Lab2

171830635 俞星凯

<table>
<tr><td>实验目的</td><td>1. 学习 socket 相关知识
2. 通过学习 ip，icmp，arp 的报文格式来理解这些协议的功能和作用
3. 掌握初步 raw sockt 编程基础</td></tr>
<tr><td>数据结构说明</td><td>1. 在 Linux 中，IP 报头格式数据结构（<netinet/ip.h>）定义如下：
struct ip
{
#if __BYTE_ORDER == __LITTLE_ENDIAN
unsigned int ip_hl:4; /* header length */
unsigned int ip_v:4; /* version */
#endif
#if __BYTE_ORDER == __BIG_ENDIAN
unsigned int ip_v:4; /* version */
unsigned int ip_hl:4; /* header length */
#endif
u_int8_t ip_tos; /* type of service */
u_short ip_len; /* total length */
u_short ip_id; /* identification */
u_short ip_off; /* fragment offset field */
#define IP_RF 0x8000 /* reserved fragment flag */
#define IP_DF 0x4000 /* dont fragment flag */
#define IP_MF 0x2000 /* more fragments flag */
#define IP_OFFMASK 0x1fff /* mask for fragmenting bits */
u_int8_t ip_ttl; /* time to live */
u_int8_t ip_p; /* protocol */
u_short ip_sum; /* checksum */
struct in_addr ip_src, ip_dst; /* source and dest address */
};
 

2. 在 Linux 中，ICMP_ECHO 和 ICMP_ECHOREPLY 数据结构（<netinet/ip_icmp.h>）定义如下：
struct icmp
{
u_int8_t icmp_type; /* type of message, see below */
u_int8_t icmp_code; /* type sub code */</td></tr>
</table>

u_int16_t icmp_cksum; /* ones complement checksum of struct */
struct ih_idseq /* echo datagram */
{
u_int16_t icd_id;
u_int16_t icd_seq;
} ih_idseq;
}

类型 TYPE(8或0)	编码 CODE (没有使用)	校验和 CHECKSUM
标志符 Identifier		顺序号 Sequence NO

程序设计的思路以及运行流程

1. raw_socket

创建一个 raw socket，不断收包，显示包的 MAC 和 IP 地址，并根据包头类型域的值来显示包的类型。

2. raw_socket_ping 注：括号内为本次程序的函数名

(1) 根据参数是主机名还是 IP 地址进行不同设置

(2) 创建一个 raw socket

(3) 发送一个包(send_packet)，需要先对包头进行封装(pack)，其中检验和字段采用检验和算法(cal_cksum)。

(4) 接收一个包(recv_packet)，然后再对包头进行解封(unpack)，计算 rtt(cal_internal)，输出相关信息。

(5) 循环(3)(4)

(6) 进行数据统计。

运行结果截图

1. raw_socket

```
MAC address: 00:0c:29:82:6c:74==>00:0c:29:25:7f:51
IP:192.168.0.2==> 192.168.1.2
Protocol:icmp
MAC address: 00:0c:29:25:7f:51==>00:0c:29:82:6c:74
IP:192.168.1.2==> 192.168.0.2
Protocol:icmp
MAC address: 00:0c:29:25:7f:51==>00:0c:29:82:6c:74
IP:192.168.0.1==> 192.168.0.2
Protocol:arp
MAC address: 00:0c:29:82:6c:74==>00:0c:29:25:7f:51
IP:192.168.0.2==> 192.168.0.1
Protocol:arp
MAC address: 00:00:00:00:00:00==>00:00:00:00:00:00
IP:127.0.0.1==> 127.0.0.1
Protocol:udp
MAC address: 00:00:00:00:00:00==>00:00:00:00:00:00
IP:127.0.0.1==> 127.0.0.1
Protocol:udp
MAC address: 00:00:00:00:00:00==>00:00:00:00:00:00
IP:127.0.0.1==> 127.0.0.1
Protocol:icmp
MAC address: 00:00:00:00:00:00==>00:00:00:00:00:00
IP:127.0.0.1==> 127.0.0.1
Protocol:icmp
```

	2. raw_socket_ping root@ubuntu:/home/user# ./raw_socket_ping 192.168.1.2 Ping 192.168.1.2(192.168.1.2): 56 bytes of data in ICMP packets. 64 bytes from 192.168.1.2: icmp_seq = 1 ttl = 63 rtt = 2.515ms 64 bytes from 192.168.1.2: icmp_seq = 2 ttl = 63 rtt = 1.894ms 64 bytes from 192.168.1.2: icmp_seq = 3 ttl = 63 rtt = 1.664ms 64 bytes from 192.168.1.2: icmp_seq = 4 ttl = 63 rtt = 1.583ms 64 bytes from 192.168.1.2: icmp_seq = 5 ttl = 63 rtt = 0.746ms 64 bytes from 192.168.1.2: icmp_seq = 6 ttl = 63 rtt = 2.925ms 64 bytes from 192.168.1.2: icmp_seq = 7 ttl = 63 rtt = 1.278ms 64 bytes from 192.168.1.2: icmp_seq = 8 ttl = 63 rtt = 1.056ms 64 bytes from 192.168.1.2: icmp_seq = 9 ttl = 63 rtt = 0.961ms 64 bytes from 192.168.1.2: icmp_seq = 10 ttl = 63 rtt = 2.854ms 64 bytes from 192.168.1.2: icmp_seq = 11 ttl = 63 rtt = 0.932ms 64 bytes from 192.168.1.2: icmp_seq = 12 ttl = 63 rtt = 1.784ms 64 bytes from 192.168.1.2: icmp_seq = 13 ttl = 63 rtt = 0.950ms 64 bytes from 192.168.1.2: icmp_seq = 14 ttl = 63 rtt = 4.721ms 64 bytes from 192.168.1.2: icmp_seq = 15 ttl = 63 rtt = 0.926ms 64 bytes from 192.168.1.2: icmp_seq = 16 ttl = 63 rtt = 1.308ms 64 bytes from 192.168.1.2: icmp_seq = 17 ttl = 63 rtt = 0.640ms 64 bytes from 192.168.1.2: icmp_seq = 18 ttl = 63 rtt = 1.702ms 64 bytes from 192.168.1.2: icmp_seq = 19 ttl = 63 rtt = 0.924ms 64 bytes from 192.168.1.2: icmp_seq = 20 ttl = 63 rtt = 0.790ms 64 bytes from 192.168.1.2: icmp_seq = 21 ttl = 63 rtt = 1.181ms 64 bytes from 192.168.1.2: icmp_seq = 22 ttl = 63 rtt = 1.179ms 64 bytes from 192.168.1.2: icmp_seq = 23 ttl = 63 rtt = 1.201ms 64 bytes from 192.168.1.2: icmp_seq = 24 ttl = 63 rtt = 1.013ms 64 bytes from 192.168.1.2: icmp_seq = 25 ttl = 63 rtt = 1.220ms 64 bytes from 192.168.1.2: icmp_seq = 26 ttl = 63 rtt = 1.361ms 64 bytes from 192.168.1.2: icmp_seq = 27 ttl = 63 rtt = 1.143ms 64 bytes from 192.168.1.2: icmp_seq = 28 ttl = 63 rtt = 0.955ms 64 bytes from 192.168.1.2: icmp_seq = 29 ttl = 63 rtt = 1.757ms 64 bytes from 192.168.1.2: icmp_seq = 30 ttl = 63 rtt = 1.185ms 64 bytes from 192.168.1.2: icmp_seq = 31 ttl = 63 rtt = 1.167ms 64 bytes from 192.168.1.2: icmp_seq = 32 ttl = 63 rtt = 0.971ms --- 192.168.1.2 ping statistics --- 32 packets transmitted, 32 received, %0 packet loss
相关参考资料	计算机网络实验教材 2.02(修订) https://blog.csdn.net/xtank_nie/article/details/39215225
对比样例程序	参考了教材上的 raw_socket 样例程序，并对其略做修改使之支持 ARP 的识别。
代码个人创新以及思考	1. raw_socket 根据 MAC 包的第 13 字节判断 IP 数据包还是 ARP 请求/应答。 2. raw_socket_ping (1) 修改了计算 rtt 的算法，使之能够正确显示到小数点后三位。 (2) 由于只需实现简易的 ping 程序，故简化了代码中的大量冗余语句。 (3) 采用系统 ping 程序的格式进行输出。
该程序的应用场景创新（非必须）	在多跳网络中每一跳都显示一次，即可判断断路位置。