实验四 静态路由编程实现

周华平

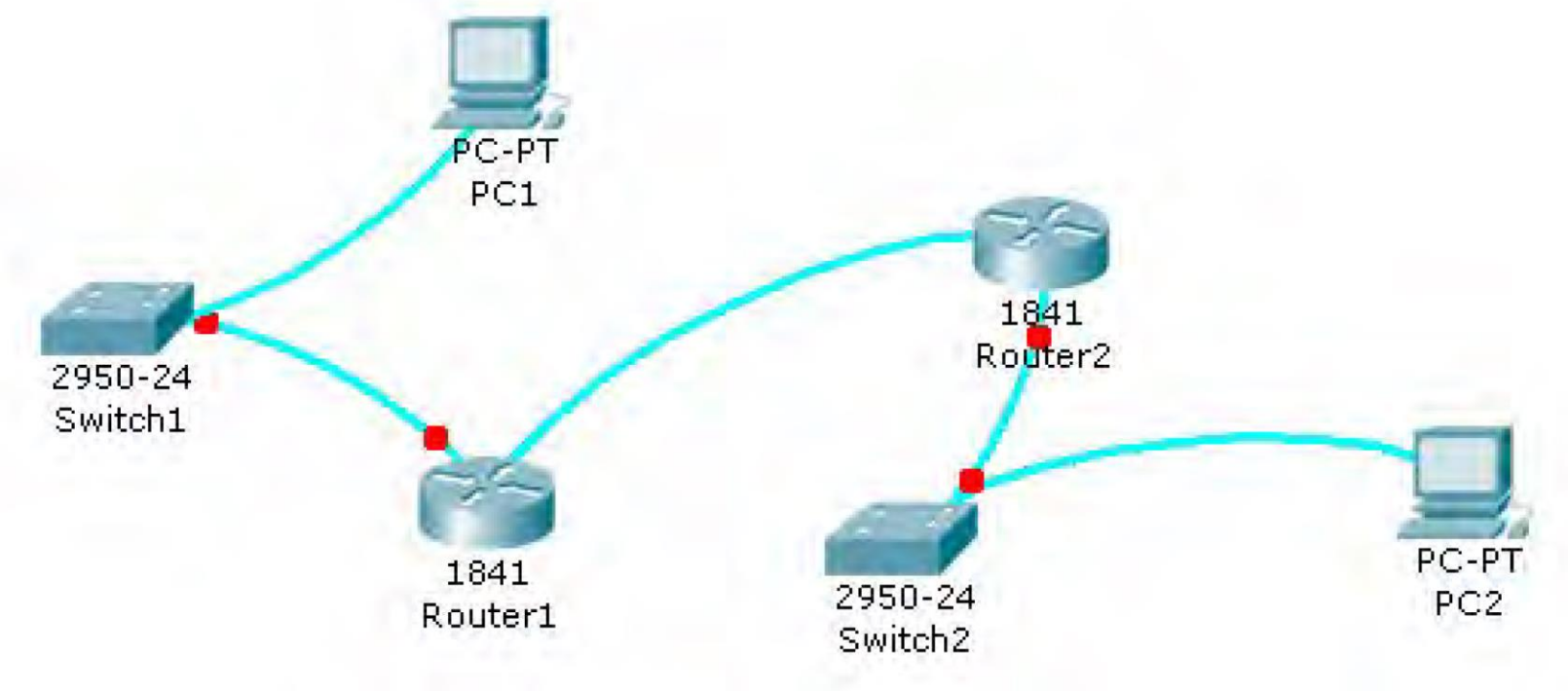
2018.5.3

实验课日程表

实验课日期	实验任务	实验报告截止时间
2018-03-08	实验1 基本网络工具集使用和协议数据单元观测	2018-03-21 23:59:59
2018-03-22	实验2 RAW SOCKET编程与以太网帧分析基础	2018-04-18 23:59:59
2018-04-05	清明假期	
2018-04-19	实验3 子网划分和NAT 配置	2018-05-06 23:59:59
2018-05-03	实验4 静态路由编程实现	2018-05-16 23:59:59
2018-05-17		2018-05-30 23:59:59
2018-05-31		2018-06-13 23:59:59
2018-06-14		2018-06-24 23:59:59
实验课日期仅供参考，如有变动，以群公告为准　田臣老师班课程QQ群：724791341		

实验拓扑

- 我们要在Router1、Router2上实现静态路由的编程，PC1、PC2实现ICMP的收发程序



Router的结构

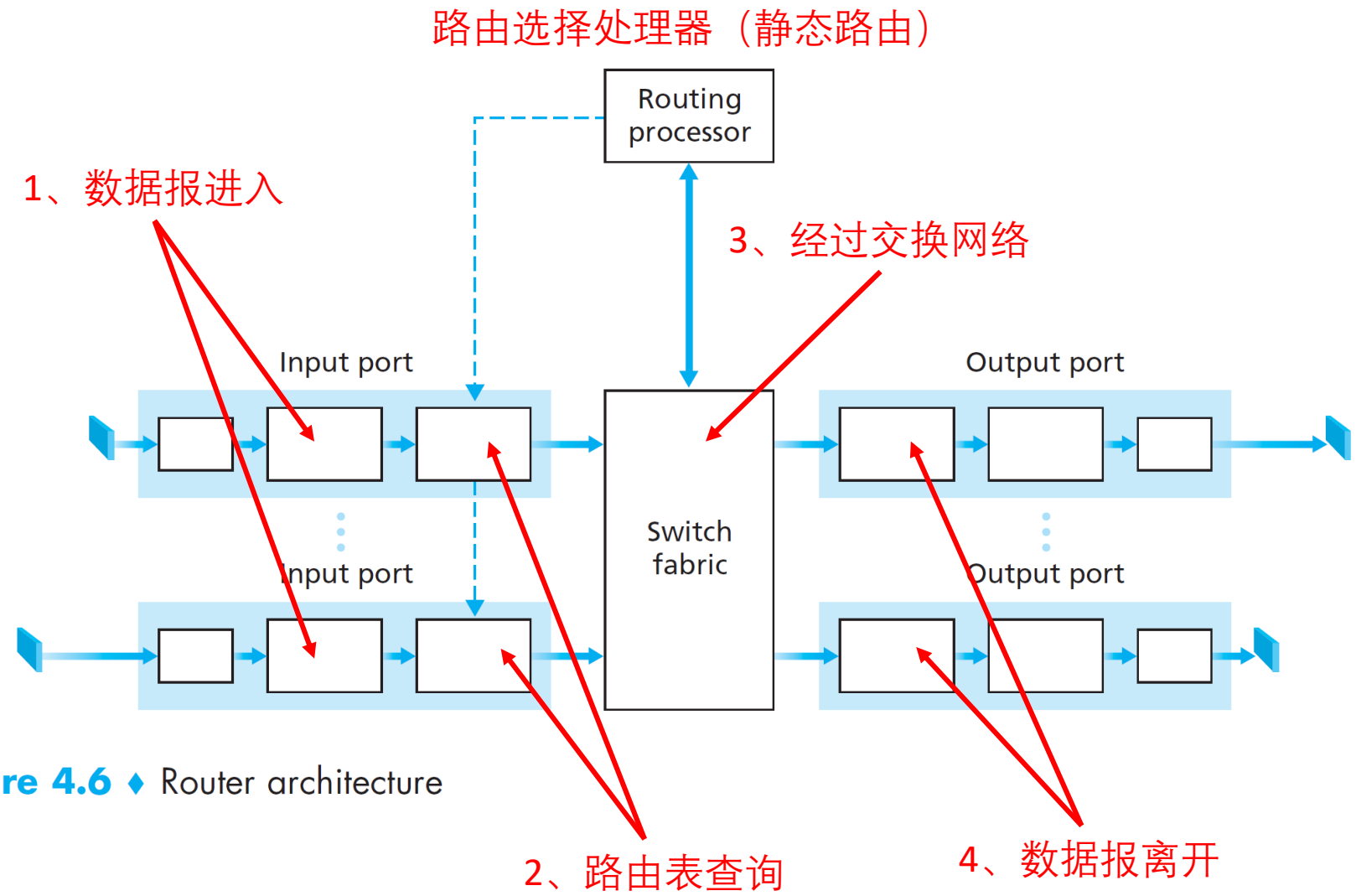


Figure 4.6 ♦ Router architecture

Router的三个表项

- Router维护3个表项：路由表(route/ip route)，ARP表(arp)，设备表(ifconfig)
- 路由表用来查找转发的设备接口，需要手工配置
- 设备表用来描述IP地址和物理接口的绑定关系，需要手工配置
- ARP表用来查找下一跳的MAC地址，需要通过ARP协议生成

ARP协议(选做)

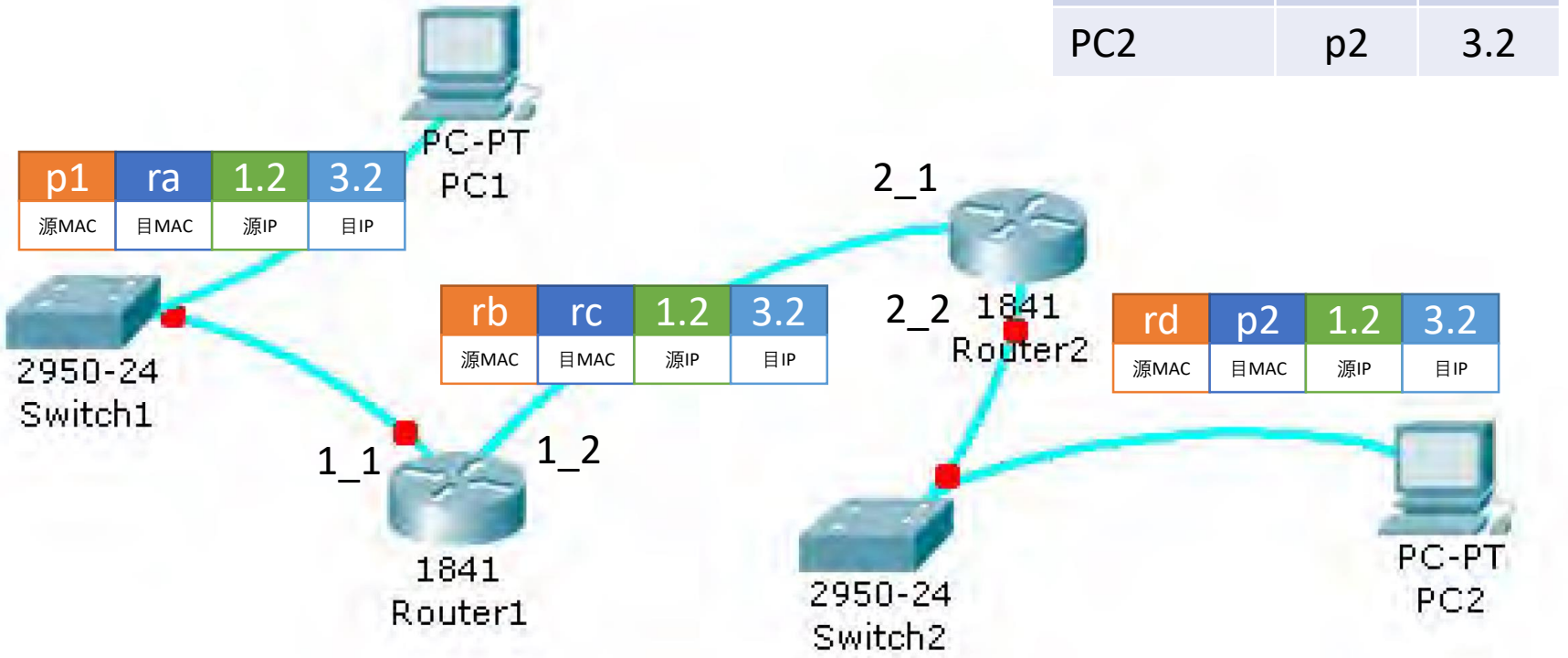
- 路由表中的下一跳地址为IP地址
- 我们需要通过该IP地址找到对应物理接口的MAC地址：IP Address->MAC Address

```
▸ Ethernet II, Src: Vmware_d6:ad:2a (00:0c:29:d6:ad:2a), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Vmware_d6:ad:2a (00:0c:29:d6:ad:2a)
    Sender IP address: 192.168.65.128
    Target MAC address: Broadcast (ff:ff:ff:ff:ff:ff)
    Target IP address: 192.168.65.2
```

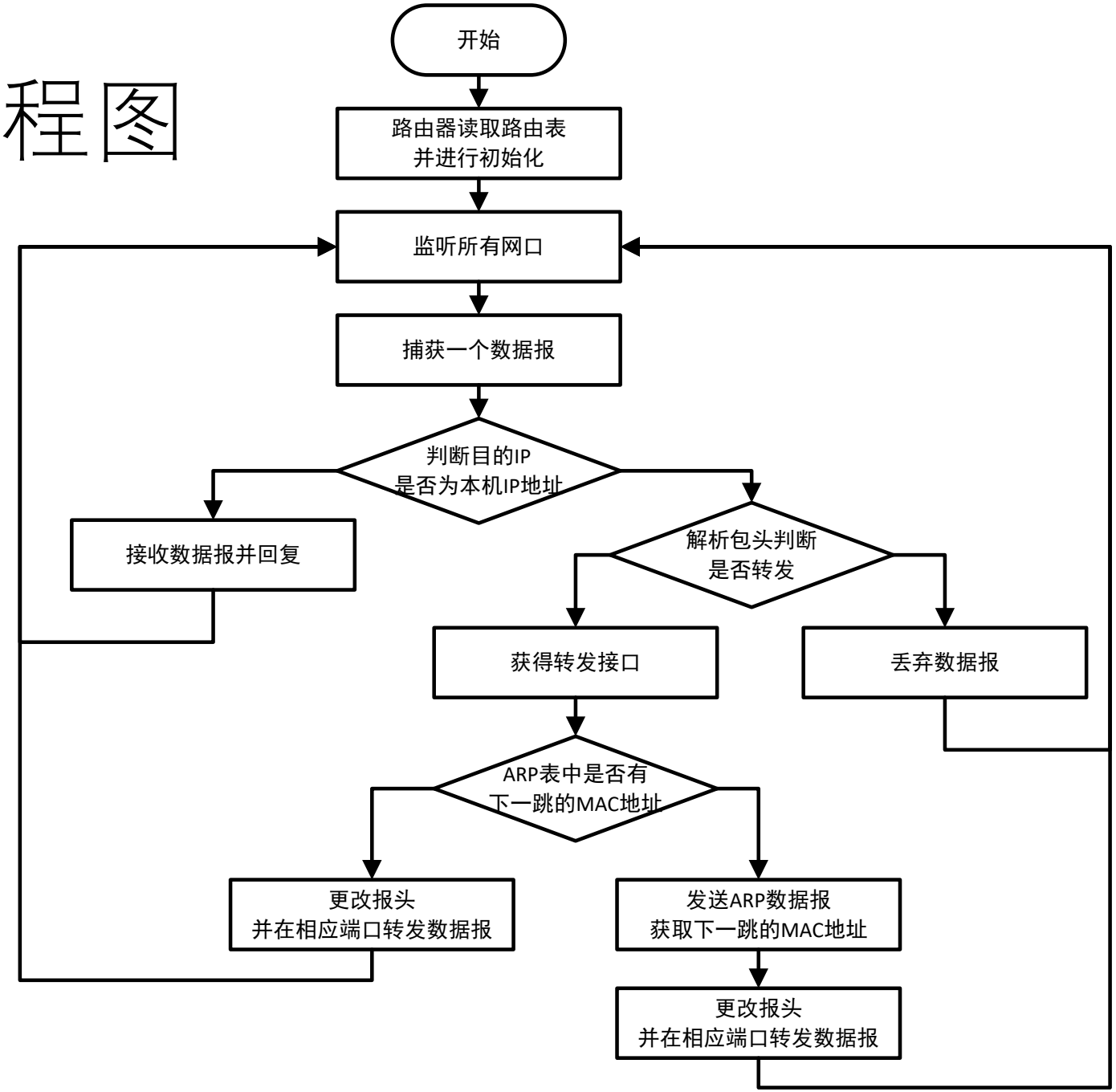
```
▸ Ethernet II, Src: Vmware_ea:0d:e1 (00:50:56:ea:0d:e1), Dst: Vmware_d6:ad:2a (00:0c:29:d6:ad:2a)
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: Vmware_ea:0d:e1 (00:50:56:ea:0d:e1)
    Sender IP address: 192.168.65.2
    Target MAC address: Vmware_d6:ad:2a (00:0c:29:d6:ad:2a)
    Target IP address: 192.168.65.128
```

数据报的变化

	MAC	IP
Router1_1	ra	1.1
Router1_2	rb	2.1
Router2_1	rc	2.2
Router2_2	rd	3.1
PC1	p1	1.2
PC2	p2	3.2



程序流程图


开始
路由器读取路由表
并进行初始化
监听所有网口
捕获一个数据报
判断目的IP
是否为本机IP地址
接收数据报并回复
解析包头判断
是否转发
获得转发接口
丢弃数据报
ARP表中是否有
下一跳的MAC地址
更改报头
并在相应端口转发数据报
发送ARP数据报
获取下一跳的MAC地址
更改报头
并在相应端口转发数据报

发送链路层数据帧

```
#include <sys/socket.h>
#include <linux/if_packet.h>
// 指定SOCK_DGRAM类型则发送/接收的数据包会自动添加/去除以太网帧头部
int sockfd = socket(AF_PACKET, SOCK_DGRAM, htons(ETH_P_IP));
struct sockaddr_ll dest_addr = {
        .sll_family = AF_PACKET,
        .sll_protocol = htons(ETH_P_IP),
        .sll_halen = ETH_ALEN,
        .sll_ifindex = if_index,
};
memcpy(&dest_addr.sll_addr, &dest_mac_addr, ETH_ALEN);
sendto(sockfd, buffer, nbytes, 0, (struct sockaddr *)
&dest_addr, sizeof(dest_addr));
```

接收链路层数据帧

```
#include <sys/socket.h>
#include <linux/if_packet.h>
struct sockaddr_ll addr;
socklen_t addr_len = sizeof(addr);
// addr中保存了链路层发送端的地址信息
recvfrom(sockfd, buffer, BUF_LEN, 0, (struct sockaddr *) &addr,
&addr_len));
```

- Raw Socket上调用`recvfrom()`默认会接收所有本机发送&接收的数据包(包括loopback设备)
- 需要根据`addr`中的相关字段对数据包来源进行判断(eg: `sll_hatype`, `sll_pkttype`, etc.)

获取物理接口信息

```
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/if.h>
const char *if_name = "eth0";
struct ifreq req;
memset(&req, 0, sizeof(req));
strncpy(req.ifr_name, if_name, IFNAMSIZ - 1);

int sockfd = socket(AF_PACKET, SOCK_DGRAM, htons(ETH_P_IP));
// get interface index
ioctl(sockfd, SIOCGIFINDEX, &req);
int if_index = req.ifr_ifindex;
// get mac addr
ioctl(sockfd, SIOCGIFHWADDR, &req);
memcpy(mac_addr, req.ifr_hwaddr.sa_data, ETH_ALEN);
```

IP相关数据结构

```
#include <netinet/in.h>

struct in_addr; // 表示ip地址的数据结构


#include <arpa/inet.h>

/* Convert Internet host address from numbers-and-dots notation
in CP into binary data and store the result in the structure
INP. */

int inet_aton (const char *cp, struct in_addr *inp);

/* Convert Internet number in IN to ASCII representation. The
return value is a pointer to an internal array containing the
string. */

char *inet_ntoa (struct in_addr in);
```

Header数据结构

```
#include <net/ethernet.h>
struct ether_header *eth_header;

#include <net/if_arp.h>
struct arphdr *arp_header;

#include <netinet/ip.h>
struct ip *ip_header;

#include <netinet/ip_icmp.h>
struct icmphdr *icmp_header;
```

字节序转换

```
#include <netinet/in.h>

// ntoh: network to host
uint32_t ntohl (uint32_t netlong);
uint16_t ntohs (uint16_t netshort);
// hton: host to network
uint32_t htonl (uint32_t hostlong);
uint16_t htons (uint16_t hostshort);
```

最后

- 实验报告截止时间：2018-05-16 23:59:59
- 本次实验相比于前三次实验难度有所提升，所以希望同学们尽早动手开始写