实验二 socket 编程

一. 实验目的

（1） 熟悉用 linux 进行套接字网络编程

（2） 熟悉各个协议的组成

（3） 学习抓包和 ping 程序的实现思路

二. 数据结构说明

（1） MyWireShark.c

（a）ip 头类型

```
typedef struct Iphdr{
	unsigned char head_version_len;//4 bits for length of head and 4 bits for the version id
	unsigned char tos;//1 byte for service type tos
	unsigned short total_len;//2 bytes for total length
	unsigned short id;//2 bytes for identity
	unsigned short flags;//flags
	unsigned char ttl;//1 bytes time to live
	unsigned char proto;//1 bytes protocol
	unsigned short check_sum;//2 bytes check sum of ip head
	unsigned int src_ip;//4 bytes source address of ip
	unsigned int dest_ip;//4 bytes destination address of ip
}IP_HEADER;
```

这是一个描述 ip 头的结构体，对应于一个网络层 ip 协议

head_version_len 表示 4 bits 的头长度和 4bits 的版本号

tos 表示 type of service 即服务类型

total_len 表示总的 ip 包长度

id 表示标识符号

flag 表示分段标记和分段偏移

ttl 表示 time to live 生存期

proto 表示运输层使用的协议

check_sum 表示校验和

src_ip 和 dest_ip 分别表示源和目的 ip 地址

(b)icmp 头类型

```
typedef struct Icmphdr{
	unsigned char type;
	unsigned char code;
	unsigned short check_sum;
	unsigned short id;
	unsigned short seq_num;

}ICMP_HEADER;
```

这是描述 icmp 段的结构体

Type 是 icmp 类型，是 request 还是 reply

Code 是 icmp 的代码字段

Check_sum 是校验和

Id 是标识符

Seq_num 是序列号

（c）tcp 头类型

```
typedef struct Tcphdr{
	unsigned short src_port;
	unsigned short dest_port;
	unsigned int seq;
	unsigned int ack;
	unsigned char len_rest;
	unsigned char flag;
	unsigned short windows;
	unsigned short check_sum;
	unsigned short urgent_pointer;
}TCP_HEADER;
```

这是描述 tcp 头的结构体

src_port表示源端口号

dest_port表示目的端口号

seq表示序列号

ack表示确认号

len_rest;头部长度和保留字段

flag是标志位字段（U,A,P,R,S,F分别表示紧急指针、ack 有效、尽快交给应用层、重建连接、发起连接、释放连接

windows是窗口大小字段，用来控制流量，单位为字节数，即本机期望一次接受的字节数

check_sum 为tcp校验和

urgent_pointer 紧急指针字段，是偏移量，和序号字段相加表示紧急数据最后一个字节的序号

(d)udp 头类型

```
typedef struct Udphdrp{
	unsigned short src_port;
	unsigned short dest_port;
	unsigned int len;
	unsigned int check_sum;
}UDP_HEADER;
```

这是描述 udp 协议包头的结构体

src_port表示源端口号

dest_port表示目的端口号

len表示长度

check_sum表示校验和

（e）arp头类型

```
typedef struct Arphdr{
	unsigned short hardware_type;
	unsigned short protocol_type;
	unsigned char hardware_address_len;
```

```
        unsigned char protocol_address_len;
        unsigned short operation_field;
        unsigned char src_mac_addr[6];
        unsigned char src_ip_addr[4];
        unsigned char dest_mac_addr[6];
        unsigned char dest_ip_addr[4];
}ARP_HEADER;
```

这是一个描述arp协议头的结构体

hardware_type表示硬件类型

protocol_type表示协议类型

hardware_address_len表示MAC地址长度，单位字节

protocol_address_len表示ip地址长度，单位字节

operation_field表示是arp请求还是arp应答

src_mac_addr[6]表示源MAC地址

src_ip_addr[4];表示源ip地址

dest_mac_addr[6]表示目的MAC地址

dest_ip_addr[4]表示目的ip地址

（2） ping.c

(a) icmp头类型

```
struct ICMPHeader
{
    type;//icmp type
    unsigned char code;//icmp code
    unsigned short checksum;//check sum
    struct{
       unsigned short id;
       unsigned short sequence;
    }echo;
    unsigned char data[0];//ICMP data parts
};
```

这是描述 icmp 段的结构体

type 是 icmp 类型，是 request 还是 reply

code 是 icmp 的代码字段

check_sum 是校验和

id 是标识符

sequence 是序列号

data是数据部分

（b）

```
struct IPHeader
{
    unsigned char headerLen:4;
    unsigned char version:4;
    unsigned char tos; //service type
```

```
    unsigned short totalLen; //total length
    unsigned short id; //tag
    unsigned short flagOffset; //3 bits flag+13 bits offset
    unsigned char ttl; //time to live
    unsigned char protocol; //protocol
    unsigned short checksum; //check sum
    unsigned int srcIP; //source ip address
    unsigned int dstIP; //destination ip address
};
```

这是一个描述 ip 头的结构体，对应于一个网络层 ip 协议

headerLen 表示 4 bits 的头长度

version 表示 4bits 的版本号

tos 表示 type of service 即服务类型

totalLen 表示总的 ip 包长度

id 表示标识符号

flagOffset 表示分段标记和分段偏移

ttl 表示 time to live 生存期

protocol 表示运输层使用的协议

checksum 表示校验和

srcIP 和 destIP 分别表示源和目的 ip 地址

三．程序设计思路和流程

（1） MyWireShark

（a）创建套接字 sock_fd

（b）读取数据 recvfrom

（c）检查数据长度是否大于 42

（d）读取 ethernet 帧头以确定 MAC 源地址、MAC 目的地址和内部协议的选择

（e）调用 ip/arp/rarp 解析函数，解析 ip 包头/arp 包/rarp 包

（f）如果是 ip 包，解析后获得网络层信息以及选择传输层协议的选择，

（g）根据内部是 icmp/tcp/udp 协议调用相应的解析函数

（h）返回（b）步骤，循环执行抓包

(2)ping

（a）创建套接字 sock_fd

（b）配置 sockfd 基本设置

（c）解析 ping 的 ip 地址，如果是 host name 先转化为 ip 地址保存在 sockaddr_in 结构中

（d）调用发包函数 send_icmp_packet.完成 icmp 包的数据装填和 send

（e）调用接受函数 receive_icmp_packet 接受包，且使用了 select 非阻塞接受

（f）如果接受了标志不正确或者长度不正确的包就丢弃，如果 0.3s 没有接受到 reply 则提示 request 包难以到达目的地

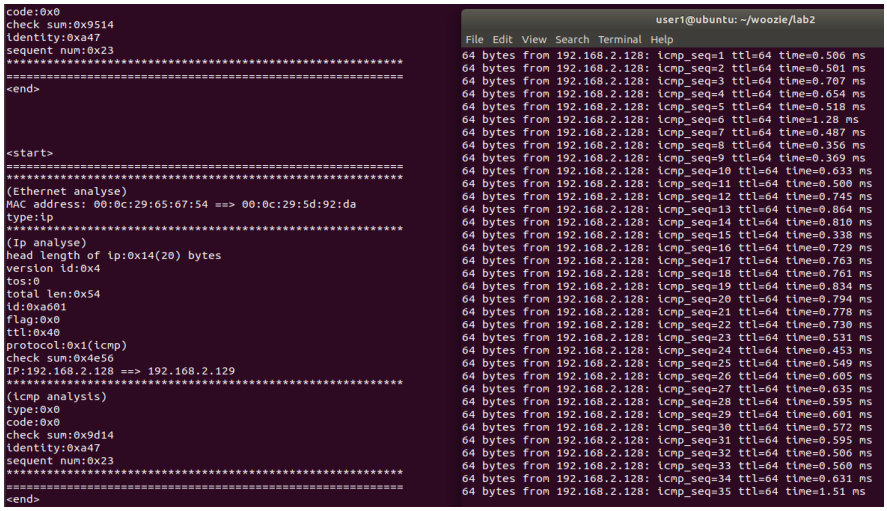
（g）如果成功接受了则解析该包，输出响应信息

（h）返回（d）步骤循环发包，接受包

四．运行结果截图

（1） MyWireShark

(a)ping 内网抓包截图



说明：每个抓包从 start 开始分析，到 end 接受，分成三个层次 parse 每个包，先是数据链路层，再到网路层和传输层

（b） 浏览 www.baidu.com 外网抓包结果

说明：可以从图中看到百度的 ip 和 mac 地址等信息

（c）更多协议的抓包结果

```
<start>
============================================================
************************************************************
(Ethernet analyse)
MAC address: 00:0c:29:5d:92:da ==> 00:0c:29:65:67:54
type:arp
(arp analysis)
hardware type:0x1
protocol type:0x800
hardward address len:0x6
protocol address len:0x4
operation field:0x2
source MAC address: 0:c:29:5d:92:da
destination MAC field: 0:c:29:65:67:54
source ip address: 192.168.2.129
destination ip address: 192.168.2.128
************************************************************
============================================================
<end>
```

说明：该图为 arp 协议截图（完成的协议有 ethernet、ip、arp、rarp、tcp、udp、icmp），其他协议函数可以查看 MyWireShark.c

（2） ping

（a）ping 程序和系统 ping 指令对比

```
user1@ubuntu:~/woozie/lab2$ sudo ./ping 192.168.2.128
PING 192.168.2.128 (192.168.2.128) 56(84) bytes of data.
64 bytes from 192.168.2.128: icmp_seq=1 ttl=64 time=0.310 ms
64 bytes from 192.168.2.128: icmp_seq=2 ttl=64 time=0.463 ms
64 bytes from 192.168.2.128: icmp_seq=3 ttl=64 time=0.359 ms
64 bytes from 192.168.2.128: icmp_seq=4 ttl=64 time=0.433 ms
64 bytes from 192.168.2.128: icmp_seq=5 ttl=64 time=0.551 ms
64 bytes from 192.168.2.128: icmp_seq=6 ttl=64 time=0.405 ms
64 bytes from 192.168.2.128: icmp_seq=7 ttl=64 time=0.421 ms
64 bytes from 192.168.2.128: icmp_seq=8 ttl=64 time=0.356 ms
64 bytes from 192.168.2.128: icmp_seq=9 ttl=64 time=0.412 ms
64 bytes from 192.168.2.128: icmp_seq=10 ttl=64 time=0.277 ms
64 bytes from 192.168.2.128: icmp_seq=11 ttl=64 time=0.853 ms
64 bytes from 192.168.2.128: icmp_seq=12 ttl=64 time=4.489 ms
64 bytes from 192.168.2.128: icmp_seq=13 ttl=64 time=0.346 ms
64 bytes from 192.168.2.128: icmp_seq=14 ttl=64 time=0.394 ms
64 bytes from 192.168.2.128: icmp_seq=15 ttl=64 time=0.555 ms
64 bytes from 192.168.2.128: icmp_seq=16 ttl=64 time=0.513 ms
64 bytes from 192.168.2.128: icmp_seq=17 ttl=64 time=0.412 ms
64 bytes from 192.168.2.128: icmp_seq=18 ttl=64 time=0.344 ms
64 bytes from 192.168.2.128: icmp_seq=19 ttl=64 time=0.353 ms
64 bytes from 192.168.2.128: icmp_seq=20 ttl=64 time=2.331 ms
64 bytes from 192.168.2.128: icmp_seq=21 ttl=64 time=0.388 ms
64 bytes from 192.168.2.128: icmp_seq=22 ttl=64 time=0.353 ms
64 bytes from 192.168.2.128: icmp_seq=23 ttl=64 time=0.402 ms
64 bytes from 192.168.2.128: icmp_seq=24 ttl=64 time=0.348 ms
64 bytes from 192.168.2.128: icmp_seq=25 ttl=64 time=0.560 ms
64 bytes from 192.168.2.128: icmp_seq=26 ttl=64 time=0.430 ms
64 bytes from 192.168.2.128: icmp_seq=27 ttl=64 time=0.415 ms
64 bytes from 192.168.2.128: icmp_seq=28 ttl=64 time=0.419 ms
64 bytes from 192.168.2.128: icmp_seq=29 ttl=64 time=0.406 ms
64 bytes from 192.168.2.128: icmp_seq=30 ttl=64 time=0.495 ms
```

```
user1@ubuntu:~$ ping 192.168.2.128
PING 192.168.2.128 (192.168.2.128) 56(84) bytes of data.
64 bytes from 192.168.2.128: icmp_seq=1 ttl=64 time=0.280 ms
64 bytes from 192.168.2.128: icmp_seq=2 ttl=64 time=0.403 ms
64 bytes from 192.168.2.128: icmp_seq=3 ttl=64 time=0.326 ms
64 bytes from 192.168.2.128: icmp_seq=4 ttl=64 time=2.52 ms
64 bytes from 192.168.2.128: icmp_seq=5 ttl=64 time=0.428 ms
64 bytes from 192.168.2.128: icmp_seq=6 ttl=64 time=2.46 ms
64 bytes from 192.168.2.128: icmp_seq=7 ttl=64 time=0.542 ms
64 bytes from 192.168.2.128: icmp_seq=8 ttl=64 time=1.32 ms
64 bytes from 192.168.2.128: icmp_seq=9 ttl=64 time=0.545 ms
64 bytes from 192.168.2.128: icmp_seq=10 ttl=64 time=0.435 ms
64 bytes from 192.168.2.128: icmp_seq=11 ttl=64 time=0.406 ms
64 bytes from 192.168.2.128: icmp_seq=12 ttl=64 time=0.475 ms
64 bytes from 192.168.2.128: icmp_seq=13 ttl=64 time=0.503 ms
64 bytes from 192.168.2.128: icmp_seq=14 ttl=64 time=0.499 ms
64 bytes from 192.168.2.128: icmp_seq=15 ttl=64 time=0.483 ms
64 bytes from 192.168.2.128: icmp_seq=16 ttl=64 time=0.498 ms
64 bytes from 192.168.2.128: icmp_seq=17 ttl=64 time=0.491 ms
64 bytes from 192.168.2.128: icmp_seq=18 ttl=64 time=1.01 ms
64 bytes from 192.168.2.128: icmp_seq=19 ttl=64 time=0.526 ms
64 bytes from 192.168.2.128: icmp_seq=20 ttl=64 time=0.592 ms
64 bytes from 192.168.2.128: icmp_seq=21 ttl=64 time=0.402 ms
64 bytes from 192.168.2.128: icmp_seq=22 ttl=64 time=0.589 ms
64 bytes from 192.168.2.128: icmp_seq=23 ttl=64 time=0.536 ms
64 bytes from 192.168.2.128: icmp_seq=24 ttl=64 time=0.505 ms
64 bytes from 192.168.2.128: icmp_seq=25 ttl=64 time=0.652 ms
64 bytes from 192.168.2.128: icmp_seq=26 ttl=64 time=0.414 ms
64 bytes from 192.168.2.128: icmp_seq=27 ttl=64 time=0.358 ms
64 bytes from 192.168.2.128: icmp_seq=28 ttl=64 time=0.508 ms
64 bytes from 192.168.2.128: icmp_seq=29 ttl=64 time=0.590 ms
64 bytes from 192.168.2.128: icmp_seq=30 ttl=64 time=0.661 ms
```

说明：如图为分别执行 sudo ./ping 192.168.2.128 和 ping 192.168.2.128 的结果对比，本机的 ip 是 192.168.2.129，左为自己编写的程序，右为系统自带的程序

（b）Ping 外网结果对比

```
user1@ubuntu:~/woozie/lab2$ sudo ./ping www.baidu.com
PING www.baidu.com (180.97.33.108) 56(84) bytes of data.
64 bytes from 180.97.33.108: icmp_seq=1 ttl=128 time=46.231 ms
64 bytes from 180.97.33.108: icmp_seq=2 ttl=128 time=46.681 ms
64 bytes from 180.97.33.108: icmp_seq=3 ttl=128 time=47.077 ms
64 bytes from 180.97.33.108: icmp_seq=4 ttl=128 time=47.397 ms
64 bytes from 180.97.33.108: icmp_seq=5 ttl=128 time=48.171 ms
64 bytes from 180.97.33.108: icmp_seq=6 ttl=128 time=46.732 ms
64 bytes from 180.97.33.108: icmp_seq=7 ttl=128 time=46.328 ms
64 bytes from 180.97.33.108: icmp_seq=8 ttl=128 time=46.856 ms
64 bytes from 180.97.33.108: icmp_seq=9 ttl=128 time=47.496 ms
64 bytes from 180.97.33.108: icmp_seq=10 ttl=128 time=46.792 ms
64 bytes from 180.97.33.108: icmp_seq=11 ttl=128 time=46.616 ms
```

```
user1@ubuntu:~$ ping www.baidu.com
PING www.a.shifen.com (180.97.33.108) 56(84) bytes of data.
64 bytes from 180.97.33.108 (180.97.33.108): icmp_seq=1 ttl=128 time=46.8 ms
64 bytes from 180.97.33.108 (180.97.33.108): icmp_seq=2 ttl=128 time=47.5 ms
64 bytes from 180.97.33.108 (180.97.33.108): icmp_seq=3 ttl=128 time=47.5 ms
64 bytes from 180.97.33.108 (180.97.33.108): icmp_seq=4 ttl=128 time=47.8 ms
64 bytes from 180.97.33.108 (180.97.33.108): icmp_seq=5 ttl=128 time=47.0 ms
64 bytes from 180.97.33.108 (180.97.33.108): icmp_seq=6 ttl=128 time=47.9 ms
64 bytes from 180.97.33.108 (180.97.33.108): icmp_seq=7 ttl=128 time=46.9 ms
64 bytes from 180.97.33.108 (180.97.33.108): icmp_seq=8 ttl=128 time=47.0 ms
64 bytes from 180.97.33.108 (180.97.33.108): icmp_seq=9 ttl=128 time=47.2 ms
64 bytes from 180.97.33.108 (180.97.33.108): icmp_seq=10 ttl=128 time=46.8 ms
64 bytes from 180.97.33.108 (180.97.33.108): icmp_seq=11 ttl=128 time=45.8 ms
```

说明：.左为自己的 ping，右为系统的 ping

（c）icmp 不可达的回应对比

```
PING 192.168.2.22 (192.168.2.22) 56(84) bytes of data.
From This Host icmp_seq=1 Destination Host Unreachable
From This Host icmp_seq=2 Destination Host Unreachable
From This Host icmp_seq=3 Destination Host Unreachable
From This Host icmp_seq=4 Destination Host Unreachable
From This Host icmp_seq=5 Destination Host Unreachable
From This Host icmp_seq=6 Destination Host Unreachable
From This Host icmp_seq=7 Destination Host Unreachable
From This Host icmp_seq=8 Destination Host Unreachable
From This Host icmp_seq=9 Destination Host Unreachable
From This Host icmp_seq=10 Destination Host Unreachable
```

```
PING 192.168.2.22 (192.168.2.22) 56(84) bytes of data.
From 192.168.2.129 icmp_seq=1 Destination Host Unreachable
From 192.168.2.129 icmp_seq=2 Destination Host Unreachable
From 192.168.2.129 icmp_seq=3 Destination Host Unreachable
From 192.168.2.129 icmp_seq=4 Destination Host Unreachable
From 192.168.2.129 icmp_seq=5 Destination Host Unreachable
From 192.168.2.129 icmp_seq=6 Destination Host Unreachable
From 192.168.2.129 icmp_seq=7 Destination Host Unreachable
From 192.168.2.129 icmp_seq=8 Destination Host Unreachable
From 192.168.2.129 icmp_seq=9 Destination Host Unreachable
From 192.168.2.129 icmp_seq=10 Destination Host Unreachable
```

说明：每当 0.3s 没有回应就会输出 unreachable 信息，左为自己 ping，右为系统的 ping

五． 相关参考资料

（1） 计算机网络实验教材 2.03（修订）

详细阅读实验要求，模仿框架代码的思路，抓包程序修改成层次结构更清晰的代码

（2） socket.ppt

仔细思考其中中对 ping 程序思路的讲解

（3） linux 网络编程.pdf

通过里面讲解 socket 编程章节，初步理解套接字编程内涵，并熟悉常见函数

（4） 计算机网络自顶而下方法

阅读课本关于各种协议的讲解，熟悉套接字编程的知识点，查找相关协议头结构

（5） 搜索引擎使用

查找不熟悉的协议头结构体，查找一些函数库以及相关套接字函数用法

比如如何将 recv 函数由 block 改成 unblock，了解到 select 函数等使用方法

六．对比样例程序

参考样例的部分是代码的思路：都是先创建套接字，然后不断地循环接受数据，一直到以太网帧头的解析都进行了参考。

七．个人创新及思考

(1) MyWireShark 创新

(a)内部协议的解析使用了结构体和指针偏移的思路，避免了直接的 bit 级别对数据进行解析

(b)代码层次化，根据网络协议的层次通过条件语句将解析分层，然后调用特定的解析接口函数进行解析，如 ip、arp、icmp、rarp、tcp、udp 相应的 analyse 函数

(2)ping 创新

（a）除了 ip 地址，还可以解析 host name

（b）接受函数使用了非阻塞的过程，这样发送给难以到达的目的地不会卡在 recv 函数中

八．程序的其他应用场景

（1） MyWireShark

比如检测查看有无恶意的网络数据行为

（2） ping

比如检测当前网络的连接情况