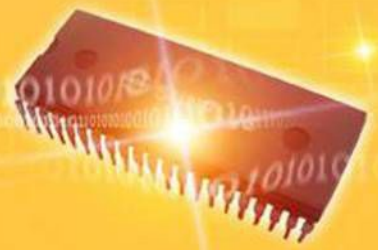
嵌入式系统工程师



原始套接字

Tel: 400-705-9680 , Email: edu@sunplusapp.com , BBS: bbs.sunplusedu.com

大纲

- TCP、UDP开发回顾
- 原始套接字概述、创建
- 数据包详解
- 编程实例—分析MAC数据包
- 练习—网络数据分析器
- sendto发送数据
- 练习—MAC地址扫描器（ARP）
- 练习—飞鸽欺骗（UDP）
- 练习—三次握手连接器（TCP）

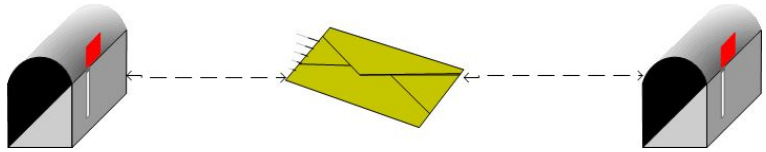
大纲

- TCP、UDP开发回顾
- 原始套接字概述、创建
- 数据包详解
- 编程实例—分析MAC数据包
- 练习—网络数据分析器
- sendto发送数据
- 练习—MAC地址扫描器（ARP）
- 练习—飞鸽欺骗（UDP）
- 练习—三次握手连接器（TCP）

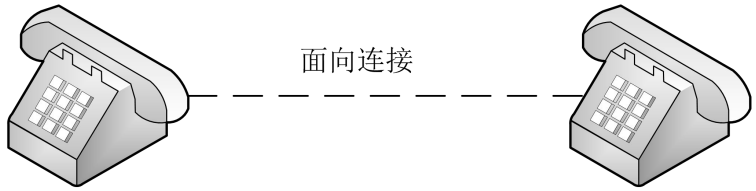
Tel: 400-705-9680 , Email: edu@sunplusapp.com , BBS: bbs.sunplusedu.com

TCP、UDP套接字

- 数据报式套接字（SOCK_DGRAM）
  - 无连接的socket,针对无连接的 UDP 服务
  - 可通过邮件模型来进行对比



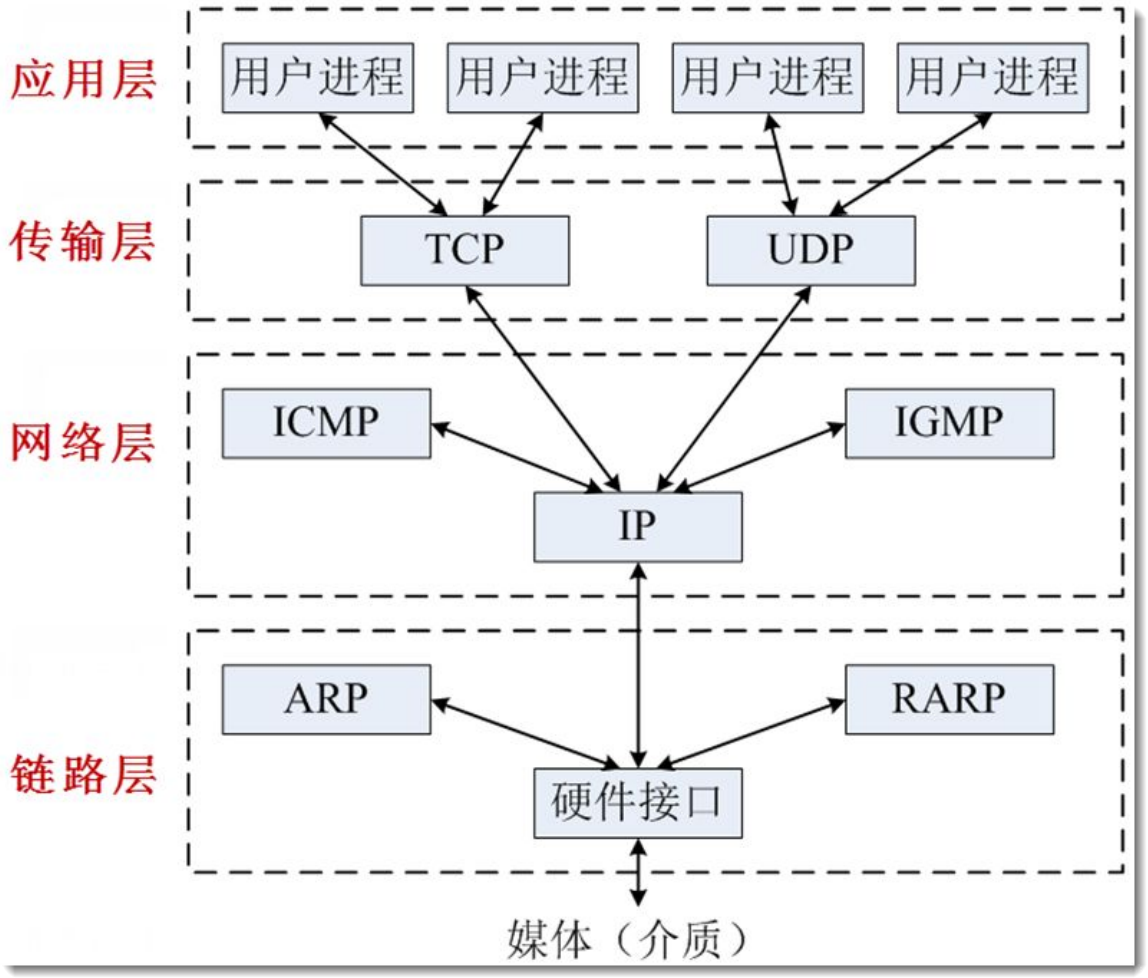
- 流式套接字（SOCK_STREAM）
  - 面向连接的socket,针对面向连接的 TCP 服务
  - 可通过电话模型来进行对比



- 这两类套接字似乎涵盖了TCP/IP 应用的全部

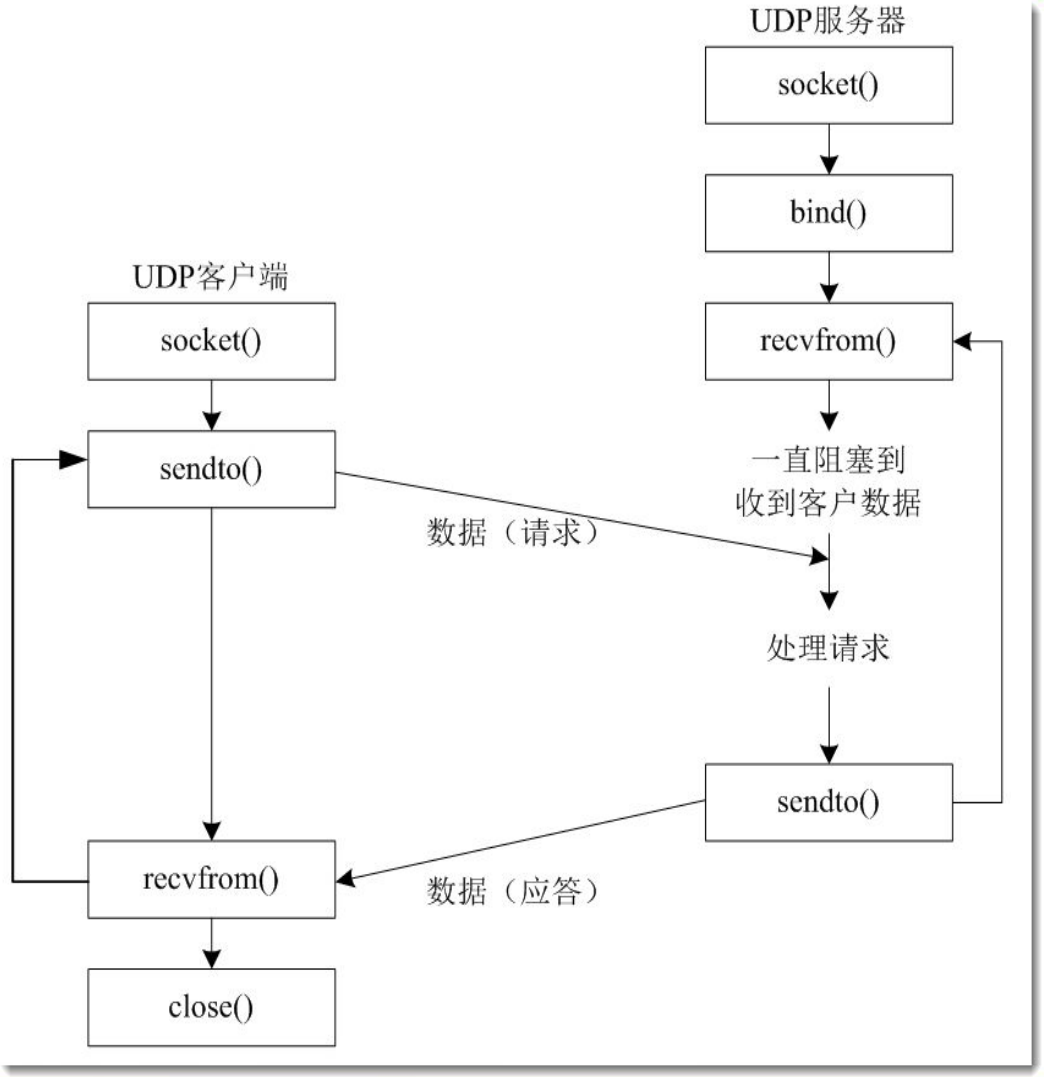
TCP、UDP开发回顾

- TCP与UDP各自有独立的port互不影响
- 一个进程可同时拥有多个port
- 不必关心tcp/ip协议实现的过程



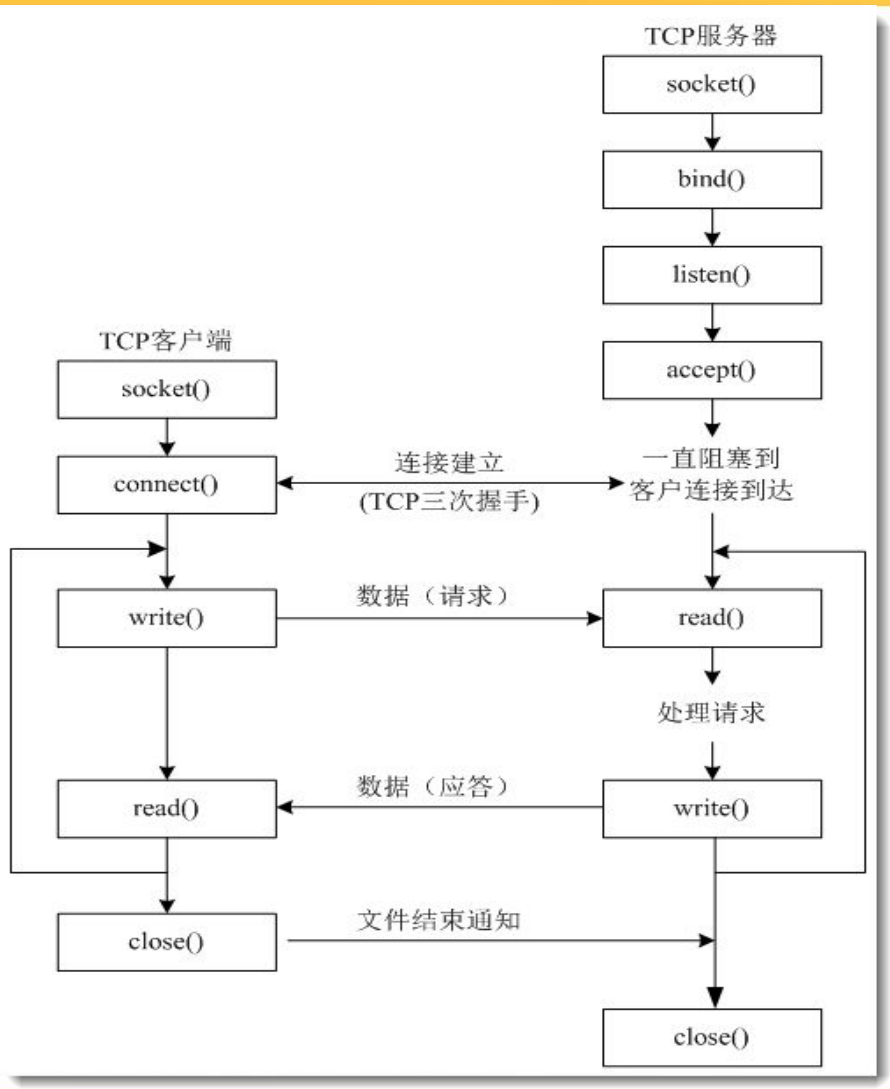
UDP编程开发回顾

- client
  - 创建socket接口
  - 定义sockaddr_in变量，其中ip、port为目的主机的信息
  - 可发送0长度的数据包
- server
  - bind本地主机的ip、port等信息
  - 接收到的数据包中包含来源主机的ip、port信息



TCP编程开发回顾

- client
  - connect来建立连接
  - write、read收发数据
  - 不可发送0长度的数据
- server
  - bind本地主机的ip、port等信息
  - listen把主动套接字变为被动
  - accept会有新的返回值
  - 多进程、线程完成并发



想一想

- 能否截获网络中的数据？
- 怎样发送一个自定义的IP包？
- 怎样伪装本地的IP、MAC？
- 网络攻击是怎么回事？
- 路由器、交换机怎样实现？

研究方向

- 方向：
  1. TCP/IP协议栈
  2. 原始套接字
  3. 网络开发工具包libpcap/libnet
- 书籍：
  1. 《TCP/IP详解 卷一》★
  2. 《UNIX网络编程 卷一》第三版★

大纲

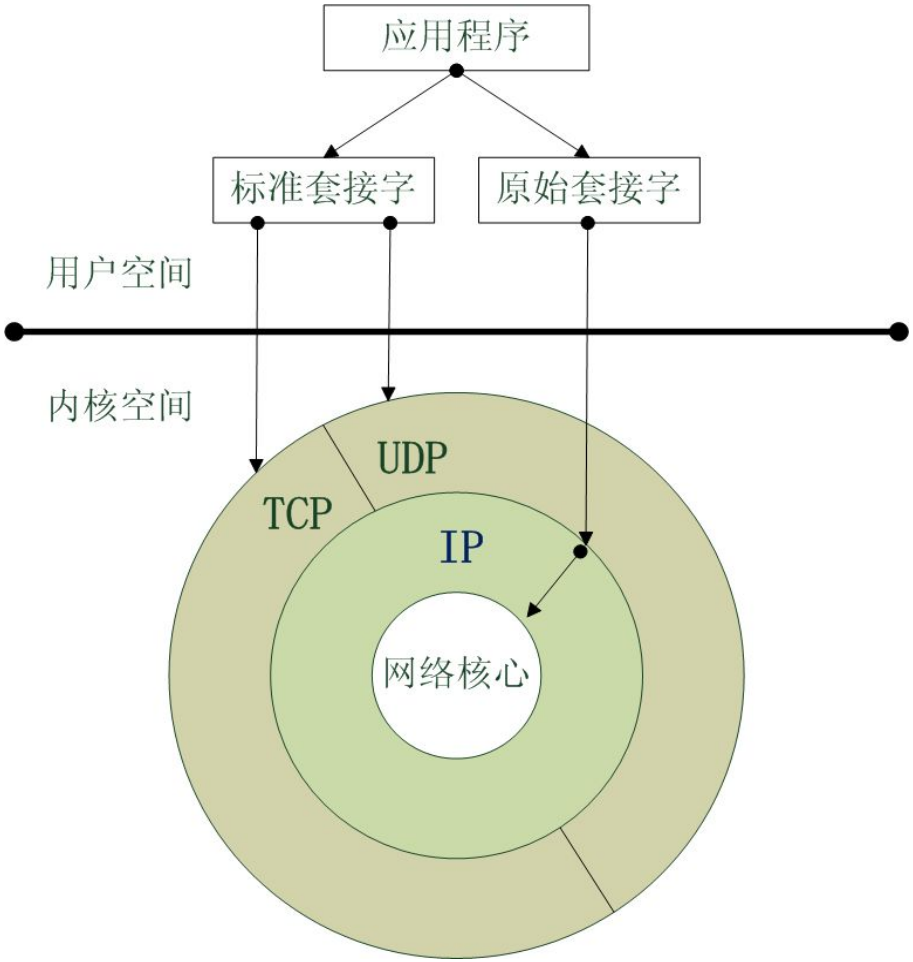
- TCP、UDP开发回顾
- 原始套接字概述、创建
- 数据包详解
- 编程实例—分析MAC数据包
- 练习—网络数据分析器
- sendto发送数据
- 练习—MAC地址扫描器（ARP）
- 练习—飞鸽欺骗（UDP）
- 练习—三次握手连接器（TCP）

原始套接字概述

- 原始套接字（SOCK_RAW）
  - 一种不同于SOCK_STREAM、SOCK_DGRAM的套接字，它实现于系统核心
  - 可以接收本机网卡上所有的数据帧（数据包），对于监听网络流量和分析网络数据很有作用
  - 开发人员可发送自己组装的数据包到网络上
  - 广泛应用于高级网络编程
  - 网络专家、黑客通常会用此来编写奇特的网络程序

套接字功能对比

- 流式套接字只能收发TCP协议的数据
- 数据报套接字只能收发UDP协议的数据
- 原始套接字可以收发内核没有处理的数据包
  因此，要访问其他协议发送的数据需要使用原始套接字(SOCK_RAW)



创建原始套接字

- int socket(PF_PACKET, SOCK_RAW, protocol)
- 功能：创建链路层的原始套接字
- protocol：指定可以接收或发送的数据包类型
  - ETH_P_IP:IPV4数据包
  - ETH_P_ARP:ARP数据包
  - ETH_P_ALL:任何协议类型的数据包
- 返回值：
  - 成功(>0):链路层套接字
  - 失败(<0):出错

创建原始套接字

- 创建链路层的原始套接字

```
sock_raw_fd = socket(PF_PACKET,SOCK_RAW,
    htons(ETH_P_ALL));
```

  - 已过时，不再使用

```
sock_raw_fd = socket(AF_INET, SOCK_PACKET,
htons(ETH_P_ALL));
```

- 头文件
  - #include <sys/socket.h>
  - #include <netinet/ether.h>

大纲

- TCP、UDP开发回顾
- 原始套接字概述、创建
- 数据包详解
- 编程实例—分析MAC数据包
- 练习—网络数据分析器
- sendto发送数据
- 练习—MAC地址扫描器（ARP）
- 练习—飞鸽欺骗（UDP）
- 练习—三次握手连接器（TCP）

数据包详解

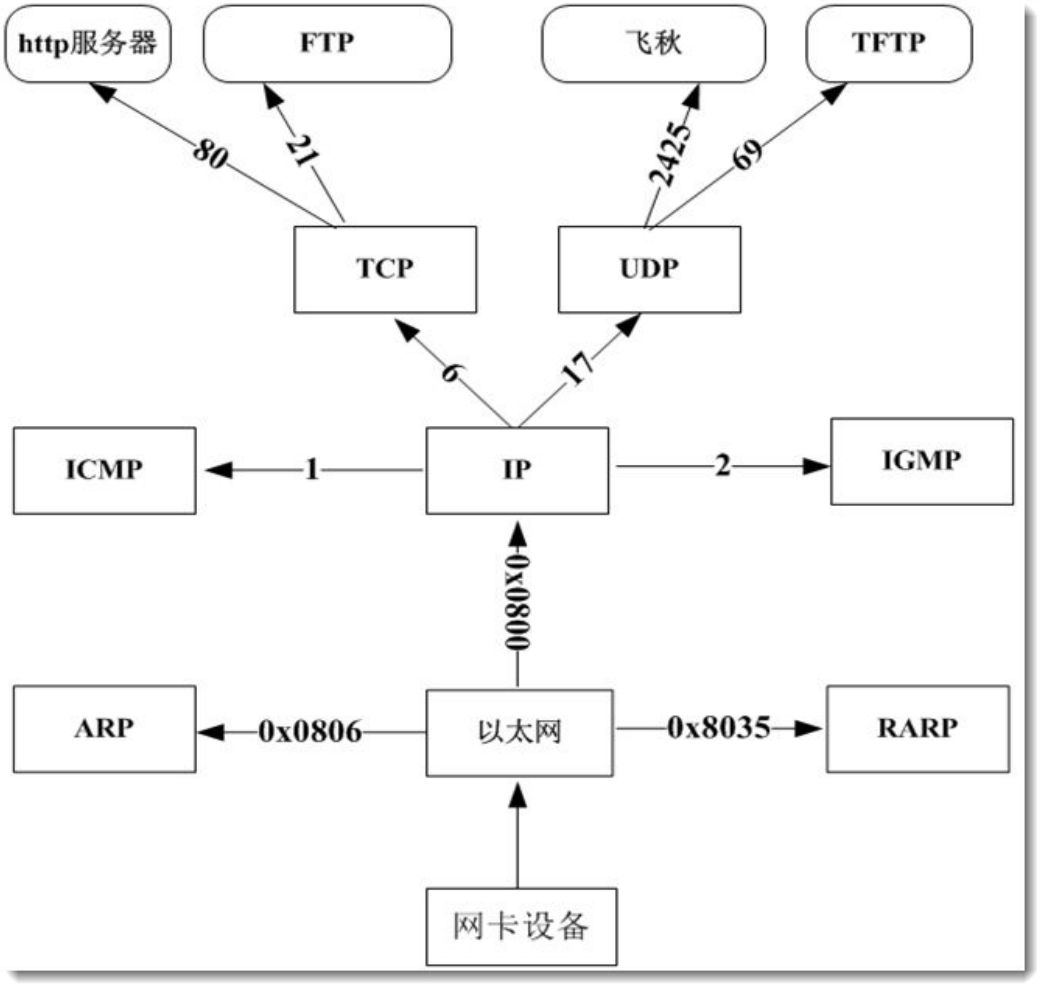
- 使用原始套接字进行编程开发时，首先要对不同协议的数据包进行学习，需要手动对IP、TCP、UDP、ICMP等包头进行组装或者拆解
- ubuntu12.04中描述网络协议结构的文件如下：

```
root@edu-T: /usr/include/netinet
root@edu-T:/usr/include/netinet# ls
ether.h   if_ether.h  if_tr.h  in.h        ip6.h   ip_icmp.h  tcp.h
icmp6.h   if_fddi.h   igmp.h   in_systm.h  ip.h    tags       udp.h
root@edu-T:/usr/include/netinet#
```

```
root@edu-T: /usr/include/net
root@edu-T:/usr/include/net# ls
ethernet.h  if.h         if_ppp.h     if_slip.h   ppp_defs.h  tags
if_arp.h    if_packet.h  if_shaper.h  ppp-comp.h  route.h
root@edu-T:/usr/include/net#
```

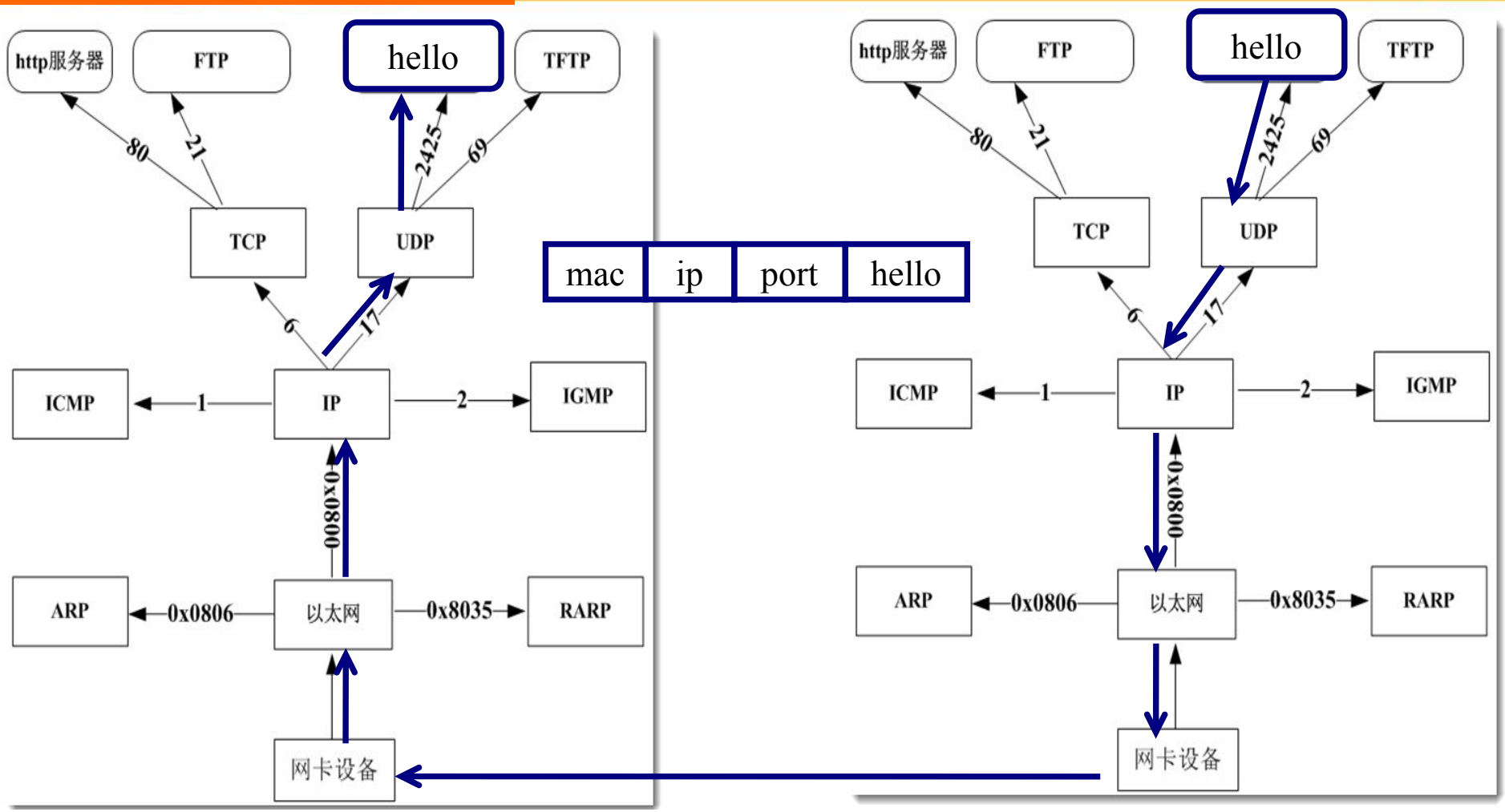
数据包详解

- 在TCP/IP协议栈中的每一层为了能够正确解析出上层的数据包，从而使用一些“协议类型”来标记，详细如右图



网络协议分析图

组装/拆解udp数据包流程



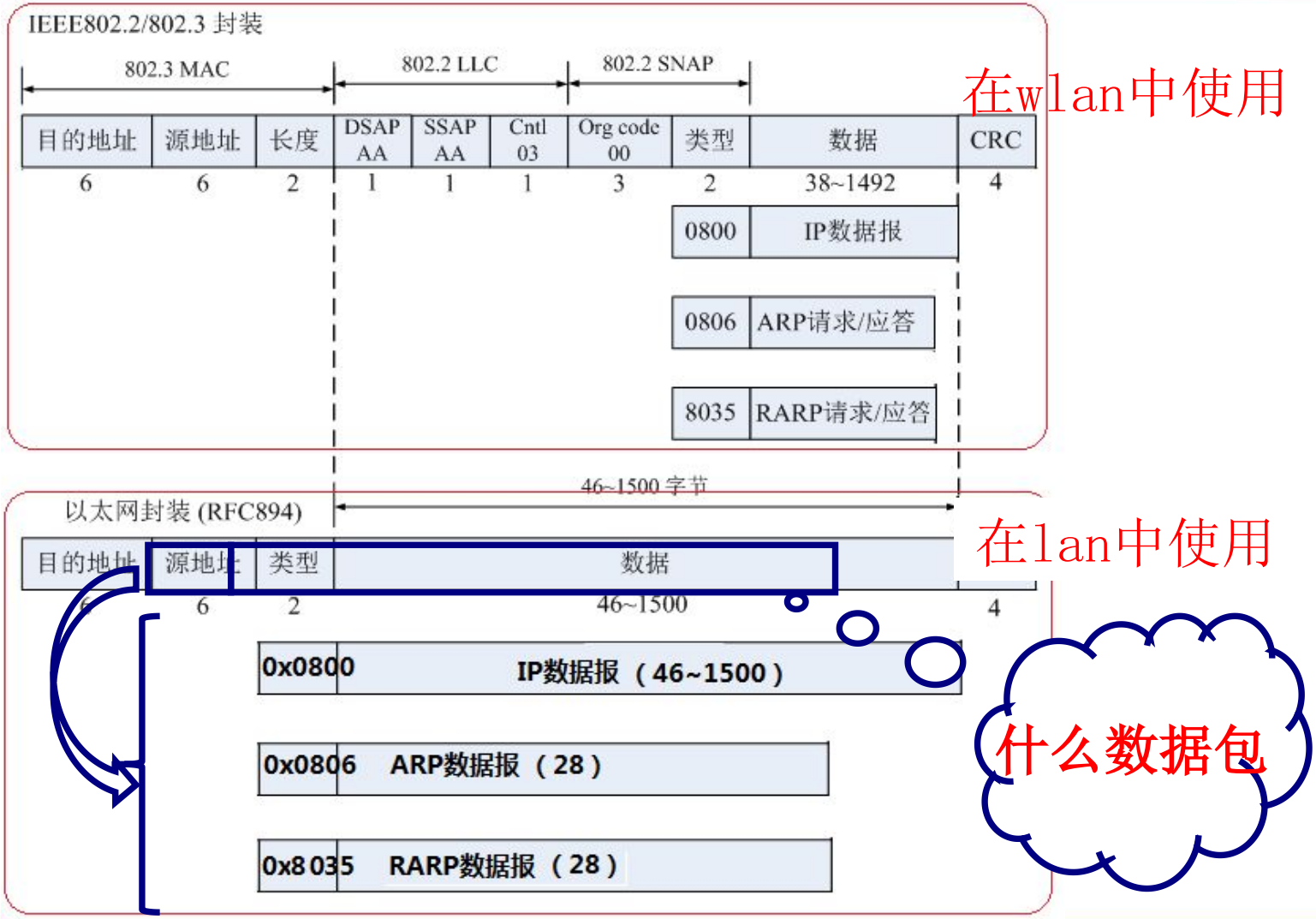
想一想：tcp呢？

UDP封包格式

port来标记给哪个进程

IP封包格式

Ethernet封包格式



TCP封包格式

代表：4B的个数

port来标记给哪个进程

给哪个进程

ICMP封包格式

类型	代码	描述
0	0	回显应答
8	0	回显请求

ICMP回显请求和回显应答格式

注：不同的类型值以及代码值，代表不同的功能

大纲

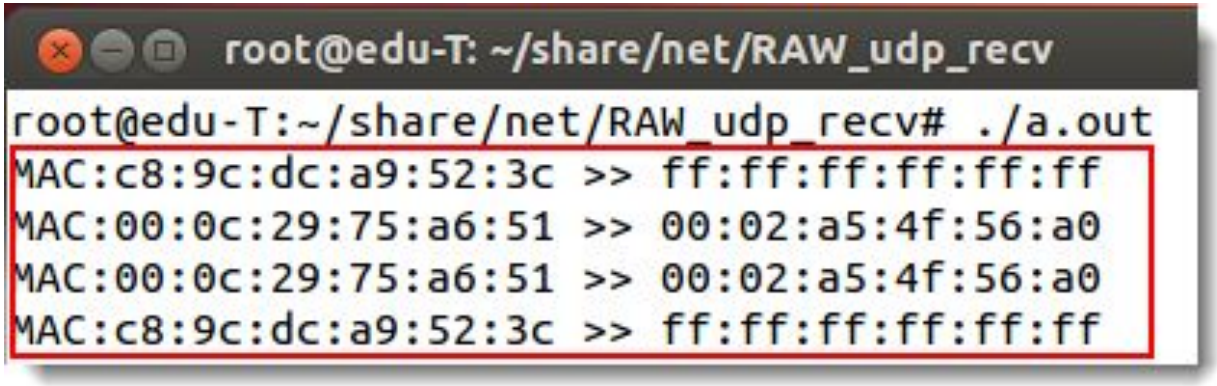
- TCP、UDP开发回顾
- 原始套接字概述、创建
- 数据包详解
- 编程实例—分析MAC数据包
- 练习—网络数据分析器
- sendto发送数据
- 练习—MAC地址扫描器（ARP）
- 练习—飞鸽欺骗（UDP）
- 练习—三次握手连接器（TCP）

分析MAC数据包

➢ 链路层数据格式

```
⊞ Destination: Elitegro_b7:0f:19 (c8:9c:dc:b7:0f:19)
⊞ Source: b8:88:e3:e1:10:e6 (b8:88:e3:e1:10:e6)
  Type: IP (0x0800)
```

➢ 示例效果



分析MAC数据包

- 参考代码analysis_mac.c

```c
#include <stdio.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <netinet/ether.h>
int main(int argc,char *argv[])
{
    unsigned char buf[1024] = "";
    int sock_raw_fd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
    while(1)
    {
        unsigned char src_mac[18] = "";
        unsigned char dst_mac[18] = "";
        //获取链路层的数据帧
        recvfrom(sock_raw_fd, buf, sizeof(buf),0,NULL,NULL);
        //从buf里提取目的mac、源mac
        sprintf(dst_mac,"%02x:%02x:%02x:%02x:%02x:%02x", \
        buf[0], buf[1], buf[2], buf[3], buf[4], buf[5]);
        sprintf(src_mac,"%02x:%02x:%02x:%02x:%02x:%02x", \
        buf[6], buf[7], buf[8], buf[9], buf[10], buf[11]);
        //打印源MAC、目的MAC
        printf("MAC:%s >> %s\n", src_mac, dst_mac);
    }
    return 0;
}
```

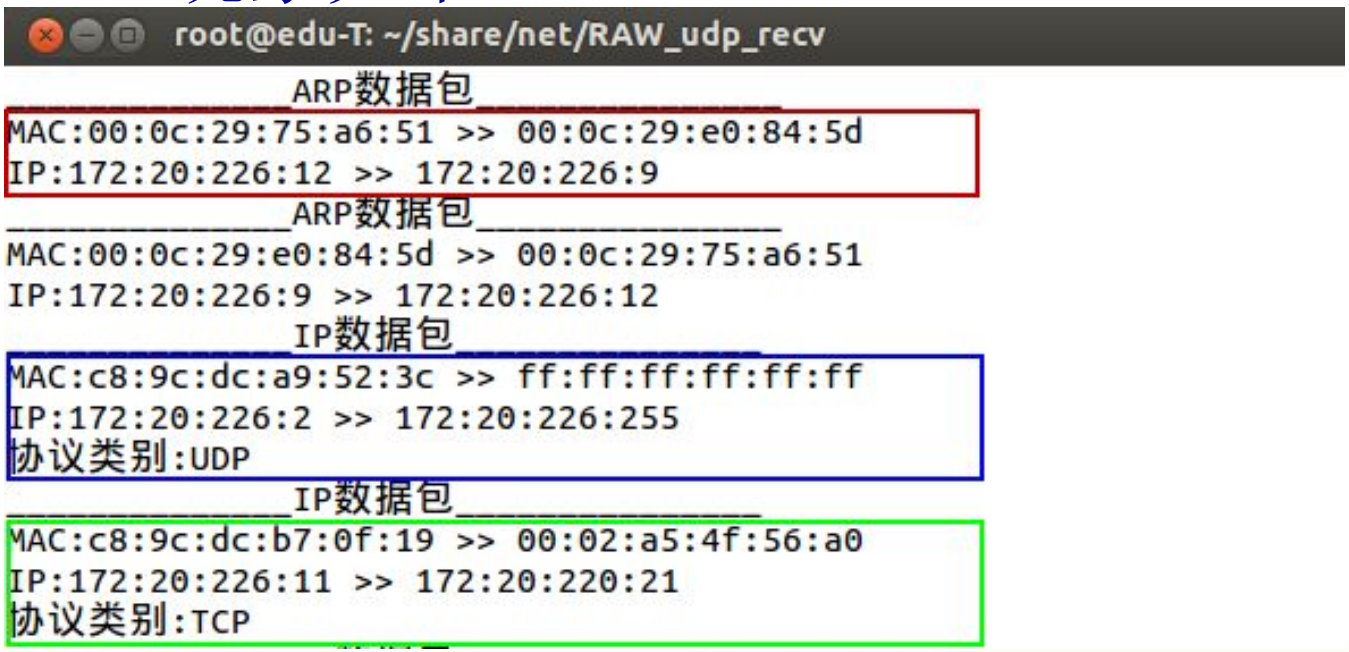
1. 创建链路层原始套接字
2. 获取链路层上的数据
3. 根据格式解析数据

大纲

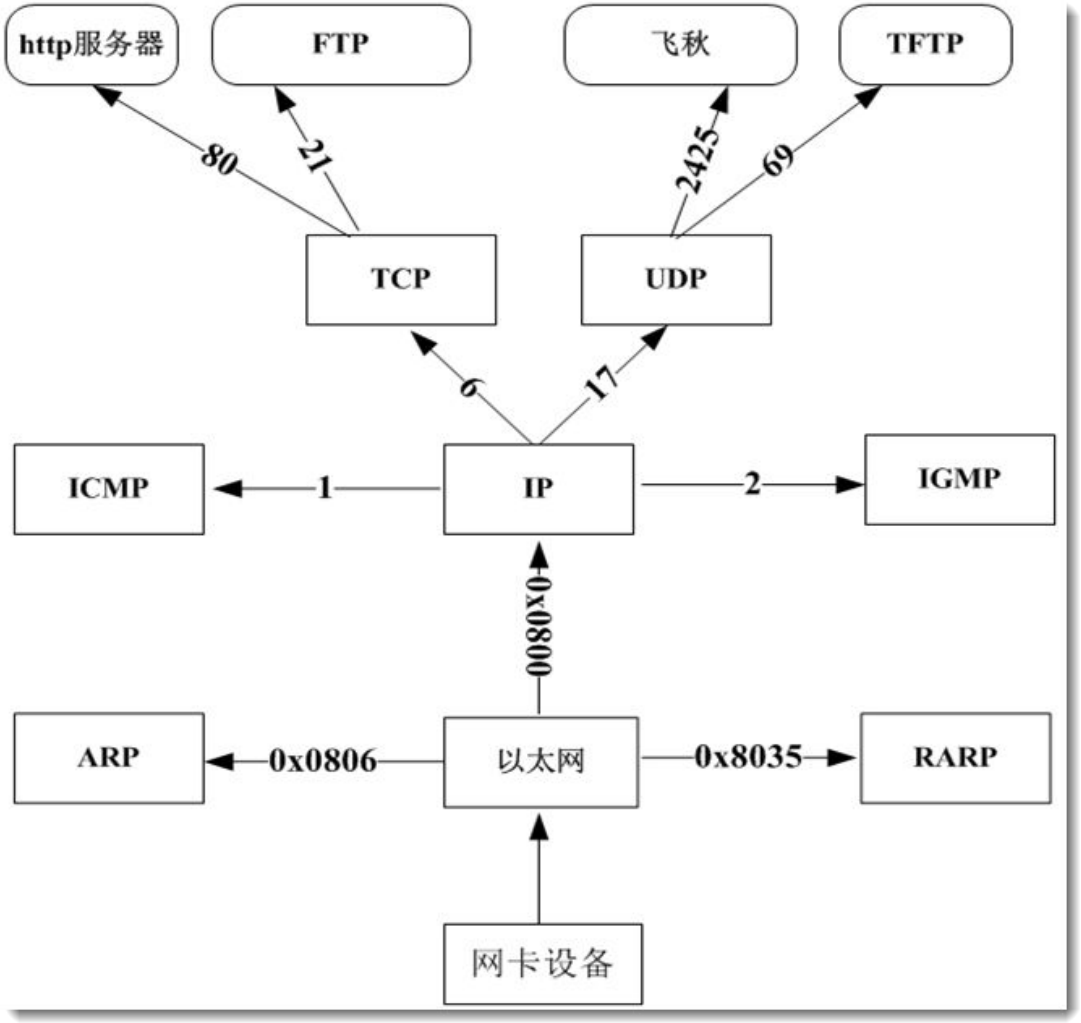
- TCP、UDP开发回顾
- 原始套接字概述、创建
- 数据包详解
- 编程实例—分析MAC数据包
- 练习—网络数据分析器
- sendto发送数据
- 练习—MAC地址扫描器（ARP）
- 练习—飞鸽欺骗（UDP）
- 练习—三次握手连接器（TCP）

网络数据分析器

- 在很多时候需要对网络上的数据进行抓取，然后进行分析，此“网络数据分析器”就是模仿现实开发中的抓包工具而进行的
- 运行demo现象如下：

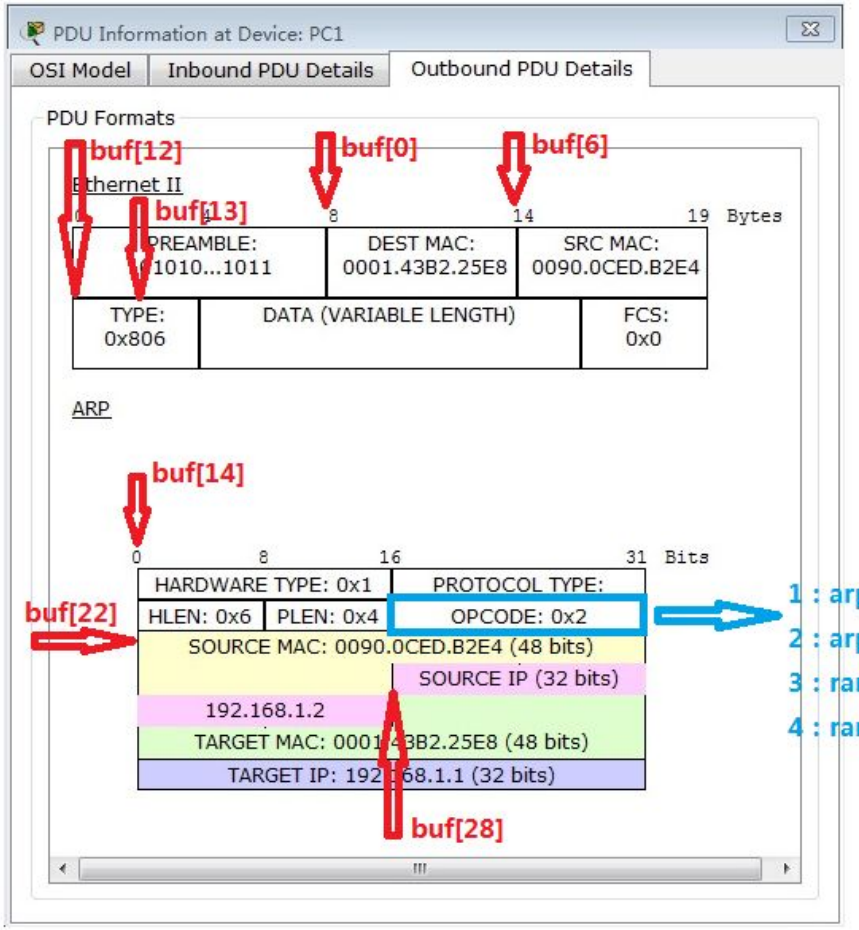


网络协议分析图



网络数据分析器

➢ ARP数据解析图

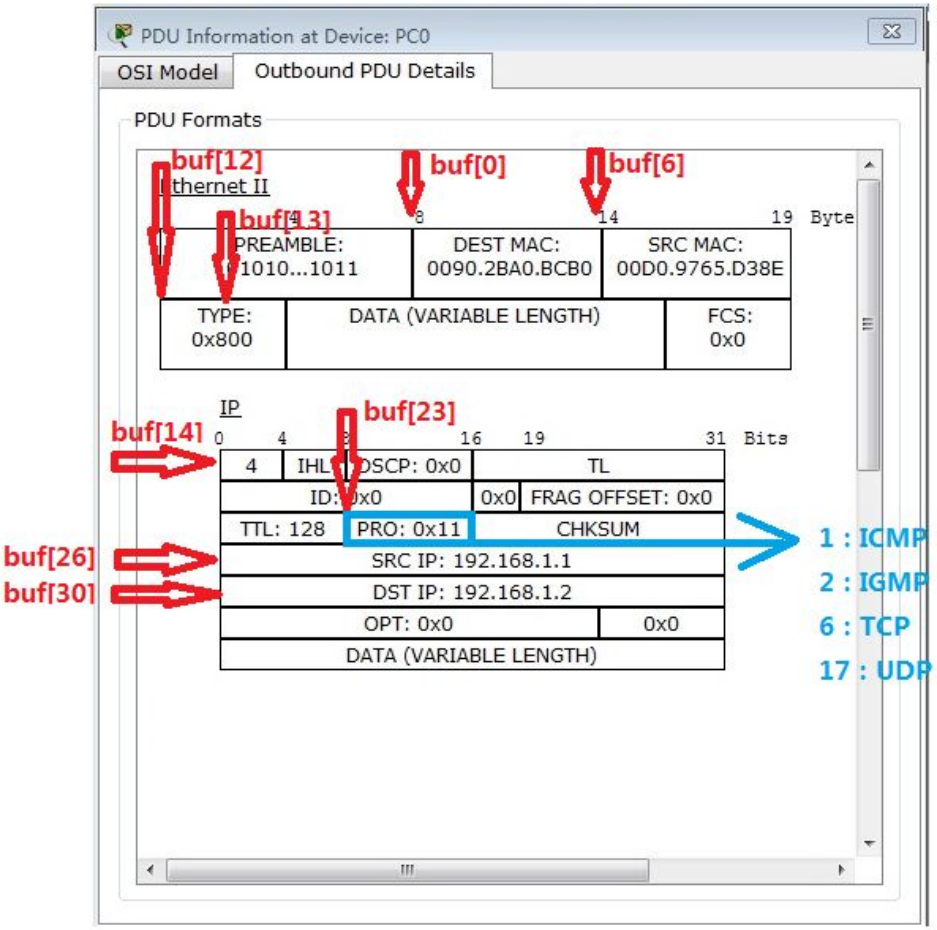


➢ 说明

- ARP的TYPE为0x0806
- buf为unsinged char
- 所有数据均为大端

网络数据分析器

➢ IP数据解析图

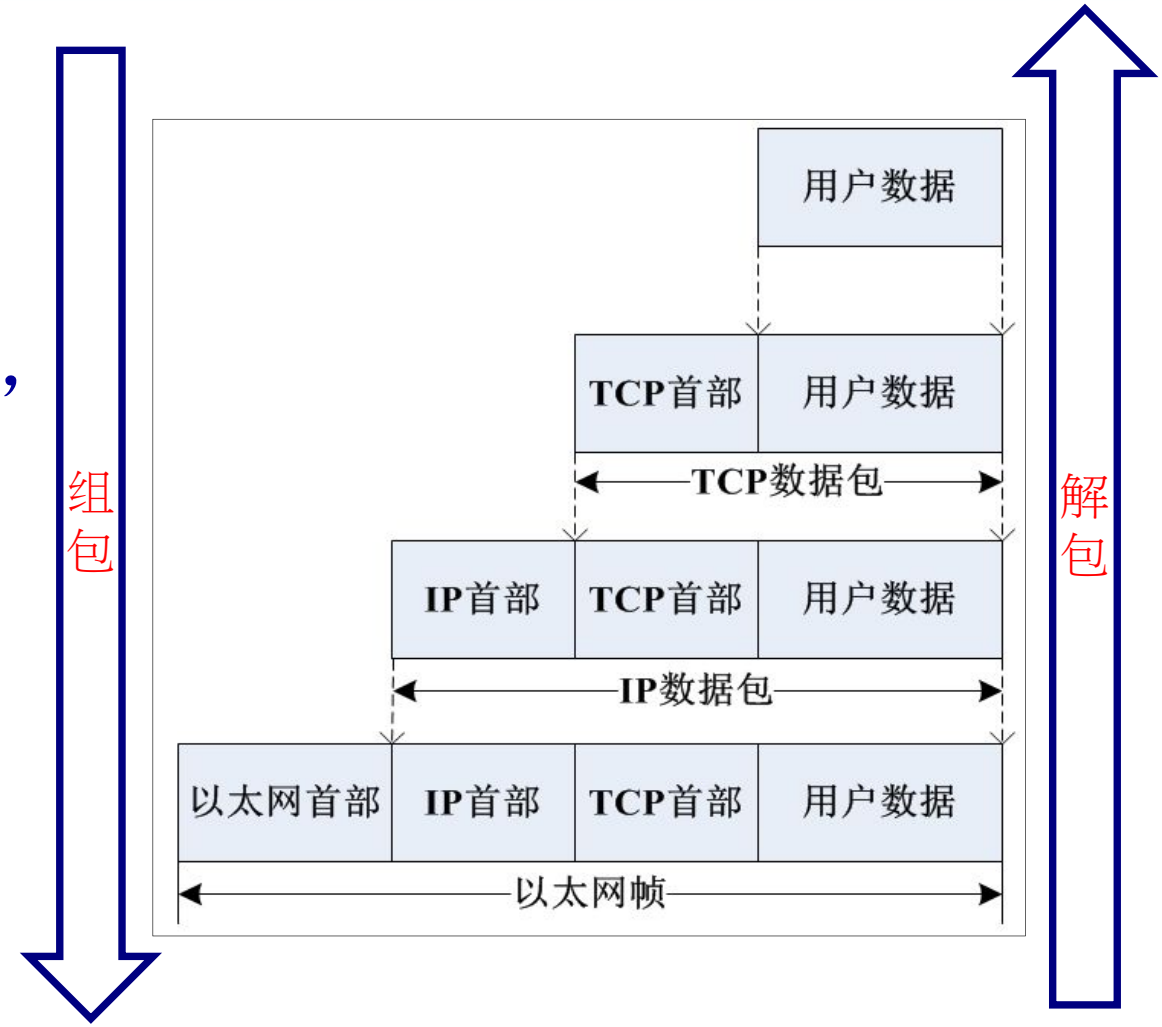


➢ 说明

- IP的TYPE为0x0800
- buf为unsinged char
- 所有数据均为大端

网络数据分析器

- 如右图所示，是网上的数据包的组包过程；其解包过程正好相反，首先分析以太网得到MAC然后再依次分析，比如IP、PORT等等



网络数据分析器——要求

- 要求
  - 分析出ARP/IP/RARP
  - 分析出MAC
- 扩展
  - 在完成基本要求的前提下，分析PORT
- 提示
  - 以root权限运行

06-01-data-analysis.c

- 想一想
  - 如何捕捉途经网卡的数据?

混杂模式

- 混杂模式
  - 指一台机器的网卡能够接收所有经过它的数据包，而不论其目的地址是否是它。
  - 一般计算机网卡都工作在非混杂模式下，如果设置网卡为混杂模式需要root权限
- linux下设置
  - 设置混杂模式：ifconfig eth0 promisc
  - 取消混杂模式：ifconfig eth0 -promisc
- windos下通过特定的软件实现

混杂模式

➢ linux下通过程序设置网卡混杂模式

```
struct ifreq ethreq;
strncpy(ethreq.ifr_name, "eth0", IFNAMSIZ);
if(ioctl(sock_raw_fd, SIOCGIFFLAGS, &ethreq) != 0)//获取eth0网络接口标志
{
    perror("ioctl");
    close(sock_raw_fd);
    exit(-1);
}
```

1. 获取网络接口标志

```
ethreq.ifr_flags |= IFF_PROMISC;
if(ioctl(sock_raw_fd, SIOCSIFFLAGS, &ethreq) != 0)//设置eth0网络接口标志
{
    perror("ioctl");
    close(sock_raw_fd);
    exit(-1);
}
```

2. 设置网络接口标志

大纲

- TCP、UDP开发回顾
- 原始套接字概述、创建
- 数据包详解
- 编程实例—分析MAC数据报
- 练习—网络数据分析器
- sendto发送数据
- 练习—MAC地址扫描器（ARP）
- 练习—飞鸽欺骗（UDP）
- 练习—三次握手连接器（TCP）

sendto发送数据

- 用sendto发送原始套接字数据
  - sendto(sock_raw_fd, msg, msg_len, 0, (struct sockaddr*)&sll, sizeof(sll));
- 注意：
  - sock_raw_fd：原始套接字
  - msg:发送的消息（封装好的协议数据）
  - sll:本机网络接口，指发送的数据应该从本机的哪个网卡出去，而不是以前的目的地址
- 想一想：如何定义sll?

sendto发送数据

- 本机网络接口
  - struct sockaddr_ll sll;
  - #include <netpacket/packet.h>
- struct sockaddr_ll

```
2  struct sockaddr_ll
3  {
4  unsigned short int sll_family;      /*一般为PF_PACKET*/
5  unsigned short int sll_protocol;    /*上层协议*/
6  int sll_ifindex;                    /*接口类型*/
7  unsigned short int sll_hatype;      /*报头类型*/
8  unsigned char sll_pkttype;          /*包类型*/
9  unsigned char sll_halen;            /*地址长度*/
10 unsigned char sll_addr[8];          /*MAC地址*/
11 };
```

- 只需要对sll.sll_ifindex赋值，就可使用

sendto发送数据

- 发送数据demo

```
26  /*将网络接口赋值给原始套接字地址结构*/
27  struct sockaddr_ll sll;
28  bzero(&sll, sizeof(sll));
29  sll.sll_ifindex = /*获取本机出去的接口地址*/;
30
31  int len = sendto(sock_raw_fd, msg, sizeof(msg), 0,\
32  (struct sockaddr*)&sll, sizeof(sll));
33
```

如何获得？

- 通过ioctl来获取网络接口地址
  - int ioctl(int fd, int request,void *)
  - #include <sys/ioctl.h>

sendto发送数据

- ioctl获取接口示例

```
18  struct ifreq ethreq;                                        //网络接口地址
19  strncpy(ethreq.ifr_name, "eth0", IFNAMSIZ);                 //指定网卡名称
20  if(-1 == ioctl(sock_raw_fd, SIOCGIFINDEX, &ethreq))        //获取网络接口
21  {
22      perror("ioctl");
23      close(sock_raw_fd);
24      exit(-1);
25  }
26
27  struct sockaddr_ll sll;
28  bzero(&sll, sizeof(sll));
29  sll.sll_ifindex = ethreq.ifr_ifindex;
30
31  int len = sendto(sock_raw_fd, msg, sizeof(msg), 0,\
32  (struct sockaddr*)&sll, sizeof(sll));
```

1. 获取网络接口

2. 给sll赋值

3. 发送

- 想一想：
  - ioctl的参数、struct ifreq结构类型

sendto发送数据

- ioctl参数对照表

类别	request	说明	数据类型
接口	SIOCGIFINDEX	获取网络接口	struct ifreq
	SIOCSIFADDR	设置接口地址	struct ifreq
	SIOCGIFADDR	获取接口地址	struct ifreq
	SIOCSIFFLAGS	设置接口标志	struct ifreq
	SIOCGIFFLAGS	获取接口标志	struct ifreq

```
18  struct ifreq ethreq;                                  //网络接口地址
19  strncpy(ethreq.ifr_name, "eth0", IFNAMSIZ); //指定网卡名称
20  ioctl(sock_raw_fd, SIOCGIFINDEX, &ethreq)   //获取网络接口
21
```

- struct ifreq：#include <net/if.h>
- IFNAMSIZ 16

sendto发送数据

➢ sendto发送数据的整体过程

```
18  struct ifreq ethreq;                                    //网络接口地址
19  strncpy(ethreq.ifr_name, "eth0", IFNAMSIZ);             //指定网卡名称
20  if(-1 == ioctl(sock_raw_fd, SIOCGIFINDEX, &ethreq)) //获取网络接口
21  {
22      perror("ioctl");
23      close(sock_raw_fd);
24      exit(-1);
25  }
26
27  struct sockaddr_ll sll;
28  bzero(&sll, sizeof(sll));
29  sll.sll_ifindex = ethreq.ifr_ifindex;
30
31  int len = sendto(sock_raw_fd, msg, sizeof(msg), 0,\
32  (struct sockaddr*)&sll, sizeof(sll));
```

1. 获取网络接口

2. 给sll赋值

3. 发送

大纲

- TCP、UDP开发回顾
- 原始套接字概述、创建
- 数据包详解
- 编程实例—分析MAC数据报
- 练习—网络数据分析器
- sendto发送数据
- 练习—MAC地址扫描器（ARP）
- 练习—飞鸽欺骗（UDP）
- 练习—三次握手连接器（TCP）

MAC地址扫描器

- 想一想：

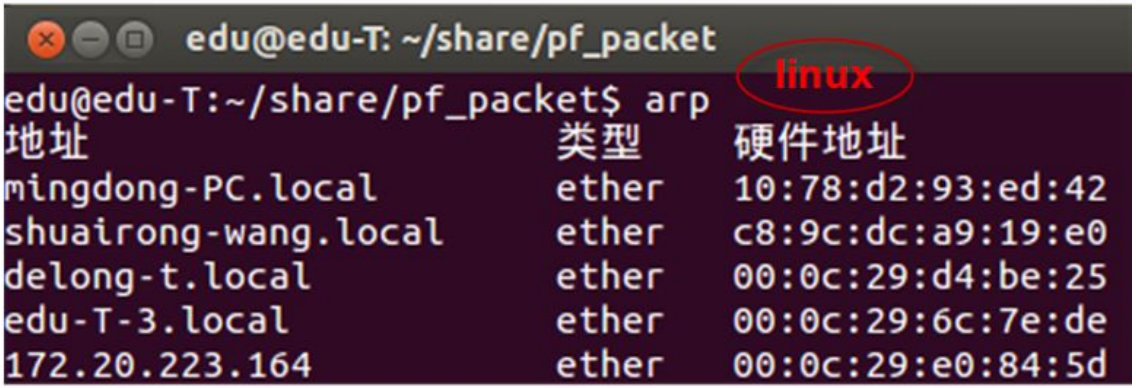
  如果A(192.168.1.1)向B(192.168.1.2)发送一个数据包，那么需要的条件有ip、port、使用的协议（TCP/UDP）之外还需要MAC地址，因为在以太网数据包中MAC地址是必须要有的；问怎样才能知道对方的MAC地址？使用什么协议呢？

MAC地址扫描器——协议

- ARP（Address Resolution Protocol，地址解析协议）
  - 是TCP/IP协议族中的一个
  - 主要用于查询指定ip所对应的的MAC
  - 请求方使用广播来发送请求
  - 应答方使用单播来回送数据
  - 为了在发送数据的时候提高效率在计算中会有一个ARP缓存表，用来暂时存放ip所对应的MAC，在linux中使用ARP即可查看,在xp中使用ARP -a

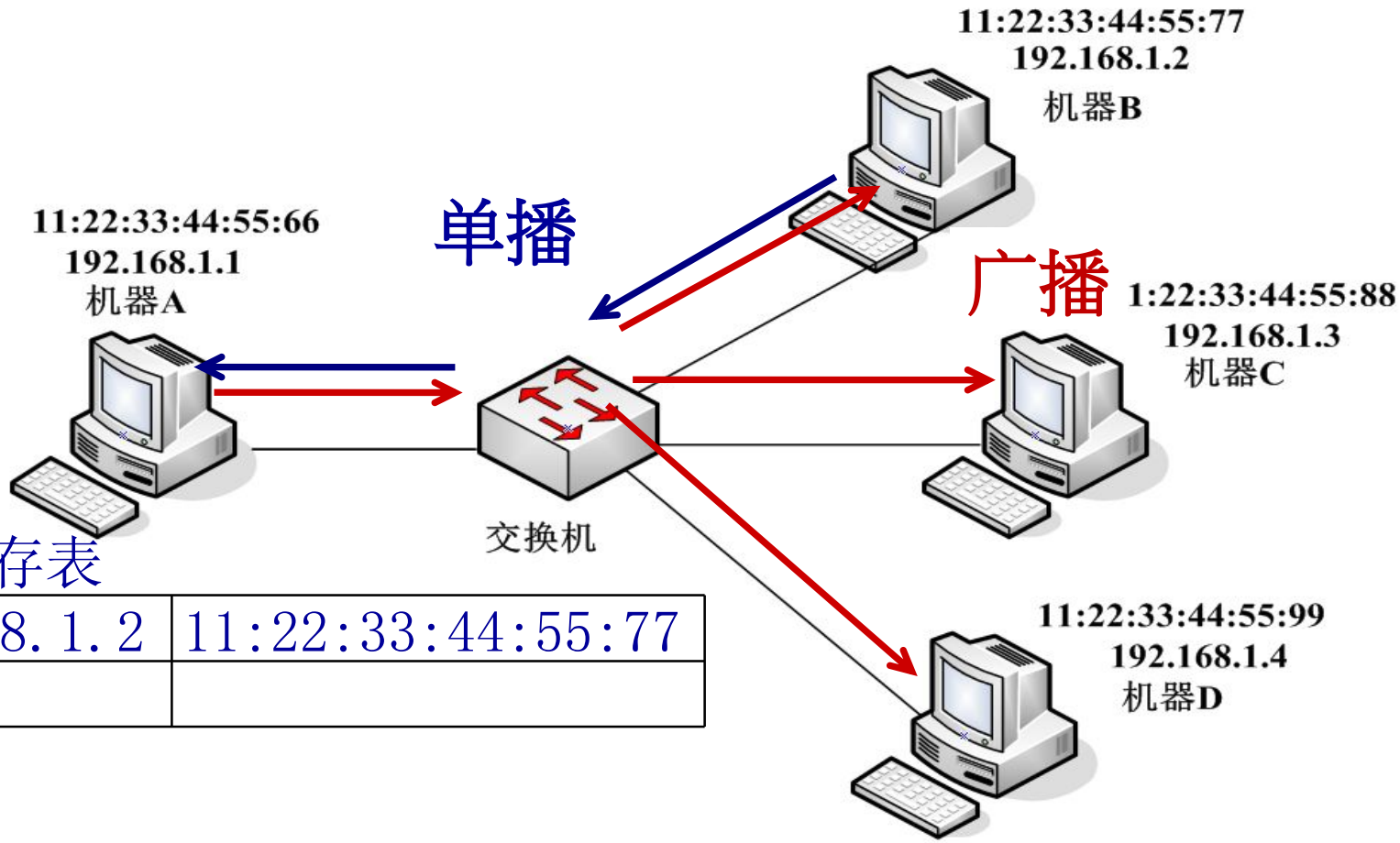
MAC地址扫描器——协议

- 在linux与xp系统下查看ARP的方式



MAC地址扫描器——原理

- 以机器A获取机器B的MAC为例

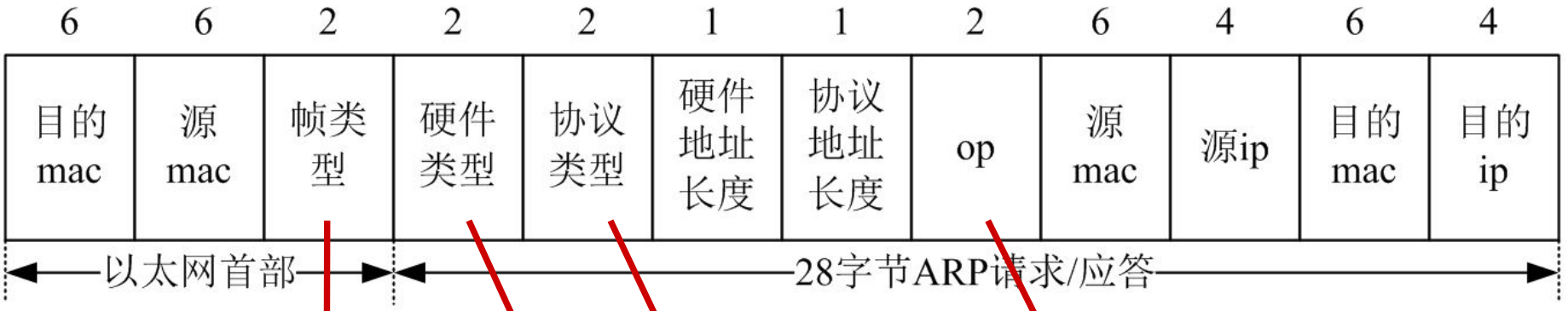


ARP缓存表

192.168.1.2	11:22:33:44:55:77

MAC地址扫描器——协议

➢ ARP协议格式



0x0800:IP
0x0806:ARP
0x8035:RARP

0x0800:IP

1:以太网地址

1:ARP请求
2:ARP应答
3:RARP请求
4:RARP应答

MAC地址扫描器——示例

- 向指定IP发送ARP请求(demo)
  - 获取172.20.226.11的MAC地址

```
11  int main(int argc,char *argv[])
12  {
13      //1.创建通信用的原始套接字
14      int sock_raw_fd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
15
16      //2.根据各种协议首部格式构建发送数据报
17      unsigned char send_msg[1024] = {
18          //--------------组MAC--------14------
19          0xff, 0xff, 0xff, 0xff, 0xff, 0xff, //dst_mac: FF:FF:FF:FF:FF:FF
20          0x00, 0x0c, 0x29, 0x75, 0xa6, 0x51, //src_mac: 00:0c:29:75:a6:51
21          0x08, 0x06,                         //类型: 0x0806 ARP协议
22
23          //--------------组ARP--------28-----
24          0x00, 0x01, 0x08, 0x00,             //硬件类型1(以太网地址),协议类型0x0800(IP)
25          0x06, 0x04, 0x00, 0x01,             //硬件、协议地址分别是6、4, op:(1: arp请求, 2: arp应答)
26          0x00, 0x0c, 0x29, 0x75, 0xa6, 0x51, //发送端的MAC地址
27          172,  20,   226,  12,               //发送端的IP地址
28          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, //目的MAC地址(由于要获取对方的MAC,所以目的MAC置零)
29          172,  20,   226,  11                //目的IP地址
30      };
```

MAC地址扫描器——示例

```
32      //3.数据初始化
33      struct sockaddr_ll sll;                    //原始套接字地址结构
34      struct ifreq ethreq;                       //网络接口地址
35      strncpy(ethreq.ifr_name, "eth0", IFNAMSIZ); //指定网卡名称
36      //4.将网络接口赋值给原始套接字地址结构
37      ioctl(sock_raw_fd, SIOCGIFINDEX, (char *)&ethreq);
38      bzero(&sll, sizeof(sll));
39      sll.sll_ifindex = ethreq.ifr_ifindex;
40      sendto(sock_raw_fd, send_msg, 42, 0 , (struct sockaddr *)&sll, sizeof(sll));
41
42      //5.接收对方的ARP应答
43      unsigned char recv_msg[1024] = "";
44      recvfrom(sock_raw_fd, recv_msg, sizeof(recv_msg), 0, NULL, NULL);
45      if(recv_msg[21] == 2)                      //ARP应答
46      {
47          char resp_mac[18] = "";      //arp响应的MAC
48          char resp_ip[16] = "";       //arp响应的IP
49          sprintf(resp_mac, "%02x:%02x:%02x:%02x:%02x:%02x", \
50          recv_msg[22],recv_msg[23],recv_msg[24],recv_msg[25],recv_msg[26],recv_msg[27]);
51
52          sprintf(resp_ip, "%d.%d.%d.%d", recv_msg[28], recv_msg[29], recv_msg[30], recv_msg[31]);
53          printf("IP:%s - MAC:%s\n",resp_ip, resp_mac);
54      }
55      return 0;
56  }
```

MAC地址扫描器——要求

- 要求

  获取到当前网段中所有机器的MAC地址

- 提示
  - 每次指定一个机器发送MAC请求，通过发送多次ARP，即可得到所有的机器的MAC
  - ARP的发送和接收各使用一个线程

MAC地址扫描器——demo

- MAC地址扫描器运行现象如下图所示

```
edu@edu-T: ~/share/6-day-AF_INET
edu@edu-T:~/share/6-day-AF_INET$ sudo ./a.out
172.20.223.254  -  00:02:a5:4f:56:a0
172.20.223.250  -  00:3e:2a:d6:49:f8
172.20.223.244  -  5c:63:bf:a2:08:fa
172.20.223.245  -  00:0c:29:5b:12:a8
172.20.223.218  -  08:90:90:90:90:90
172.20.223.216  -  c8:9c:dc:fd:94:72
172.20.223.210  -  12:25:34:21:65:54
172.20.223.178  -  c8:9c:dc:ba:68:f7
172.20.223.154  -  00:09:c0:ff:ec:48
172.20.223.153  -  00:09:c0:ff:ec:48
172.20.223.148  -  00:1e:90:2d:65:19
```

06-02-sock_raw_arp.c

大纲

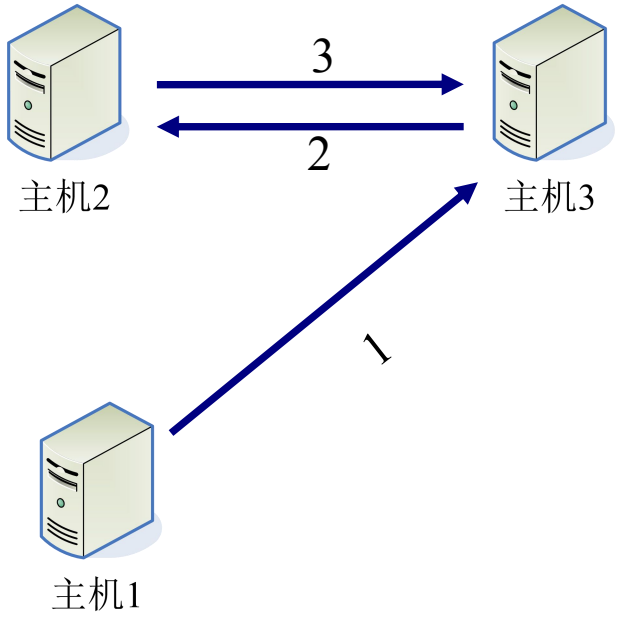
- TCP、UDP开发回顾
- 原始套接字概述、创建
- 数据包详解
- 编程实例—分析MAC数据报
- 练习—网络数据分析器
- sendto发送数据
- 练习—MAC地址扫描器（ARP）
- 练习—飞鸽欺骗（UDP）
- 练习—三次握手连接器（TCP）

飞鸽欺骗（UDP）

- 飞鸽格式
  - 版本:用户名:主机名:命令字:附加消息
- 演示：

飞鸽欺骗（UDP）

- 目的：挑拨离间
- 组包过程：



MAC头部	IP头部	UDP头部	消息

- 飞鸽消息格式

```
sprintf(msg, "1:%d:%s:%s:%d:%s", \
        123,"sunplusedu", "sunplusedu",32,"ok");
```

- 注意：msg指的是udp报文头中的数据

飞鸽欺骗（UDP）

- MAC、IP、UDP报文头参考前面的数据包详解
  - 在对UDP校验的时候需要在UDP报文之间加上伪头部
  - IP校验的时候不需要伪头部

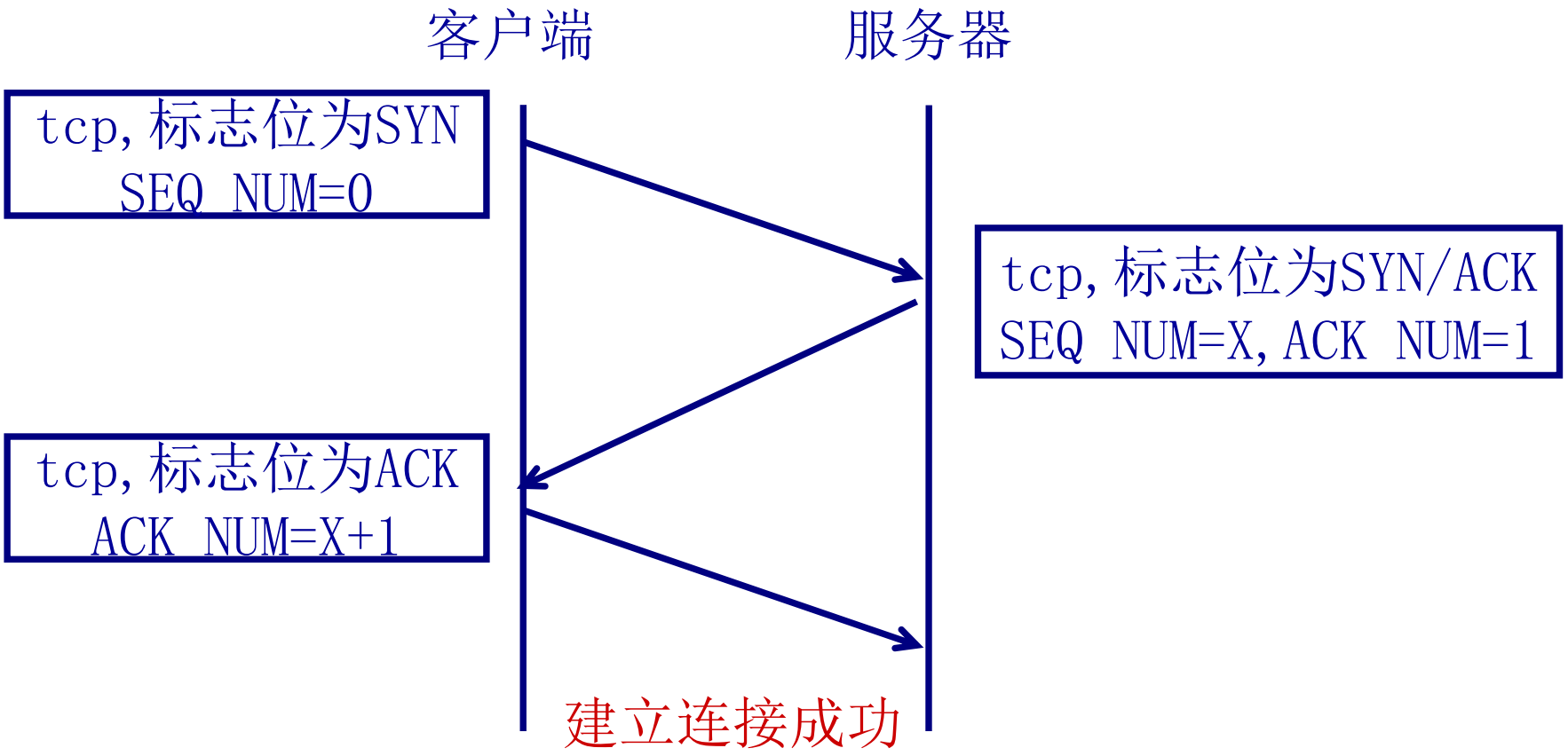
06-03-sock_raw_udp_fq.c

大纲

- TCP、UDP开发回顾
- 原始套接字概述、创建
- 数据包详解
- 编程实例—分析MAC数据报
- 练习—网络数据分析器
- sendto发送数据
- 练习—MAC地址扫描器（ARP）
- 练习—飞鸽欺骗（UDP）
- 练习—三次握手连接器（TCP）

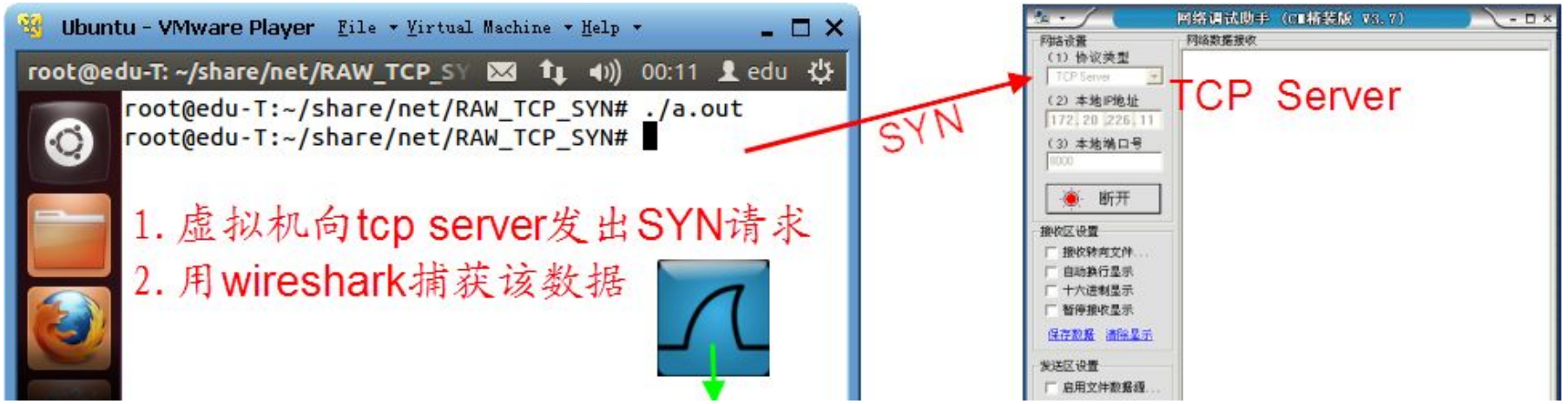
三次握手连接器

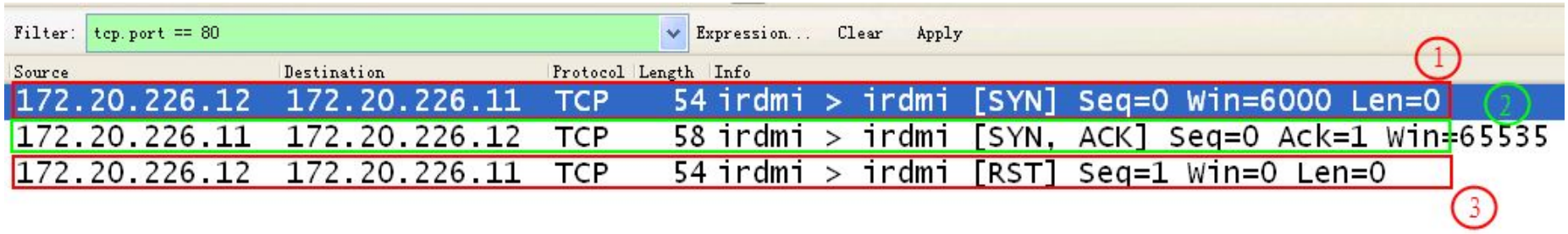
- TCP数据包（发送一个SYN数据包）



三次握手连接器

- 发送一个SYN数据包





06-04-sock_raw_tcp_syn.c

凌阳教育
www.sunplusedu.com

值得信赖的教育品牌

Tel: 400-705-9680 , Email: edu@sunplusapp.com , BBS: bbs.sunplusedu.com