

# 计算机网络编程

## 第6章 IP数据包的捕获与解析

信息工程学院 方徽星  
fanghuixing@hotmail.com

# 大纲

- 设计目的
- 相关知识
- 例题分析

# 1. 设计目的

- IP包是网络层中进行数据传输的基本单位
- 熟悉IP包结构, 具有重要的意义
  - 理解网络协议的概念
  - 网络层次结构
  - 协议执行过程以及
  - 网络问题处理的一般方法
- 通过截获与解析标准格式的IP包
  - 了解头部中各个字段的含义与用途
  - 深入理解网络协议的工作原理

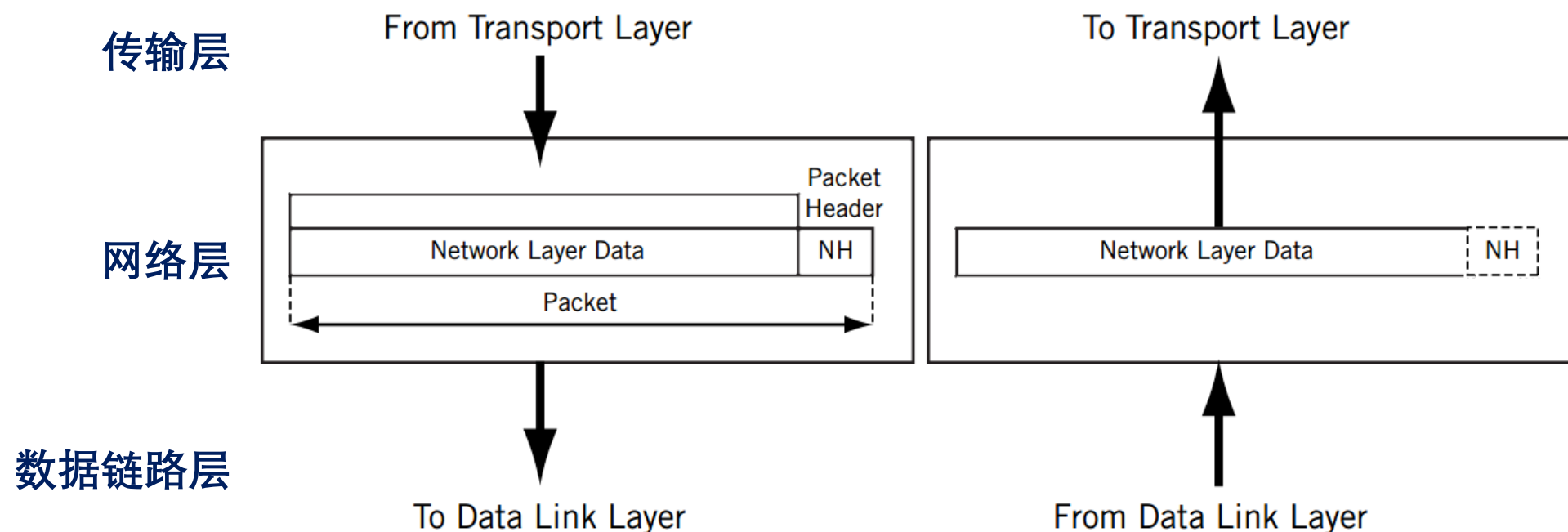
## OSI参考模型



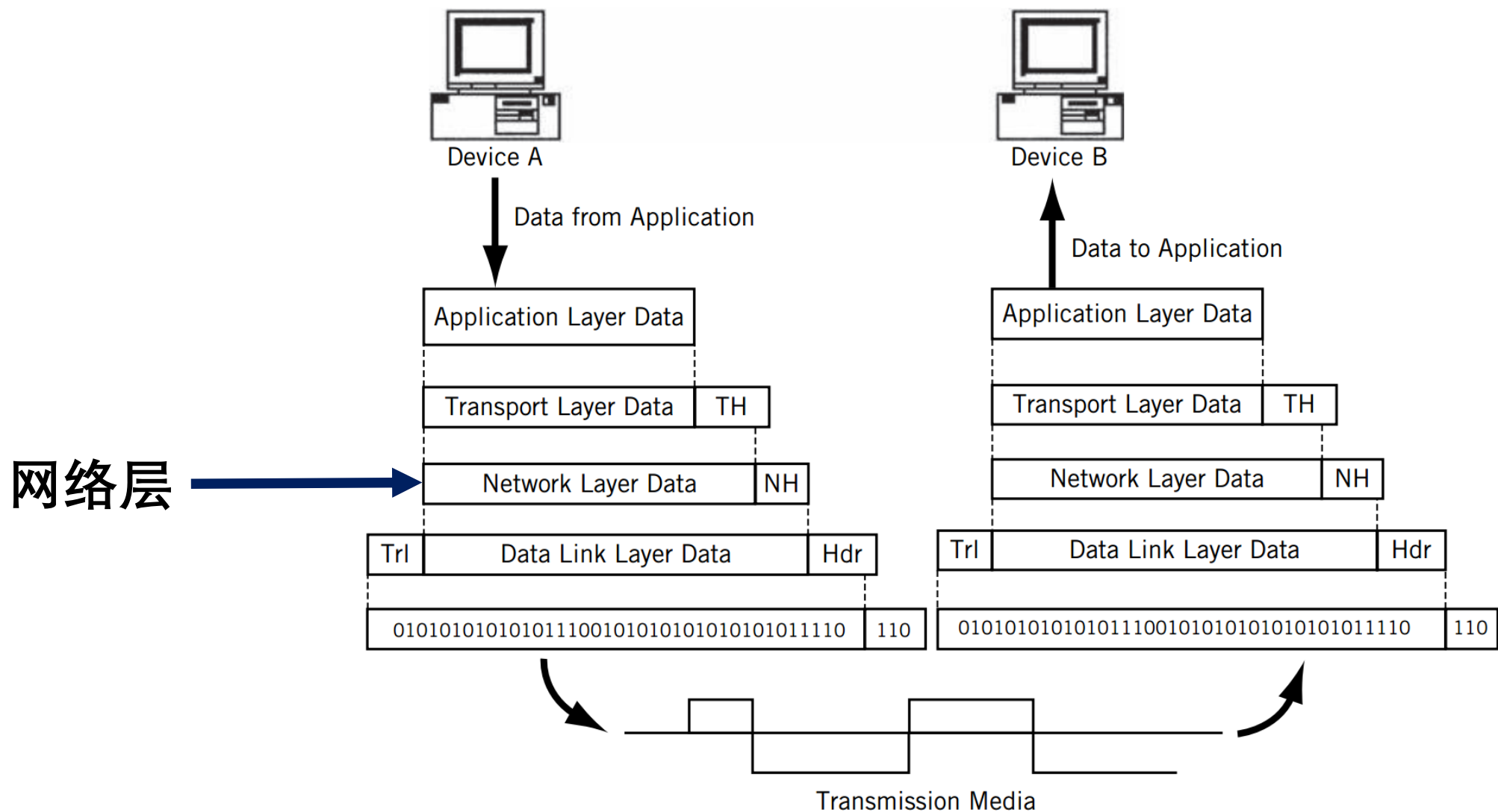
**IP包(分组)**

## 2. 相关知识—网络层的基本概念

- 网络层主要功能：
  - 通过路由选择算法，为分组通过通信子网选择最适当的路径
  - 实现拥塞控制、流量控制与网络互联等功能

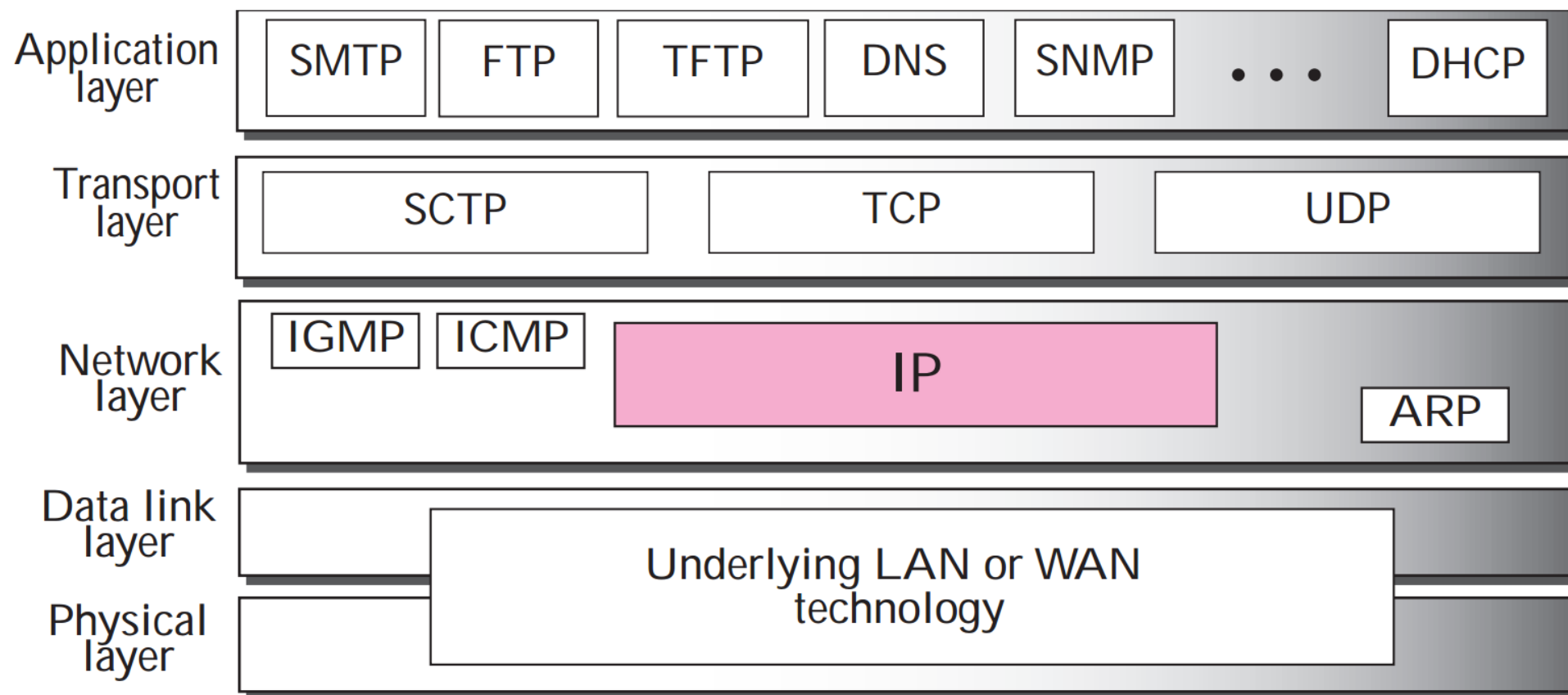


## 2. 相关知识—网络层的基本概念



## 2. 相关知识—网络层的基本概念

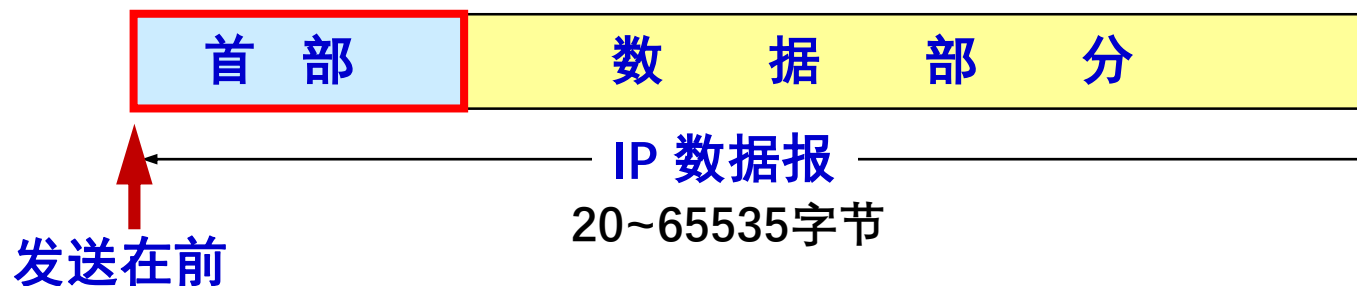
- IP协议是TCP/IP协议体系中的核心协议



## 2. 相关知识—网络层的基本概念

- TCP/IP协议体系主要特点
  - 开放的协议标准
  - 独立于特点的计算机硬件与操作系统
  - 独立于特定的网络硬件，可运行在局域网、广域网
  - 统一的网络地址分配方案，所有网络设备在Internet中都有唯一地IP地址
  - 标准化的应用层协议，可提供多种拥有大量用户的网络服务

## 2. 相关知识—IP数据包的结构





## 2. 相关知识—IP数据包的结构



- 版本——占 4 位，指 IP 协议的版本
- 目前的 IP 协议版本号为 4 (即 IPv4)

## 2. 相关知识—IP数据包的结构



- 首部长度——占 4 位，可表示的最大数值是 15(单位为 4 字节)
- 因此 IP 的首部长度的最大值是 60 字节

## 2. 相关知识—IP数据包的结构



- 区分服务——占 8 位，用来获得更好的服务，在旧标准中叫做服务类型，但实际上一直未被使用过
- 1998 年改名为区分服务，只有在使用区分服务（DiffServ）时，这个字段才起作用
- 一般的情况下都不使用这个字段

## 2. 相关知识—IP数据包的结构



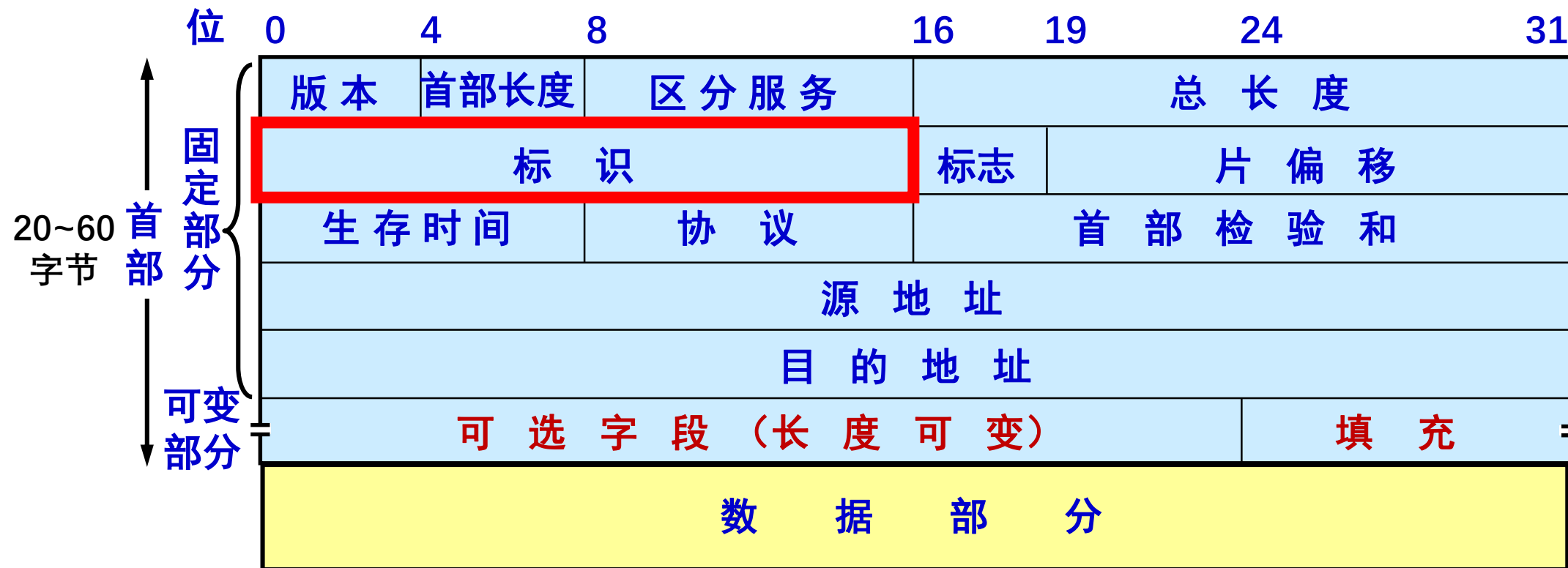
- 总长度——占 16 位，指首部和数据之和的长度，
- 单位为字节，因此数据报的最大长度为 65535 字节。
- 总长度必须不超过最大传送单元 MTU

## 2. 相关知识—IP数据包的结构

典型MTU数值

Link Protocol	Typical MTU Limit	Maximum IP Packet
Ethernet	1518	1500
IEEE 802.3	1518	1492
<i>Gigabit Ethernet</i>	9018	9000
IEEE 802.4	8191	8166
<i>IEEE 802.5 (Token Ring)</i>	4508	4464
FDDI	4500	4352
SMDS/ATM	9196	9180
<i>Frame relay</i>	4096	4091
<i>SDLC</i>	2048	2046

## 2. 相关知识—IP数据包的结构



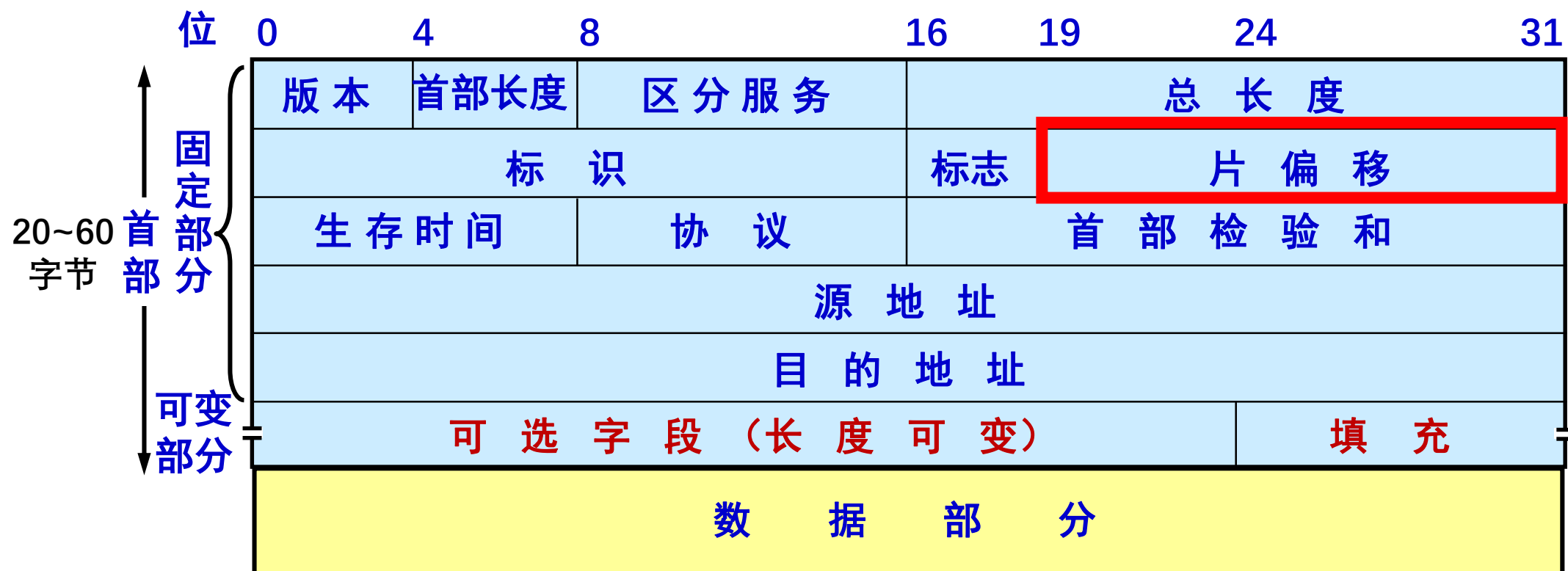
- 标识(identification) ——占 16 位
- 它是一个计数器，用来产生 IP 数据包的标识

## 2. 相关知识—IP数据包的结构



- 最低位是 MF (More Fragment)
- MF = 1 表示后面“还有分片”；MF = 0 表示最后一个分片
- 中间位是 DF (Don't Fragment)
- 只有当 DF = 0 时才允许分片

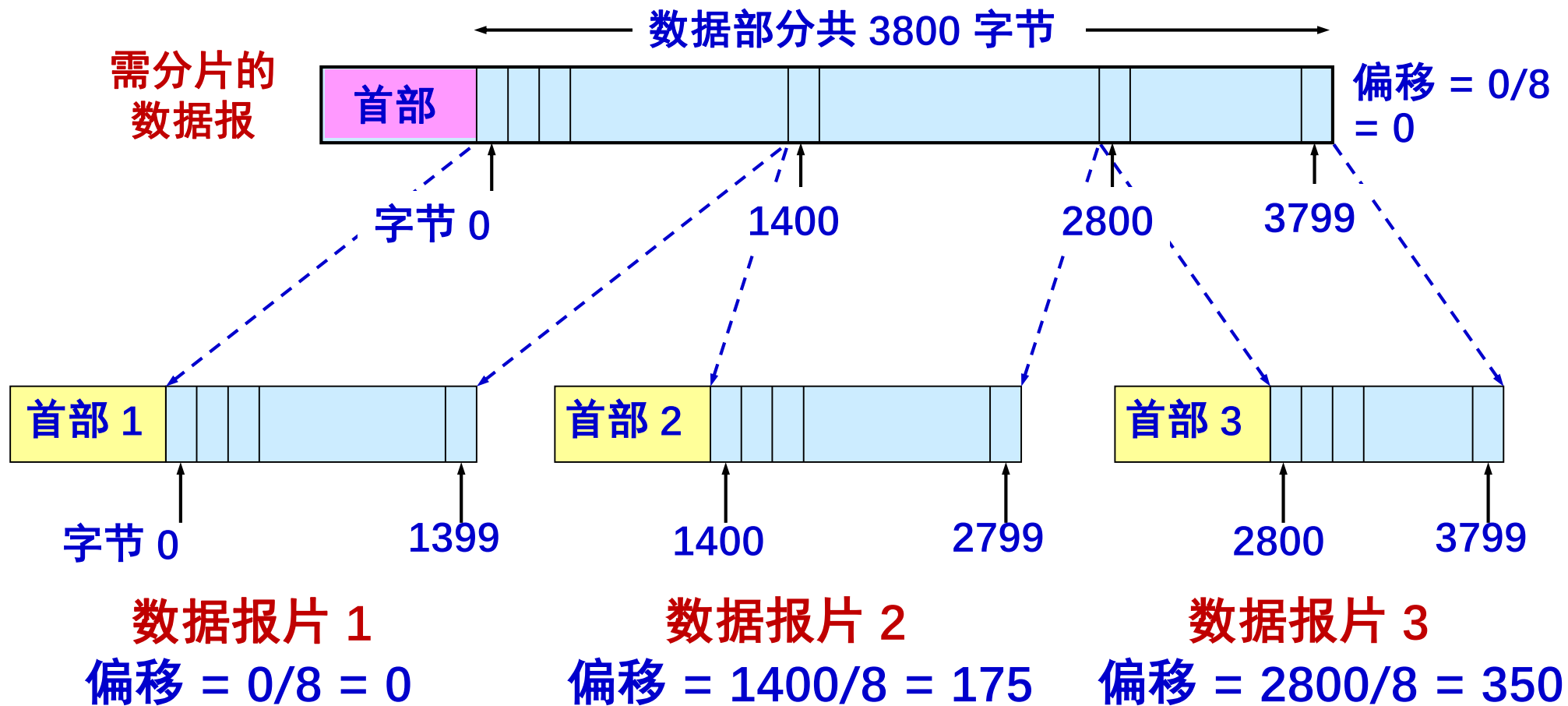
## 2. 相关知识—IP数据包的结构



- 片偏移——占13 位
- 指出较长的分组在分片后，某片在原分组中的相对位置
- 片偏移以 8 个字节为偏移单位



## 2. 相关知识—IP数据包的结构



## 2. 相关知识—IP数据包的结构



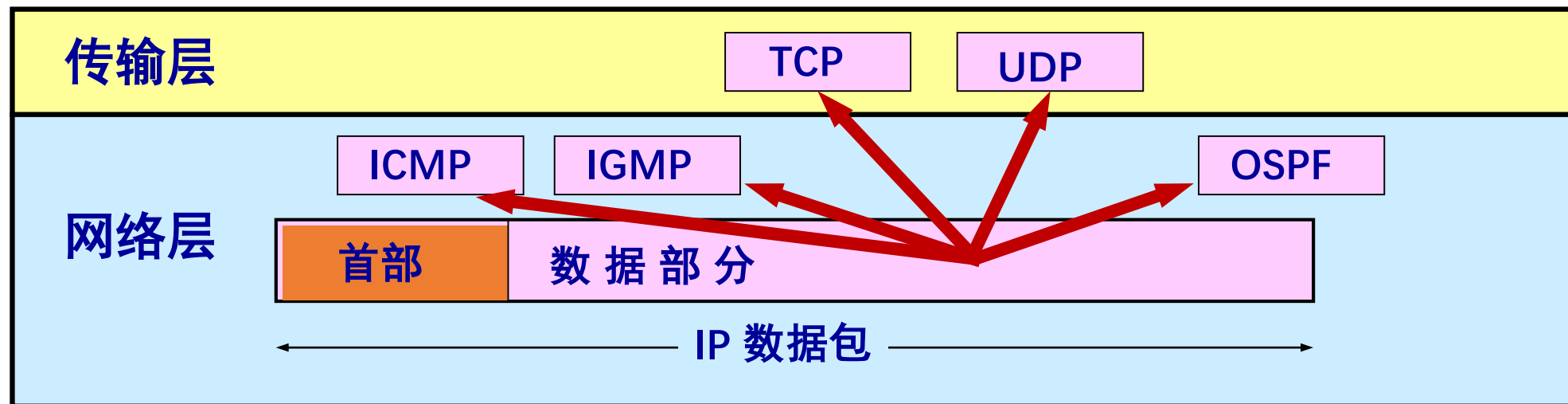
- 生存时间——占8位，记为 TTL (Time To Live)
- 指示数据包在网络中可通过的路由器数的最大值
- 初值由源主机设置，通过路由器转发后，TTL减1
- TTL值为0时，丢弃分组并发送ICMP报文通知源主机

## 2. 相关知识—IP数据包的结构



- 协议——占8位，指出此数据包携带的数据使用何种协议
- IP数据包可以封装多种协议 PDU（协议数据单元）

## 2. 相关知识—IP数据包的结构



字段值	协议名称	字段值	协议名称
1	ICMP(Internet Control Message Protocol)	17	UDP(User Datagram Protocol)
2	IGMP(Internet Group Management Protocol)	41	IPv6
6	TCP(Transmission Control Protocol)	46	RSVP(Resource Reservation Protocol)
8	EGP(Exterior Gateway Protocol)	89	OSPF(Open Shortest Path First)

参考: <http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>

## 2. 相关知识—IP数据包的结构



- 首部检验和——占16 位，只检验数据报的首部；不检验数据部分
- 不采用 CRC 检验码而采用简单的计算方法

## 2. 相关知识—IP数据包的结构

- 首部校验和的计算过程

- 将IP分组头看成是**16位**字组成的二进制比特序列
- 将**校验和**字段置0
- 对16位字进行求和运算，如果**最高位出现进位**，则将**进位**加到结果的最低位（**Ones' Complement Addition**）
- 将最终求和**结果取反**，得校验和

## 2. 相关知识—IP数据包的结构

### ● 发送端计算校验和

4, 5, and 0	→	01000101	00000000
28	→	00000000	00011100
1	→	00000000	00000001
0 and 0	→	00000000	00000000
4 and 17	→	00000100	00010001
0	→	00000000	00000000
10.12	→	00001010	00001100
14.5	→	00001110	00000101
12.6	→	00001100	00000110
7.9	→	00000111	00001001
Sum	→	01110100	01001110
Checksum	→	10001011	10110001

4	5	0	28	
1			0	0
4	17		0	
10.12.14.5				
12.6.7.9				

各位取反后得到校验和

Substitute for 0

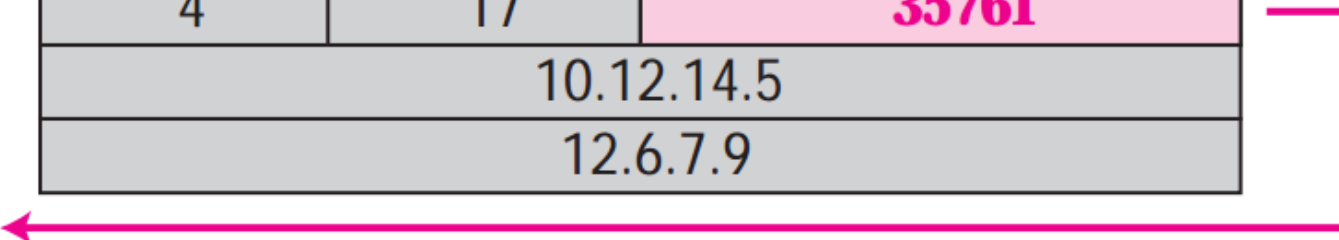


## 2. 相关知识—IP数据包的结构

### ● 接收端计算校验和

4, 5, and 0	→	01000101	00000000
28	→	00000000	00011100
1	→	00000000	00000001
0 and 0	→	00000000	00000000
4 and 17	→	00000100	00010001
Checksum	→	10001011	10110001
10.12	→	00001010	00001100
14.5	→	00001110	00000101
12.6	→	00001100	00000110
7.9	→	00000111	00001001
Sum	→	1111 1111	1111 1111
Checksum	→	0000 0000	0000 0000

4	5	0	28	
1			0	0
4	17		35761	
10.12.14.5				
12.6.7.9				





## 2. 相关知识—IP数据包的结构

- 二进制位相加

4, 5和0	01000101 00000000
28	00000000 00011100
	<hr/>
	01000101 00011100
1	00000000 00000001
	<hr/>
	01000101 00011101
0和0	00000000 00000000
	<hr/>
	01000101 00011101
4和17	00000100 00010001
	<hr/>
	01001001 00101110

## 2. 相关知识—IP数据包的结构

- 二进制位相加

01001001 00101110	
00000000 00000000	0
<hr/>	
01001001 00101110	
00001010 00001100	10.12
<hr/>	
01010011 00111010	
00001110 00000101	14.5
<hr/>	
01100001 00111111	
00001100 00000110	12.6
<hr/>	
01101101 01000101	
00000111 00001001	7.9
<hr/>	
01110100 01001110	和

## 2. 相关知识—IP数据包的结构

- 二进制位相加，有进位的例子

		0001 0110	
		+ 1111 1111	
		<hr/>	
端回进位 (end-around carry)	→	1 0001 0101	
		+ 0000 0001	
		<hr/>	
		0001 0110	← 进位加在最低位

## 2. 相关知识—IP数据包的结构



源地址和目的地址都各占 4 字节

## 2. 相关知识—IP数据包的结构

- 可选字段（0~40字节）
  - 很少使用
  - 选项码：确定选项的具体功能
  - 长度：选项数据的大小
  - 选项数据：根据具体功能设定
  - 如果出现头部长度的不是4字节整数倍的情况，则需要填充0来凑齐

### 3. 例题分析—设计要求

- 编写程序来捕获网络中传输的IPv4数据包
- 将得到的解析结果显示出来
- 只解析头部中除选项外的各字段值
- 不需要解析出具体的服务类型和协议类型

### 3. 例题分析—设计要求

- 具体要求

- 要求程序为命令行程序。例如，可执行文件名为PackParse.exe，则程序的命令行格式为：

**PackParse packet\_sum**

其中， packet\_sum为捕获IPv4数据包的数量

- 要求将部分字段内容显示在控制台上，具体格式为：

```
-----  
版本： xx  
头部长度： xx  
服务类型： xx, xx  
总长度： xx  
标识符： xx  
标志位： xx,DF,MF  
片偏移： xx  
生存周期： xx  
协议： xx  
头部校验和： xx  
源IP地址： xx.xx.xx.xx  
目的IP地址： xx.xx.xx.xx  
-----
```

### 3. 例题分析—设计要求

- 具体要求

- 有良好的编程规范与注释。编程所使用的操作系统、语言和编译环境不限，但是在提交的说明文档中需要加以注明
- 撰写说明文档，包括程序的开发思路、工作流程、关键问题、解决思路以及进一步的改进等内容



### 3. 例题分析—关键问题

- 创建原始套接字

- 为了通过网卡来截获传输中的IPv4数据包，需对套接字进行编程

- 流式套接字

- 数据包套接字



只能响应  
与本地网卡硬件地址匹配  
或者广播的数据包

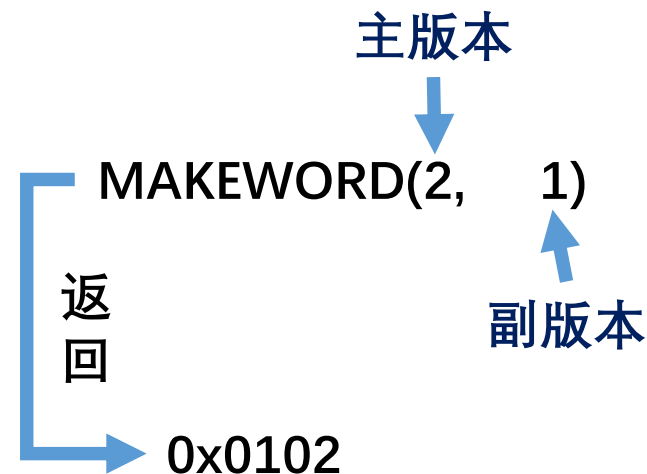
- 原始套接字：可接收与本地网卡的硬件地址不匹配的数据包 ★

### 3. 例题分析—关键问题

- 创建原始套接字

```
WSADATA WSAData;  
//初始化Winsock, 设置版本, 成功返回0  
if (WSAStartup(MAKEWORD(2,2), &WSAData)!=0)  
    ...  
//创建原始Socket  
SOCKET sock;  
sock = socket(AF_INET, SOCK_RAW, IPPROTO_IP);  
if ((sock==INVALID_SOCKET)  
    ...
```

MAKEWORD宏函数可以构建WORD型版本信息

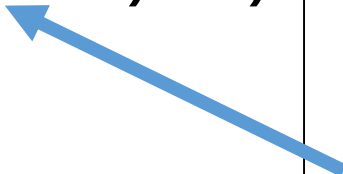


### 3. 例题分析—关键问题

- 创建原始套接字

```
WSADATA WSAData;  
//初始化Winsock, 设置版本, 成功返回0  
if (WSAStartup(MAKEWORD(2,2), &WSAData)!=0)  
    ...  
//创建原始Socket  
SOCKET sock;  
sock = socket(AF_INET, SOCK_RAW, IPPROTO_IP);  
if ((sock==INVALID_SOCKET)  
    ...
```

WSAStartup()调用结束后  
WSAData中填充已初始化的  
库信息

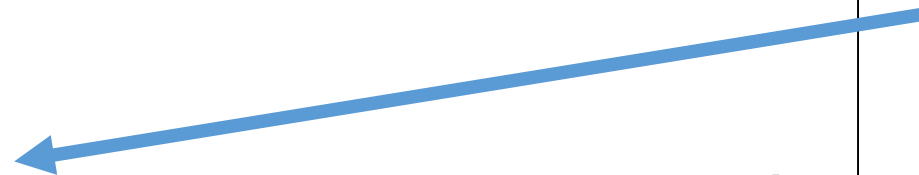


### 3. 例题分析—关键问题

- 创建原始套接字

```
WSADATA WSAData;  
//初始化Winsock, 设置版本, 成功返回0  
if (WSAStartup(MAKEWORD(2,2), &WSAData)!=0)  
    ...  
//创建原始Socket  
SOCKET sock;  
sock = socket(AF_INET, SOCK_RAW, IPPROTO_IP);  
if ((sock==INVALID_SOCKET)  
    ...
```

地址族	含义
AF_INET	IPv4网络协议中使用的地址族
AF_INET6	IPv6网络协议中使用的地址族



### 3. 例题分析—关键问题

- 创建原始套接字

```
WSADATA WSAData;  
//初始化Winsock, 设置版本, 成功返回0  
if (WSAStartup(MAKEWORD(2,2), &WSAData)!=0)  
    ...  
//创建原始Socket  
SOCKET sock;  
sock = socket(AF_INET, SOCK_RAW, IPPROTO_IP);  
if ((sock==INVALID_SOCKET)  
    ...
```

套接字类型	含义
SOCK_STREAM	流式套接字
SOCK_DGRAM	数据报套接字
SOCK_RAW	原始套接字

通信协议: IP

### 3. 例题分析—关键问题

- 创建原始套接字

```
WSADATA WSAData;  
//初始化Winsock, 设置版本, 成功返回0  
if (WSAStartup(MAKEWORD(2,2), &WSAData)!=0)  
    ...  
  
//创建原始Socket  
SOCKET sock;  
sock = socket(AF_INET, SOCK_RAW, IPPROTO_IP);  
if (sock==INVALID_SOCKET)  
    ...
```

- socket()函数执行成功返回套接字句柄(描述符)
- 失败返回INVALID\_SOCKET
- 套接字句柄是一个整型值
- 考虑以后扩展性, 使用SOCKET类型进行封装
- 句柄可以看成对象的别名

### 3. 例题分析—关键问题

- 初始化Socket结构

- 为了设置对IPv4头部的操作模式，需要调用setsockopt函数

```
BOOL flag = true;  
setsockopt(sock, IPPROTO_IP, IP_HDRINCL, (char *) &flag, sizeof(flag));
```

套接字句柄

参数层次：IP协议

需要设置的参数：IP\_HDRINCL

参数值：true

表示用户自己处理IPv4头部

参数值占用的  
字节数

### 3. 例题分析—关键问题

- 初始化Socket结构

准备Socket  
地址信息

```
char hostName[128];
gethostname(hostname, 100); //获得主机名
hostent * pHostIP;
//获得本地IP地址(网络字节序)
pHostIP = gethostbyname(hostName);
sockaddr_in addr_in;
addr_in.sin_family = AF_INET;
addr_in.sin_port = htons(6000); //端口转成网络字节序
addr_in.sin_addr = *(in_addr *) pHostIP->h_addr_list[0];
bind(sock, (PSOCKADDR) & addr_in, sizeof(addr_in));
```

Socket绑定  
本地网卡



### 3. 例题分析—关键问题

- **接收IPv4数据包**

- 通常网卡不能接收目的地址不是自己的数据包
- 需要调用WSAIoctl函数将网卡设置为混杂模式(promiscuous mode)
- 当接收的数据包中的协议类型与原始套接字匹配，接收的数据包就被复制到套接字的缓冲区中

### 3. 例题分析—关键问题

- 接收IPv4数据包

先把网卡设置为混杂模式

```
#define IO_RCVALL _WSAIOW(IOC_VENDOR, 1)
DWORD dwBufferLen[10];
DWORD dwBufferInLen = 1;
DWORD dwBytesReturned = 0;
WSAIoctl(sock, IO_RCVALL, &dwBufferInLen, sizeof(dwBufferInLen),
&dwBufferLen, sizeof(dwBufferLen), &dwBytesReturned, NULL, NULL);
```

### 3. 例题分析—关键问题

- 接收IPv4数据包

#### 再捕获IP数据包

```
#define BUFFER_SIZE 65535
char buffer[BUFFER_SIZE]; //缓冲区
while(数据包捕获未结束)
{
    recv(sock, buffer, BUFFER_SIZE, 0);
    ...
}
```

### 3. 例题分析—关键问题

- 定义IPv4头部数据结构
  - 在对IPv4头部各字段进行解析之前，首先需要构造相对应的数据结构

```
typedef struct IP_HEAD
{
    ...
} ip_head;
```

### 3. 例题分析—关键问题

- 定义IPv4头部数据结构

```
union
{
    unsigned char Version;//版本(前4位)
    unsigned char HeadLen;//头部长度(后4位)
};
unsigned char ServiceType; //服务类型(1B)
unsigned short TotalLen;//总长度(2B)
unsigned short Identifier;//标识符(2B)
```

### 3. 例题分析—关键问题

- 定义IPv4头部数据结构

```
union
{
    unsigned short Flags; //标志位(前3位)
    unsigned short FragOffset; //片偏移(后13位)
};
unsigned char TimeToLive; //生存周期(1B)
unsigned char Protocol; //协议(1B)
unsigned short HeadChecksum; //头部校验和(2B)
```

### 3. 例题分析—关键问题

- 定义IPv4头部数据结构

```
unsigned int SourceAddr; //源IP地址(4B)  
unsigned int DestinAddr; //目的IP地址(4B)  
unsigned char Options; // 选项
```

### 3. 例题分析—关键问题

- 解析IPv4头部字段
  - 通过指针将缓冲区中的内容强制转化为ip\_head结构，然后取出字段值
  - **ip\_head ip =\* (ip\_head \*)buffer;**



### 3. 例题分析—关键问题

- 解析IPv4头部字段

字段	表达式
版本	<code>ip.Version&gt;&gt;4</code>
头部长度的	<code>ip.HeadLen&amp;0x0f</code>
服务类型	<code>ip.ServiceType&gt;&gt;5, (ip.ServiceType&gt;&gt;1) &amp; 0x0f</code>
总长度	<code>ip.TotalLen</code>
标识符	<code>ip.Identifier</code>
标志位	<code>ip.Flags&gt;&gt;15 &amp; 0x01, ip.Flags&gt;&gt;14 &amp; 0x01, ip.Flags&gt;&gt;13 &amp; 0x01</code>

### 3. 例题分析—关键问题

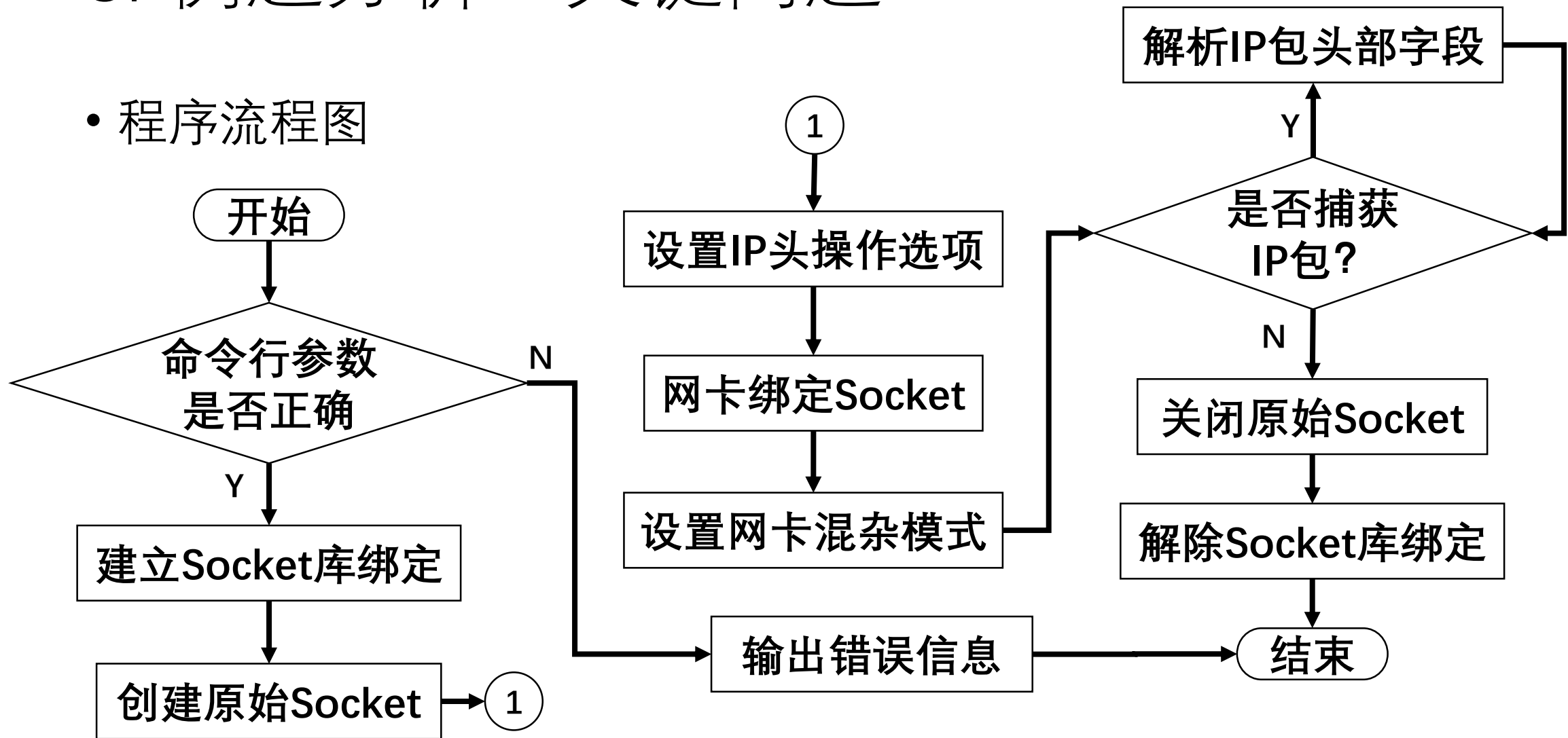
- 解析IPv4头部字段

字段	表达式
片偏移	<code>ip.FragOffset &amp; 0x1fff</code>
生命周期	<code>ip.TimeToLive</code>
协议	<code>ip.Protocol</code>
头部校验和	<code>ip.HeadChecksum</code>
源地址	<code>inet_ntoa(*(in_addr *) &amp;ip.SourceAddr)</code>
目的地址	<code>inet_ntoa(*(in_addr *) &amp;ip.DestinAddr)</code>

`inet_ntoa`将一个32位网络字节序的二进制IP地址转换成相应的点分十进制的IP地址

### 3. 例题分析—关键问题

- 程序流程图



# 程序演示

开始解析IP包:

-----  
版本: 4

头部长度的: 20

服务器类型: Priority 0, Service 0

总长度: 10240

标识符: 5751

标志位: 0, DF=0, MF=0

片偏移: 64

生存周期: 128

协议: Protocol 6

头部校验和: 32545

源IP地址: 192.168.1.102

目的IP地址: 180.163.235.136  
-----

# 本章小结

- 设计目的
  - IP包是网络层数据传输的基本单位
  - 了解头部中各个字段的含义与用途，理解网络协议的工作原理
- 相关知识
  - 网络层基本概念、包的结构
- 例题分析
  - 创建流式套接字、IPv4头操作模式设置、网络混杂模式，解析IPv4头