实验 8 数据链路层数据操作及 ARP

目录

实验目的

熟悉数据链路层数据的获取方法，能够从数据链路层获取网络层、传输层和应用层的数据，掌握 ARP 协议。

实验要求

1. 了解数据链路层数据的获取方法，包括设置套接口以捕获链路帧的编程方法、从套接口读取链路帧的编程方法、定位 IP 包头的编程方法、定位 TCP 报头的编程方法、定位 UDP 报头的编程方法和定位应用层报文数据的编程方法

2. 使用 SOCK_PACKET 编写 ARP 请求程序

   - 了解 ARP 协议
   - 使用发送 ARP 请求数据
   - 使用 ARP 命令查看 ARP 表并验证

实验环境

实验环境采用 **vscode remote + wsl**，操作系统的版本为 **Ubuntu 20.04.2 LTS**，如下图所示

```
❯ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.2 LTS
Release:        20.04
Codename:       focal
```

操作系统版本

内核版本如下图所示

```
❯ uname -a
Linux qiufeng 5.10.16.3-microsoft-standard-WSL2
```

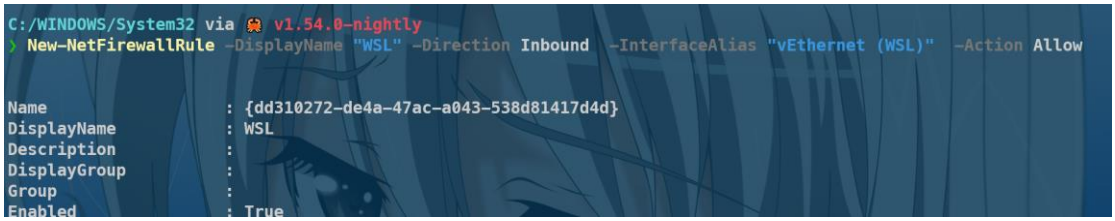
内核版本

gcc 版本为 9.3.0

```
❯ gcc --version
gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

gcc 版本

需要注意的是，由于 windows 防火墙阻止了 wsl2 到主机的通信，因此我们必须使用在 powershell 中执行如下命令允许 wsl 请求通过防火墙



设置防火墙

实验步骤

1. 了解链路层数据获取和处理的方法

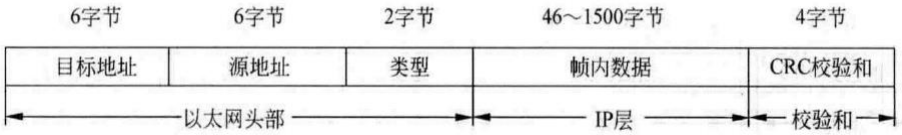
在 linux 中，要实现数据链路层的访问通常是通过编写内核驱动程序来完成的，在应用层使用 ***SOCK_PACKET***，即原始套接字，也能够完成一部分的功能。

要实现**捕获链路帧**，需要创建原始套接字。在如下代码中，*0x0003* 表示截取的数据帧的类型为不确定，即处理所有的包

```c
//* 创建原始套接字
int fd;
fd = socket(AF_INET, SOCK_PACKET, htons(0x0003));
```

创建原始套接字

要实现**链路帧的读取**，需要了解以太网帧的基本结构



以太网帧的基本结构

其中以太网头部的结构体 *ethhdr* 定义如下图所示

```c
struct ethhdr {
    unsigned char   h_dest[ETH_ALEN];   /* destination eth addr */
    unsigned char   h_source[ETH_ALEN]; /* source ether addr    */
    __be16      h_proto;        /* packet type ID field */
} __attribute__((packed));
```

结构体定义

我们可以通过 *read* 函数从原始套接字中读取链路帧。

要**定位 IP 包头**，如果以太网帧头部信息中指示了协议类型为 ***0x0800***，说明报文内容即为 IP 协议，IP 包头位于以太网帧数据部分的开始位置。

在 linux 中，IP 数据包报头的结构体 ***iphdr*** 定义如下

```
struct iphdr
  {
#if __BYTE_ORDER == __LITTLE_ENDIAN
    unsigned int ihl:4;
    unsigned int version:4;
#elif __BYTE_ORDER == __BIG_ENDIAN
    unsigned int version:4;
    unsigned int ihl:4;
#else
# error "Please fix <bits/endian.h>"
#endif
    uint8_t tos;
    uint16_t tot_len;
    uint16_t id;
    uint16_t frag_off;
    uint8_t ttl;
    uint8_t protocol;
    uint16_t check;
    uint32_t saddr;
    uint32_t daddr;
    /*The options start here. */
  };
```

ip 包头结构体

用类型为 ***iphdr*** 的结构体指针指向链路帧载荷数据的起始位置就可以得到 IP 报头信息

要**定位 TCP 包头**，需要解析 IP 包头中 IP 头部的长度，其长度× 4即为 TCP 头部的起始位置，对应代码为

```
//* ip 包头
struct iphdr *p_iphdr;
if (p_iphdr->protocol == 6) {
    //* 定位 tcp 包头
    struct tcphdr *p_tcphdr = (struct tcphdr *)(p_iphdr + p_iphdr->ihl * 4);
}
```

定位 TCP 包头

要**定位 UDP 包头**，方法与定位 TCP 包头类似，只不过协议类型由 6 变为了

17。对应代码如下

```
//* ip 包头
struct iphdr *p_iphdr;
if (p_iphdr->protocol == 17) {
    //* 定位 udp 包头
    struct udphdr *p_udphdr = (struct udphdr *)(p_iphdr + p_iphdr->ihl * 4);
}
```

定位 UDP 包头

要**定位应用层数据**，即定位 TCP 报文或者 UDP 报文的数据部分，对应的代码如下所示

```
//* ip 包头
struct iphdr *p_iphdr;
//* TCP 数据包
if (p_iphdr->protocol == 6) {
    struct tcphdr *p_tcphdr = (struct tcphdr *)(p_iphdr + p_iphdr->ihl * 4);
    //! 定位 tcp 应用部分数据地址
    char *app_data = (char *)p_tcphdr + 20;
}
//* UDP 数据包
else if (p_iphdr->protocol == 17) {
    struct udphdr *p_udphdr = (struct udphdr *)(p_iphdr + p_iphdr->ihl * 4);
    //! 定位 udp 应用部分数据地址
    char *app_data = (char *)(p_udphdr + p_udphdr->len);
}
```

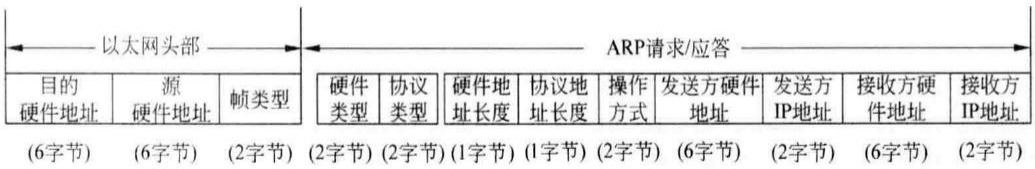
定位应用层数据

2. 使用 SOCK_PACKET 编写 ARP 请求程序

2.1 了解 ARP 协议

ARP 协议的全称是 ***Address Resolution Protocol***(地址解析协议)，它是一个通过用于实现从 IP 地址到 MAC 地址的映射，即询问目标 IP 对应的 MAC 地址的一种协议。

ARP 协议的以太网帧格式如下图所示



ARP 协议以太网帧格式

其由结构体 ***arphdr*** 定义

```
struct arphdr
  {
    unsigned short int ar_hrd;    /* Format of hardware address.  */
    unsigned short int ar_pro;    /* Format of protocol address.  */
    unsigned char ar_hln;   /* Length of hardware address.  */
    unsigned char ar_pln;   /* Length of protocol address.  */
    unsigned short int ar_op;   /* ARP opcode (command).  */
```

arphdr 结构体

每个字段的含义如下

表 11.6 ARP 在以太网上请求包的值和含义

成员	成 员 含 义	值	值 含 义
ar_hrd	硬件类型	1	硬件地址为以太网接口
ar_pro	协议类型	0x0800	高层协议为 IP 协议
ar_hln	硬件地址长度	6	6 字节，即 MAC 地址 48 位
ar_pln	协议地址长度	4	IP 协议地址长度为 32 位
ar_op	ARP 操作码	1	ARP 请求

ARP 结构体成员的含义

2.2 使用发送 ARP 请求数据

定义 ARP 报文结构体 ***arp_packet***，其各成员的意义已经在注释中标出

```c
struct arp_packet {
    //* arp 头部
    struct arphdr ar_head;
    //* 源 mac 地址
    unsigned char ar_sha[ETH_ALEN];
    //* 源 ip 地址
    struct in_addr ar_sip;
    //* 目的 mac 地址
    unsigned char ar_tha[ETH_ALEN];
    //* 目的 ip 地址
    struct in_addr ar_tip;
}__attribute__((packed));
```

ARP 报文结构体

首先使用 ***socket*** 函数，并指定 ***SOCK_RAW*** 参数创建原始套接字

```c
// create sock_packet socket
int fd = socket(PF_PACKET, SOCK_RAW, htonl(ETH_P_ARP));
if (fd == -1) {
    perror("create socket error\n");
    exit(-1);
}
```

创建原始套接字

使用函数 ***ioctl*** 并指定参数 ***SIOCGIFADDR*** 获取本机 IP 地址

```c
if (ioctl(fd, SIOCGIFADDR, ifrPtr) == -1) {
    perror("get ip address error\n");
    exit(-1);
}
netaddr = ((struct sockaddr_in *) &(ifrPtr->ifr_addr))->sin_addr;
```

获取本机ip

获取本机 IP 地址

使用函数 ***ioctl*** 并指定参数 ***SIOGIFNETMASK*** 获取本机 IP 地址对应的子网掩码

```c
if (ioctl(fd, SIOCGIFNETMASK, ifrPtr) == -1) {
    perror("get netmask error\n");
    exit(-1);
}
netmask = ((struct sockaddr_in *) &(ifrPtr->ifr_netmask))->sin_addr;
```

获取子网掩码

使用函数 ***ioctl*** 并指定参数 ***SIOCGIFHWADDR*** 获取本机 mac 地址

```c
if (ioctl(fd, SIOCGIFHWADDR, ifrPtr) == -1) {
    perror("get net interface hwaddr error\n");
    exit(-1);
}
memcpy(hwaddr.sll_addr, ifrPtr->ifr_hwaddr.sa_data, ETH_ALEN);
```

获取 mac 地址

接着利用上述获得的信息填充 ARP 包头

```c
//* 填充 arp 包头
char ef[ETH_FRAME_LEN];
struct ethhdr *p_eth = (struct ethhdr *) ef;
memset(p_eth->h_dest, 0xff, ETH_ALEN);
memcpy(p_eth->h_source, hwaddr.sll_addr, ETH_ALEN);
p_eth->h_proto = htons(ETH_P_ARP);

struct arp_packet *p_arp = (struct arp_packet *) (ef + ETH_HLEN);
p_arp->ar_head.ar_hrd = htons(ARPHRD_ETHER);
p_arp->ar_head.ar_pro = htons(ETH_P_IP);
p_arp->ar_head.ar_hln = ETH_ALEN;
p_arp->ar_head.ar_pln = 4;
p_arp->ar_head.ar_op = htons(ARPOP_REQUEST);
memcpy(p_arp->ar_sha, hwaddr.sll_addr, ETH_ALEN);
p_arp->ar_sip = netaddr;
memset(p_arp->ar_tha, 0, ETH_ALEN);
p_arp->ar_tip = pingaddr;
```

填充 ARP 包头

最后发送 ARP 请求报文并对接收到的报文进行解析，打印目的 MAC 地址

```c
write(fd, ef, 60);
while (1) {
    read(fd, ef, sizeof(ef));
    struct arp_packet *recv_arp = (struct arp_packet *) (ef + ETH_HLEN);
    if (recv_arp->ar_tip.s_addr == netaddr.s_addr) {
        printMac("mac: ", recv_arp->ar_sha);
        break;
    }
}
return 0;
```

发送和解析 ARP 报文

2.3 使用 ARP 命令查看 ARP 表并验证

见实验测试部分

实验测试

执行 ***ifconfig*** 命令，获取本机 IP 地址为 ***172.31.158.147***，子网掩码为 ***255.255.255.240***，mac 地址为 ***00:15:5d:01:bb:56***

```
❯ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.31.158.147  netmask 255.255.240.0  broadcast 172.31.159.255
        inet6 fe80::215:5dff:fe01:bb56  prefixlen 64  scopeid 0x20<link>
        ether 00:15:5d:01:bb:56  txqueuelen 1000  (Ethernet)
        RX packets 70196  bytes 49358075 (49.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 39684  bytes 14036140 (14.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

ifconfig

执行 ***netstat -rn*** 命令，获取网关地址为 ***172.31.144.1***

```
❯ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         172.31.144.1    0.0.0.0         UG        0 0          0 eth0
172.31.144.0    0.0.0.0         255.255.240.0   U         0 0          0 eth0
```

netstat -rn

由于 wsl 不存在其他主机，因此我们直接向网关发送 ARP 请求，希望获取网关的 MAC 地址

直接使用 ***arping*** 命令，可以得到网关的 mac 地址为 ***00:15:5d:58:00:45***

```
❯ sudo arping 172.31.144.1
ARPING 172.31.144.1
42 bytes from 00:15:5d:58:00:45 (172.31.144.1): index=0 time=198.600 usec
42 bytes from 00:15:5d:58:00:45 (172.31.144.1): index=1 time=284.900 usec
```

arping

使用 gcc 编译程序

```
❯ gcc arp.c -o arp
```

gcc 编译

执行命令 ***sudo ./arp 172.31.144.1***，获得相同的 mac 地址 ***00:15:5d:58:00:45***

```
❯ sudo ./arp 172.31.144.1
source ip: 172.31.158.147
source netmask: 255.255.240.0
source mac: 00-15-5d-01-bb-56
target ip: 172.31.144.1
target mac: 00-15-5d-58-00-45
```

获取网关 mac 地址

查看 ARP 表，可以发现存在映射

```
❯ arp -v
Address                  HWtype  HWaddress           Flags Mask            Iface
qiufeng.mshome.net       ether   00:15:5d:58:00:45   C                     eth0
Entries: 1      Skipped: 0      Found: 1
```

ARP 映射

实验体会

本次实验使用了原始套接字实现了 ARP 请求程序的编写。相比于流式套接字，原始套接字更加底层，具有的功能更多，可以直接访问链路数据帧，并通过对包头的逐层解析，进而访问 IP 层、TCP/UDP 层和应用层的数据。

通过这次实验，让我对原始套接字的使用有了更进一步的了解，明白了如何寻找更高层次的报文信息，同时加深了对 ARP 协议的理解。