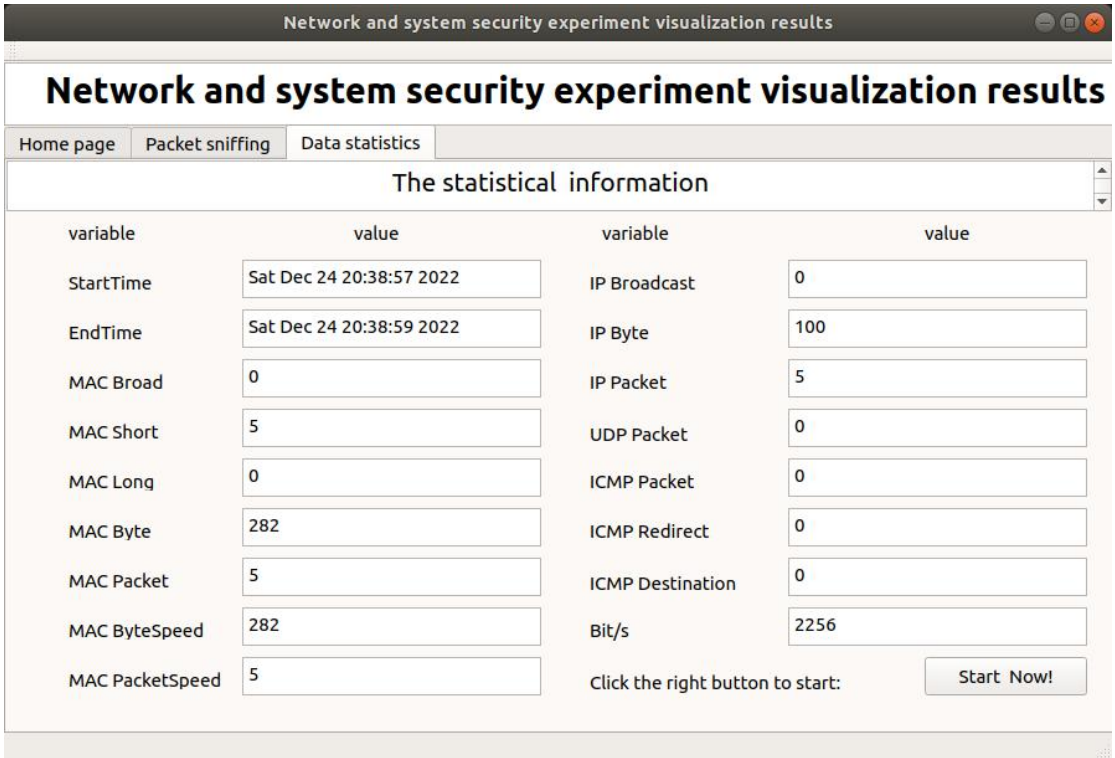


当然在命令行也有对应数据包输出，仅展示部分：

```
Packet Number:4
 Protocal:IP
+--------+--------+--------+--------+
|IV:4|HL:05|T:00000000|T-Length: 703|
+--------+--------+--------+--------+
|Identifier: 32796|FF:0D0|FO:      0|
+--------+--------+--------+--------+
|TTL:  64|Pro:   6|Checksum:   13963|
+--------+--------+--------+--------+
|Source IP Address:  192.168.176.132|
+--------+--------+--------+--------+
|Dest   IP Address:   34.117.237.239|
+--------+--------+--------+--------+
Protocol:TCP
+--------+--------+--------+--------+
|Source Port:42914|Dest Port:    443|
+--------+--------+--------+--------+
|sequnce Number:          1405299041|
+--------+--------+--------+--------+
|Acknowlegement Number:   1766546312|
+--------+--------+--------+--------+
|Do  5|RR|F:0AP000|Window Size:64240|
+--------+--------+--------+--------+
|Cheksum:    34920|Urgent-P:       0|
+--------+--------+--------+--------+
```

```
Packet Number:5
 Protocal:IP
+--------+--------+--------+--------+
|IV:4|HL:05|T:00000000|T-Length:  40|
+--------+--------+--------+--------+
|Identifier: 28749|FF:000|FO:      0|
+--------+--------+--------+--------+
|TTL: 128|Pro:   6|Checksum:   18673|
+--------+--------+--------+--------+
|Source IP Address:   34.117.237.239|
+--------+--------+--------+--------+
|Dest   IP Address:  192.168.176.132|
+--------+--------+--------+--------+
Protocol:TCP
+--------+--------+--------+--------+
|Source Port:  443|Dest Port:  42914|
+--------+--------+--------+--------+
|sequnce Number:          1766546312|
+--------+--------+--------+--------+
|Acknowlegement Number:   1405299704|
+--------+--------+--------+--------+
|Do  5|RR|F:0A0000|Window Size:64240|
+--------+--------+--------+--------+
|Cheksum:    17765|Urgent-P:       0|
+--------+--------+--------+--------+
```

切换页面到数据统计模块，点击 start now 按钮进行统计：



可视化的功能全部都是从单个 c 文件一步一步移植过来的，因此功能十分相似在报告里就不重复展示了。

六、 实验总结

通过本次实验，我掌握了如何利用 AF 地址簇建立的网络数据包捕获接口捕获流经本网卡的所有原始数据包，并在此基础上对捕获的数据包进行统计分析。同时，知道了如何通过编写过滤条件，对网络数据包进行过滤，提取出所关心的网络数据。在实验过程中，不仅加深了我对 TCP/IP 协议族中常见协议的的理解，还提高了自身的编程能力，获益匪浅。

另外也是温习了 QT 编程，感觉获益良多。

最后就是感谢所有任课老师，老师们的辛勤付出让我也十分感激！

评价表格

考核标准	得分
（1）正确理解和掌握实验所涉及的概念和原理（20%）；	
（2）按实验要求合理设计数据结构和程序结构，运行结果正确（40%）；	
（3）实验过程中，具有严谨的学习态度和认真、踏实、一丝不苟的科学作风，实验报告规范；认真记录实验数据，原理及实验结果分析准确（40%）；	
合计	