

# 练识课堂 -- Go语言资深工程师培训课程

## 1 协议的概念

### 1.1 什么是协议

从应用的角度出发，协议可理解为“规则”，是数据传输和数据的解释的规则。

假设，A、B双方欲传输文件。规定：

- 第一次，传输文件名，接收方接收到文件名，应答OK给传输方；
- 第二次，发送文件的尺寸，接收方接收到该数据再次应答一个OK；
- 第三次，传输文件内容。同样，接收方接收数据完成后应答OK表示文件内容接收成功。

由此，无论A、B之间传递何种文件，都是通过三次数据传输来完成。A、B之间形成了一个最简单的数据传输规则。双方都按此规则发送、接收数据。A、B之间达成的这个相互遵守的规则即为协议。

这种仅在A、B之间被遵守的协议称之为**原始协议**。当此协议被更多的人采用，不断的增加、改进、维护、完善。最终形成一个稳定的、完整的文件传输协议，被广泛应用于各种文件传输过程中。该协议就成为一个**标准协议**。最早的ftp协议就是由此衍生而来。

TCP协议注重数据的传输。http协议着重于数据的解释。

### 1.2 典型协议

传输层 常见协议有TCP/UDP协议。

应用层 常见的协议有HTTP协议，FTP协议。

网络层 常见协议有IP协议、ICMP协议、IGMP协议。

网络接口层 常见协议有ARP协议、RARP协议。

TCP传输控制协议（Transmission Control Protocol）是一种面向连接的、可靠的、基于字节流的传输层通信协议。

UDP用户数据报协议（User Datagram Protocol）是OSI参考模型中一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务。

HTTP超文本传输协议（Hyper Text Transfer Protocol）是互联网上应用最为广泛的一种网络协议。

FTP文件传输协议（File Transfer Protocol）

IP协议是因特网互联协议（Internet Protocol）

ICMP协议是Internet控制报文协议（Internet Control Message Protocol）它是TCP/IP协议族的一个子协议，用于在IP主机、路由器之间传递控制消息。

IGMP协议是 Internet 组管理协议（Internet Group Management Protocol），是因特网协议家族中的一个组播协议。该协议运行在主机和组播路由器之间。

ARP协议是正向地址解析协议（Address Resolution Protocol），通过已知的IP，寻找对应主机的MAC地址。

RARP是反向地址转换协议，通过MAC地址确定IP地址。

## 1.3 分层模型

- **物理层：**

主要定义物理设备标准，如网线的接口类型、光纤的接口类型、各种传输介质的传输速率等。它的主要作用是传输比特流（就是由1、0转化为电流强弱来进行传输，到达目的地后再转化为1、0，也就是我们常说的数模转换与模数转换）。这一层的数据叫做比特。

- **数据链路层：**

定义了如何让格式化数据以帧为单位进行传输，以及如何让控制对物理介质的访问。这一层通常还提供错误检测和纠正，以确保数据的可靠传输。如：串口通信中使用到的115200、8、N、1

- **网络层：**

在位于不同地理位置的网络中的两个主机系统之间提供连接和路径选择。Internet的发展使得从世界各站点访问信息的用户数大大增加，而网络层正是管理这种连接的层。

- **传输层：**

定义了一些传输数据的协议和端口号（WWW端口80等），如：TCP（传输控制协议，传输效率低，可靠性强，用于传输可靠性要求高，数据量大的数据），UDP（用户数据报协议，与TCP特性恰恰相反，用于传输可靠性要求不高，数据量小的数据，如QQ聊天数据就是通过这种方式传输的）。主要是将从下层接收的数据进行分段和传输，到达目的地后再进行重组。常常把这一层数据叫做段。

- **会话层：**

通过传输层(端口号：传输端口与接收端口)建立数据传输的通路。主要在你的系统之间发起会话或者接受会话请求（设备之间需要互相认识可以是IP也可以是MAC或者是主机名）。

- **表示层：**

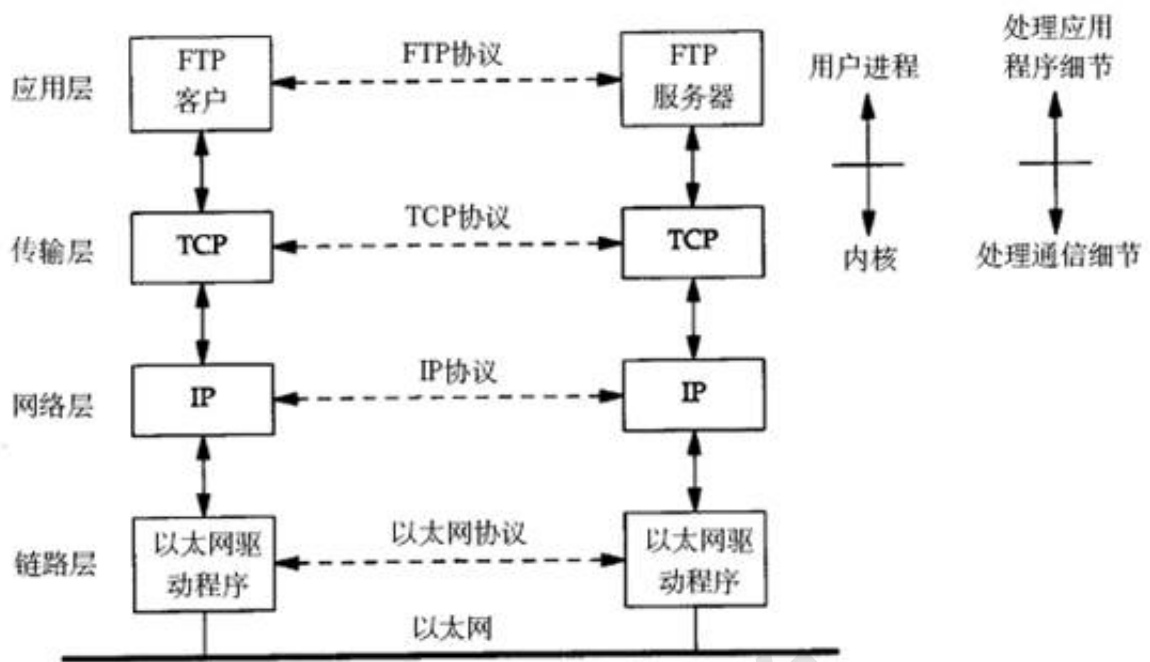
可确保一个系统的应用层所发送的信息可以被另一个系统的应用层读取。例如，PC程序与另一台计算机进行通信，其中一台计算机使用扩展二一十进制交换码(EBCDIC)，而另一台则使用美国信息交换标准码（ASCII）来表示相同的字符。如有必要，表示层会通过使用一种通格式来实现多种数据格式之间的转换。

- **应用层：**

是最靠近用户的OSI层。这一层为用户的应用程序（例如电子邮件、文件传输和终端仿真）提供网络服务。

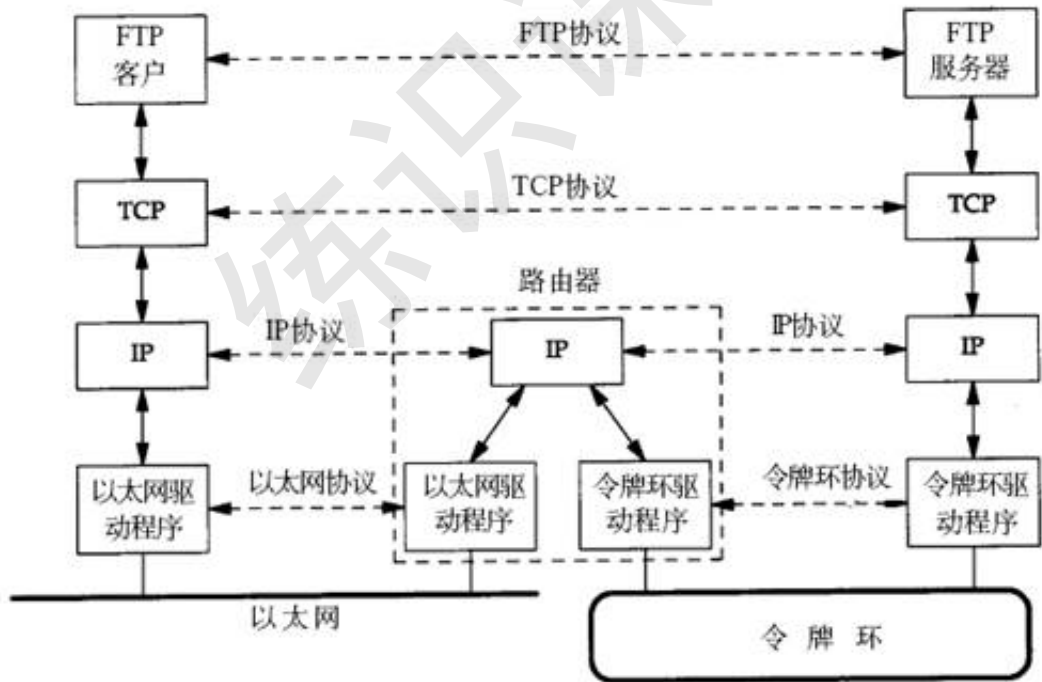
## 2 通信过程

两台计算机通过TCP/IP协议通讯的过程如下所示：



### TCP/IP通信过程

上图对应两台计算机在同一网段中的情况，如果两台计算机在不同的网段中，那么数据从一台计算机到另一台计算机传输过程中要经过一个或多个路由器，如下图所示：



### 跨路由通信

链路层有以太网、令牌环网等标准，链路层负责网卡设备的驱动、帧同步（即从网线上检测到什么信号算作新帧的开始）、冲突检测（如果检测到冲突就自动重发）、数据差错校验等工作。交换机是工作在链路层的网络设备，可以在不同的链路层网络之间转发数据帧（比如十兆以太网和百兆以太网之间、以太网和令牌环网之间），由于不同链路层的帧格式不同，交换机要将进来的数据包拆掉链路层首部重新封装之后再转发。

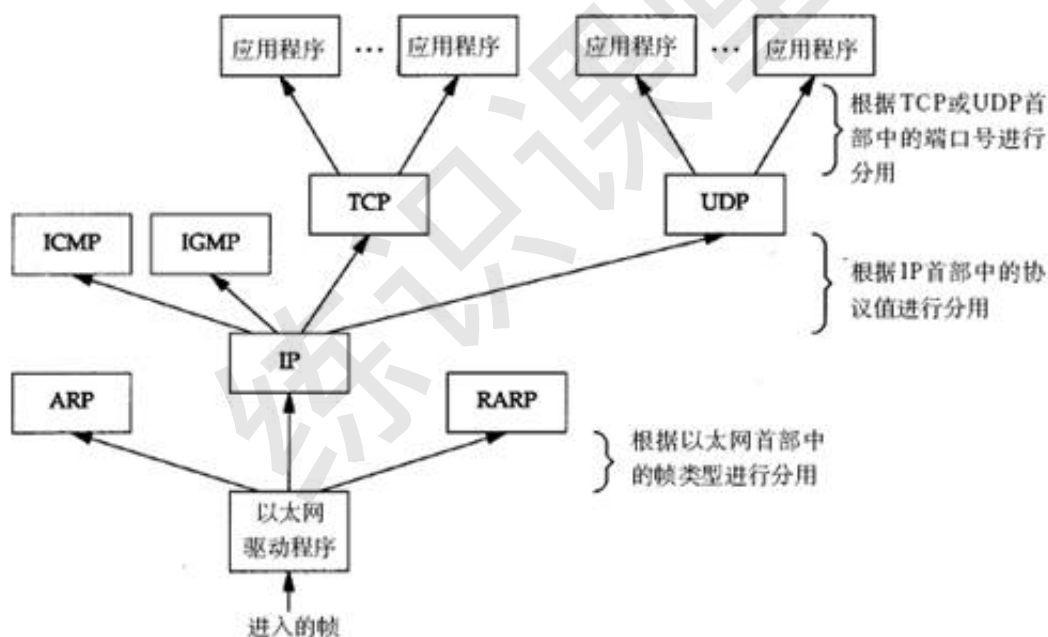
网络层的IP协议是构成Internet的基础。Internet上的主机通过IP地址来标识，Internet上有大量路由器负责根据IP地址选择合适的路径转发数据包，数据包从Internet上的源主机到目的主机往往要经过十多个路由器。路由器是工作在第三层的网络设备，同时兼有交换机的功能，可以在不同的链路层接口之间转发数据包，因此路由器需要将进来的数据包拆掉网络层和链路层两层首部并重新封装。IP协议不保证传输的可靠性，数据包在传输过程中可能丢失，可靠性可以在上层协议或应用程序中提供支持。

网络层负责点到点（ptop, point-to-point）的传输（这里的“点”指主机或路由器），而传输层负责端到端（etoe, end-to-end）的传输（这里的“端”指源主机和目的主机）。传输层可选择TCP或UDP协议。

TCP是一种面向连接的、可靠的协议，有点像打电话，双方拿起电话互通身份之后就建立了连接，然后说话就行了，这边说的话那边保证听得到，并且是按说话的顺序听到的，说完话挂机断开连接。也就是说TCP传输的双方需要首先建立连接，之后由TCP协议保证数据收发的可靠性，丢失的数据包自动重发，上层应用程序收到的总是可靠的数据流，通讯之后关闭连接。

UDP是无连接的传输协议，不保证可靠性，有点像寄信，信写好放到邮筒里，既不能保证信件在邮递过程中不会丢失，也不能保证信件寄送顺序。使用UDP协议的应用程序需要自己完成丢包重发、消息排序等工作。

目的主机收到数据包后，如何经过各层协议栈最后到达应用程序呢？其过程如下图所示：



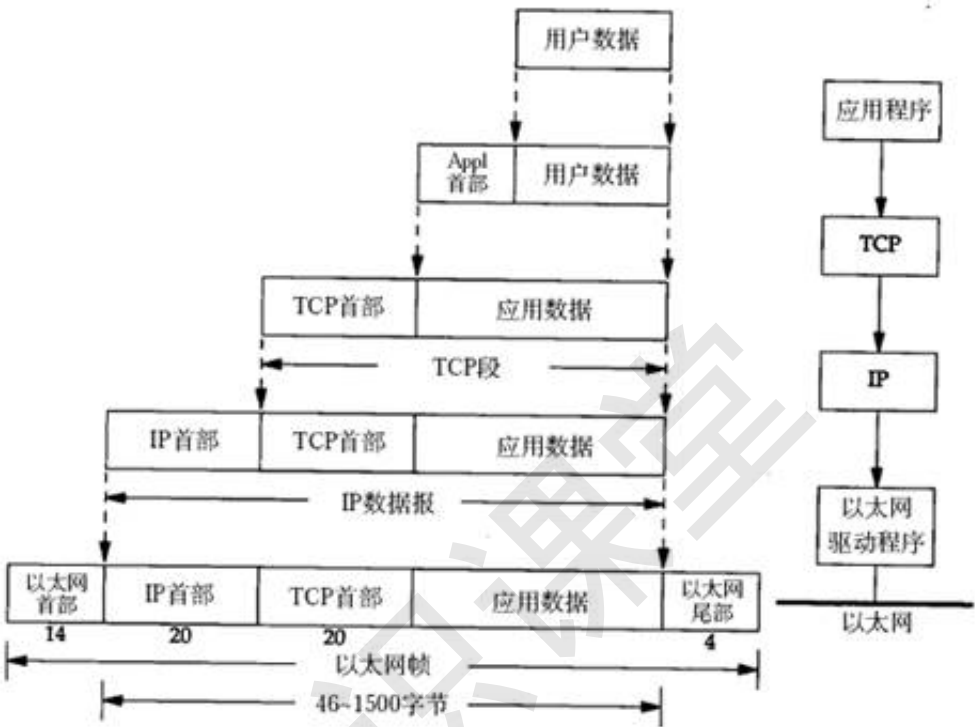
以太网驱动程序首先根据以太网首部中的“上层协议”字段确定该数据帧的有效载荷（payload，指除去协议首部之外实际传输的数据）是IP、ARP还是RARP协议的数据报，然后交给相应的协议处理。假如是IP数据报，IP协议再根据IP首部中的“上层协议”字段确定该数据报的有效载荷是TCP、UDP、ICMP还是IGMP，然后交给相应的协议处理。假如是TCP段或UDP段，TCP或UDP协议再根据TCP首部或UDP首部的“端口号”字段确定应该将应用层数据交给哪个用户进程。IP地址是标识网络中不同主机的地址，而端口号就是同一台主机上标识不同进程的地址，IP地址和端口号合起来标识网络中唯一的进程。

虽然IP、ARP和RARP数据报都需要以太网驱动程序来封装成帧，但是从功能上划分，ARP和RARP属于链路层，IP属于网络层。虽然ICMP、IGMP、TCP、UDP的数据都需要IP协议来封装成数据报，但是从功能上划分，ICMP、IGMP与IP同属于网络层，TCP和UDP属于传输层。

## 3 协议格式

### 3.1 数据包封装

传输层及其以下的机制由内核提供，应用层由用户进程提供（后面将介绍如何使用socket API编写应用程序），应用程序对通讯数据的含义进行解释，而传输层及其以下处理通讯的细节，将数据从一台计算机通过一定的路径发送到另一台计算机。应用层数据通过协议栈发到网络上时，每层协议都要加上一个数据首部（header），称为封装（Encapsulation），如下图所示：

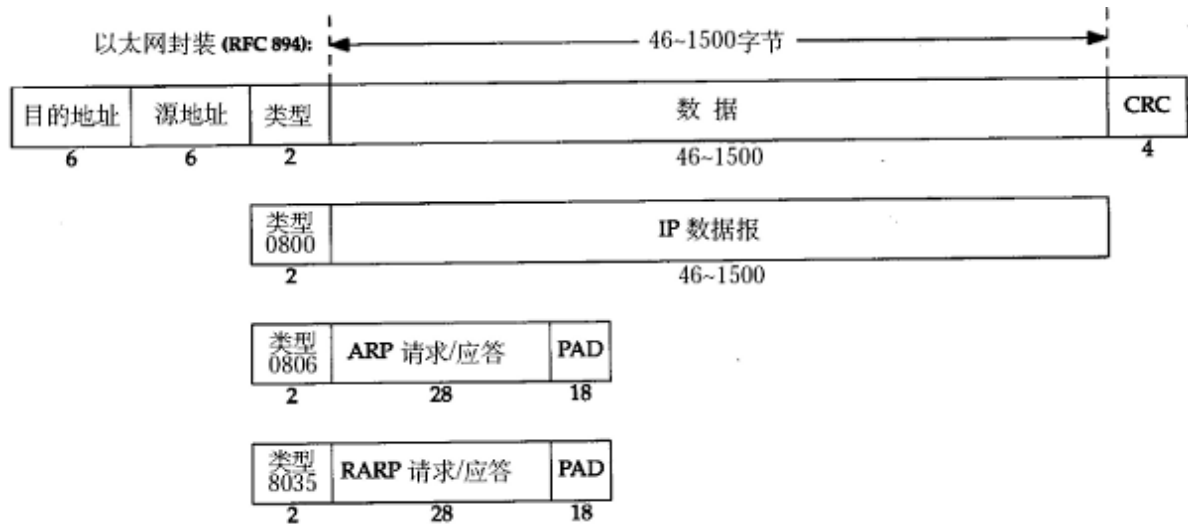


TCP/IP数据包封装

不同的协议层对数据包有不同的称谓，在传输层叫做段（segment），在网络层叫做数据报（datagram），在链路层叫做帧（frame）。数据封装成帧后发到传输介质上，到达目的主机后每层协议再剥掉相应的首部，最后将应用层数据交给应用程序处理。

### 3.2 以太网帧格式

以太网的帧格式如下所示：



### 以太网帧格式

其中的源地址和目的地址是指网卡的硬件地址（也叫MAC地址），长度是48位，是在网卡出厂时固化的。可在shell中使用ifconfig命令查看，“HWaddr 00:15:F2:14:9E:3F”部分就是硬件地址。协议字段有三种值，分别对应IP、ARP、RARP。帧尾是CRC校验码。

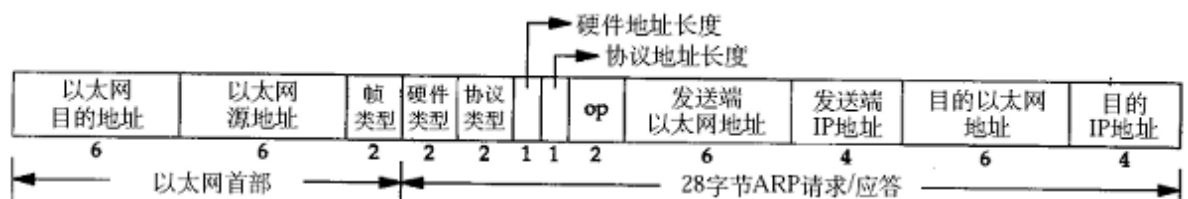
以太网帧中的数据长度规定最小46字节，最大1500字节，ARP和RARP数据包的长度不够46字节，要在后面补填充位。最大值1500称为以太网的最大传输单元（MTU），不同的网络类型有不同的MTU，如果一个数据包从以太网路由到拨号链路上，数据包长度大于拨号链路的MTU，则需要对数据包进行分片（fragmentation）。ifconfig命令输出中也有“MTU:1500”。注意，MTU这个概念指数据帧中有效载荷的最大长度，不包括帧头长度。

## 3.3 ARP数据报格式

在网络通讯时，源主机的应用程序知道目的主机的IP地址和端口号，却不知道目的主机的硬件地址，而数据包首先是被网卡接收到再去处理上层协议的，如果接收到的数据包的硬件地址与本机不符，则直接丢弃。因此在通讯前必须获得目的主机的硬件地址。ARP协议就起到这个作用。源主机发出ARP请求，询问“IP地址是192.168.0.1的主机的硬件地址是多少”，并将这个请求广播到本地网段（以太网帧首部的硬件地址填FF:FF:FF:FF:FF:FF表示广播），目的主机接收到广播的ARP请求，发现其中的IP地址与本机相符，则发送一个ARP应答数据包给源主机，将自己的硬件地址填写在应答包中。

每台主机都维护一个ARP缓存表，可以用arp -a命令查看。缓存表中的表项有过期时间（一般为20分钟），如果20分钟内没有再次使用某个表项，则该表项失效，下次还要发ARP请求来获得目的主机的硬件地址。想一想，为什么表项要有过期时间而不是一直有效？

ARP数据报的格式如下所示：



### ARP数据报格式



源MAC地址、目的MAC地址在以太网首部和ARP请求中各出现一次，对于链路层为以太网的情况是多余的，但如果链路层是其它类型的网络则有可能是必要的。硬件类型指链路层网络类型，1为以太网，协议类型指要转换的地址类型，0x0800为IP地址，后面两个地址长度对于以太网地址和IP地址分别为6和4（字节），op字段为1表示ARP请求，op字段为2表示ARP应答。

看一个具体的例子。

请求帧如下（为了清晰在每行的前面加了字节计数，每行16个字节）：

以太网首部（14字节）

```
0000: ff ff ff ff ff ff 00 05 5d 61 58 a8 08 06
```

ARP帧（28字节）

```
0000: 00 01
```

```
0010: 08 00 06 04 00 01 00 05 5d 61 58 a8 c0 a8 00 37
```

```
0020: 00 00 00 00 00 00 c0 a8 00 02
```

填充位（18字节）

```
0020: 00 77 31 d2 50 10
```

```
0030: fd 78 41 d3 00 00 00 00 00 00 00 00
```

以太网首部：目的主机采用广播地址，源主机的MAC地址是00:05:5d:61:58:a8，上层协议类型0x0806表示ARP。

ARP帧：硬件类型0x0001表示以太网，协议类型0x0800表示IP协议，硬件地址（MAC地址）长度为6，协议地址（IP地址）长度为4，op为0x0001表示请求目的主机的MAC地址，源主机MAC地址为00:05:5d:61:58:a8，源主机IP地址为c0 a8 00 37（192.168.0.55），目的主机MAC地址全0待填写，目的主机IP地址为c0 a8 00 02（192.168.0.2）。

由于以太网规定最小数据长度为46字节，ARP帧长度只有28字节，因此有18字节填充位，填充位的内容没有定义，与具体实现相关。

应答帧如下：

以太网首部

```
0000: 00 05 5d 61 58 a8 00 05 5d a1 b8 40 08 06
```

ARP帧

0000: 00 01

0010: 08 00 06 04 00 02 00 05 5d a1 b8 40 c0 a8 00 02

0020: 00 05 5d 61 58 a8 c0 a8 00 37

填充位

0020: 00 77 31 d2 50 10

0030: fd 78 41 d3 00 00 00 00 00 00 00

以太网首部：目的主机的MAC地址是00:05:5d:61:58:a8，源主机的MAC地址是00:05:5d:a1:b8:40，上层协议类型0x0806表示ARP。

ARP帧：硬件类型0x0001表示以太网，协议类型0x0800表示IP协议，硬件地址（MAC地址）长度为6，协议地址（IP地址）长度为4，op为0x0002表示应答，源主机MAC地址为00:05:5d:a1:b8:40，源主机IP地址为c0 a8 00 02（192.168.0.2），目的主机MAC地址为00:05:5d:61:58:a8，目的主机IP地址为c0 a8 00 37（192.168.0.55）。

### 3.4 IP段格式



#### IP数据报格式

IP数据报的首部长度和数据长度都是可变长的，但总是4字节的整数倍。

对于IPv4，4位版本字段是4。4位首部长度的数值是以4字节为单位的，最小值为5，也就是说首部长度最小是4x5=20字节，也就是不带任何选项的IP首部，4位能表示的最大值是15，也就是说首部长度最大是60字节。

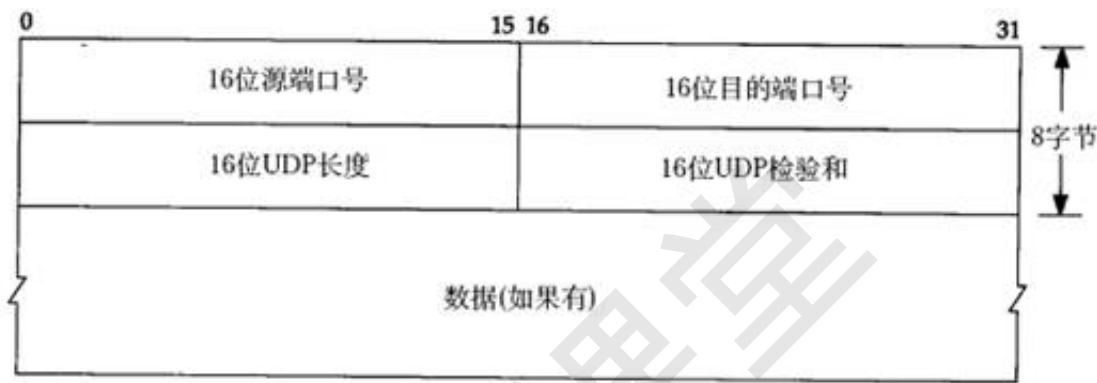


8位TOS字段有3个位用来指定IP数据报的优先级（目前已经废弃不用），还有4个位表示可选的服务类型（最小延迟、最大吐量、最大可靠性、最小成本），还有一个位总是0。总长度是整个数据报（包括IP首部和IP层payload）的字节数。每传一个IP数据报，16位的标识加1，可用于分片和重新组装数据报。3位标志和13位片偏移用于分片。

TTL（Time to live）是这样用的：源主机为数据包设定一个生存时间，比如64，每过一个路由器就把该值减1，如果减到0就表示路由已经太长了仍然找不到目的主机的网络，就丢弃该包，因此这个生存时间的单位不是秒，而是跳（hop）。

协议字段指示上层协议是TCP、UDP、ICMP还是IGMP。然后是校验和，只校验IP首部，数据的校验由更高层协议负责。IPv4的IP地址长度为32位。

### 3.5 UDP数据报格式



UDP数据段

下面分析一帧基于UDP的TFTP协议帧。

以太网首部

```
0000: 00 05 5d 67 d0 b1 00 05 5d 61 58 a8 08 00
```

IP首部

```
0000: 45 00

0010: 00 53 93 25 00 00 80 11 25 ec c0 a8 00 37 c0 a8

0020: 00 01
```

UDP首部

```
0020: 05 d4 00 45 00 3f ac 40
```

TFTP协议

```
0020: 00 01 'c':'\''q'

0030: 'w' 'e' 'r' 'q' '.' 'q' 'w' 'e' 00 'n' 'e' 't' 'a' 's' 'c' 'i'

0040: 'i' 00 'b' 'l' 'k' 's' 'i' 'z' 'e' 00 '5' '1' '2' 00 't' 'i'

0050: 'm' 'e' 'o' 'u' 't' 00 'l' '0' 00 't' 's' 'i' 'z' 'e' 00 '0'
```

**以太网首部：**源MAC地址是00:05:5d:61:58:a8，目的MAC地址是00:05:5d:67:d0:b1，上层协议类型0x0800表示IP。

**IP首部：**每一个字节0x45包含4位版本号和4位首部长度，版本号为4，即IPv4，首部长度为5，说明IP首部不带有选项字段。服务类型为0，没有使用服务。16位总长度字段（包括IP首部和IP层payload的长度）为0x0053，即83字节，加上以太网首部14字节可知整个帧长度是97字节。IP报标识是0x9325，标志字段和片偏移字段设置为0x0000，就是DF=0允许分片，MF=0此数据报没有更多分片，没有分片偏移。TTL是0x80，也就是128。上层协议0x11表示UDP协议。IP首部校验和为0x25ec，源主机IP是c0 a8 00 37（192.168.0.55），目的主机IP是c0 a8 00 01（192.168.0.1）。

**UDP首部：**源端口号0x05d4（1492）是客户端的端口号，目的端口号0x0045（69）是TFTP服务的well-known端口号。UDP报长度为0x003f，即63字节，包括UDP首部和UDP层payload的长度。UDP首部和UDP层payload的校验和为0xac40。

TFTP是基于文本的协议，各字段之间用字节0分隔，开头的00 01表示请求读取一个文件，接下来的各字段是：

```
c:\qwerq.qwe

netascii

blksize 512

timeout 10

tsize 0
```

一般的网络通信都是像TFTP协议这样，通信的双方分别是客户端和服务端，客户端主动发起请求（上面的例子就是客户端发起的请求帧），而服务端被动地等待、接收和应答请求。客户端的IP地址和端口号唯一标识了该主机上的TFTP客户端进程，服务端的IP地址和端口号唯一标识了该主机上的TFTP服务进程，由于客户端是主动发起请求的一方，它必须知道服务端的IP地址和TFTP服务进程的端口号，所以，一些常见的网络协议有默认的服务端端口，例如HTTP服务默认TCP协议的80端口，FTP服务默认TCP协议的21端口，TFTP服务默认UDP协议的69端口（如上例所示）。在使用客户端程序时，必须指定服务端的主机名或IP地址，如果不明确指定端口号则采用默认端口，请读者查阅ftp、tftp等程序的man page了解如何指定端口号。/etc/services中列出了所有well-known的服务端口和对应的传输层协议，这是由IANA（Internet Assigned Numbers Authority）规定的，其中有些服务既可以用TCP也可以用UDP，为了清晰，IANA规定这样的服务采用相同的TCP或UDP默认端口号，而另外一些TCP和UDP的相同端口号却对应不同的服务。

很多服务有well-known的端口号，然而客户端程序的端口号却不一定是well-known的，往往是每次运行客户端程序时由系统自动分配一个空闲的端口号，用完就释放掉，称为ephemeral的端口号，想想这是为什么？

前面提过，UDP协议不面向连接，也不保证传输的可靠性，例如：

发送端的UDP协议层只管把应用层传来的数据封装成段交给IP协议层就算完成任务了，如果因为网络故障该段无法发到对方，UDP协议层也不会给应用层返回任何错误信息。

接收端的UDP协议层只管把收到的数据根据端口号交给相应的应用程序就算完成任务了，如果发送端发来多个数据包并且在网络上经过不同的路由，到达接收端时顺序已经错乱了，UDP协议层也不保证按发送时的顺序交给应用层。

通常接收端的UDP协议层将收到的数据放在一个固定大小的缓冲区中等待应用程序来提取和处理，如果应用程序提取和处理的速度很慢，而发送端发送的速度很快，就会丢失数据包，UDP协议层并不报告这种错误。

因此，使用UDP协议的应用程序必须考虑到这些可能的问题并实现适当的解决方案，例如等待应答、超时重发、为数据包编号、流量控制等。一般使用UDP协议的应用程序实现都比较简单，只是发送一些对可靠性要求不高的消息，而不发送大量的数据。例如，基于UDP的TFTP协议一般只用于传送小文件（所以才叫trivial的ftp），而基于TCP的FTP协议适用于各种文件的传输。TCP协议又是如何用面向连接的服务来代替应用程序解决传输的可靠性问题呢。

### 3.6 TCP数据报格式



#### TCP数据段

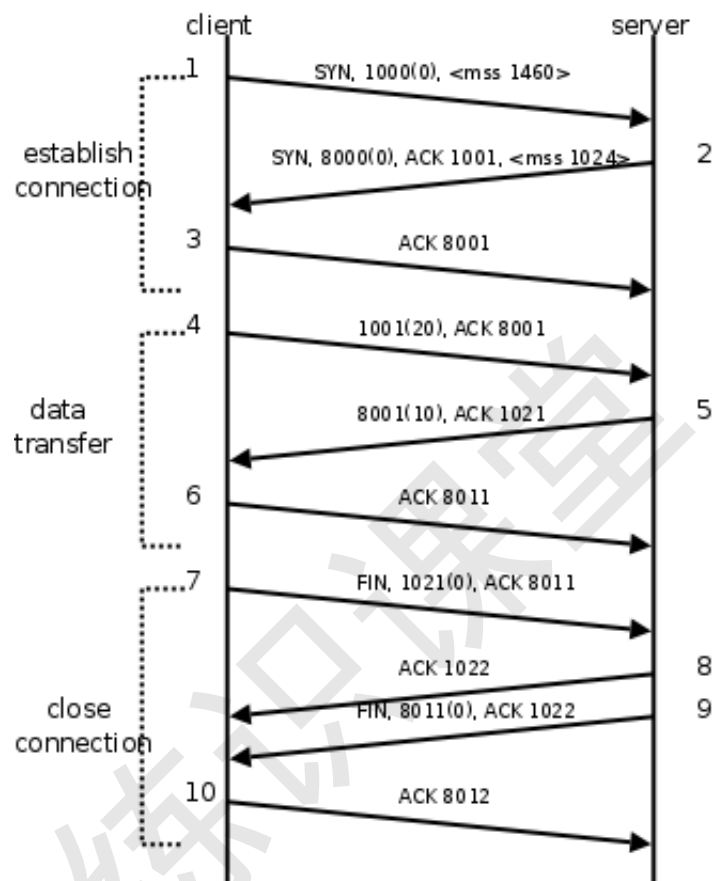
与UDP协议一样也有源端口号和目的端口号，通讯的双方由IP地址和端口号标识。32位序号、32位确认序号、窗口大小稍后详细解释。4位首部长度和IP协议头类似，表示TCP协议头的长度，以4字节为单位，因此TCP协议头最长可以是 $4 \times 15 = 60$ 字节，如果没有选项字段，TCP协议头最短20字节。URG、ACK、PSH、RST、SYN、FIN是六个控制位，本节稍后将解释SYN、ACK、FIN、RST四个位，其它位的解释从略。16位检验和将TCP协议头和数据都计算在内。紧急指针和各种选项

的解释从略。

## 4 TCP协议

### 4.1 TCP通信时序

下图是一次TCP通讯的时序图。TCP连接建立断开。包含大家熟知的三次握手和四次握手。



#### TCP通讯时序

在这个例子中，首先客户端主动发起连接、发送请求，然后服务器端响应请求，然后客户端主动关闭连接。两条竖线表示通讯的两端，从上到下表示时间的先后顺序，注意，数据从一端传到网络的另一端也需要时间，所以图中的箭头都是斜的。双方发送的段按时间顺序编号为1-10，各段中的主要信息在箭头上标出，例如段2的箭头上标着SYN, 8000(0), ACK1001,，表示该段中的SYN位置1，32位序号是8000，该段不携带有效载荷（数据字节数为0），ACK位置1，32位确认序号是1001，带有一个**mss**（Maximum Segment Size，**最大报文长度**）选项值为1024。

建立连接（三次握手）的过程：

- 客户端发送一个带SYN标志的TCP报文到服务器。这是三次握手过程中的段1。

客户端发出段1，SYN位表示连接请求。序号是1000，这个序号在网络通讯中用作临时的地址，每发一个数据字节，这个序号要加1，这样在接收端可以根据序号排出数据包的正确顺序，也可以发现丢包的情况，另外，规定SYN位和FIN位也要占一个序号，这次虽然没发数据，但是由于发了SYN位，因此下次再发送应该用序号1001。mss表示最大段尺寸，如果一个段太大，封装成帧后

超过了链路层的最大帧长度，就必须在IP层分片，为了避免这种情况，客户端声明自己的最大段尺寸，建议服务器端发来的段不要超过这个长度。

- 服务器端回应客户端，是三次握手中的第2个报文段，同时带ACK标志和SYN标志。它表示对刚才客户端SYN的回应；同时又发送SYN给客户端，询问客户端是否准备好进行数据通讯。

服务器发出段2，也带有SYN位，同时置ACK位表示确认，确认序号是1001，表示“我接收到序号1000及其以前所有的段，请你下次发送序号为1001的段”，也就是应答了客户端的连接请求，同时也给客户端发出一个连接请求，同时声明最大尺寸为1024。

- 客户必须再次回应服务器端一个ACK报文，这是报文段3。

客户端发出段3，对服务器的连接请求进行应答，确认序号是8001。在这个过程中，客户端和服务器分别给对方发了连接请求，也应答了对方的连接请求，其中服务器的请求和应答在一个段中发出，因此一共有三个段用于建立连接，称为“三方握手（three-way-handshake）”。在建立连接的同时，双方协商了一些信息，例如双方发送序号的初始值、最大段尺寸等。

在TCP通讯中，如果一方收到另一方发来的段，读出其中的目的端口号，发现本机并没有任何进程使用这个端口，就会应答一个包含RST位的段给另一方。例如，服务器并没有任何进程使用8080端口，我们却用telnet客户端去连接它，服务器收到客户端发来的SYN段就会应答一个RST段，客户端的telnet程序收到RST段后报告错误Connection refused：

```
$ telnet 192.168.0.200 8080

Trying 192.168.0.200...

telnet: Unable to connect to remote host: Connection refused
```

数据传输的过程：

- 客户端发出段4，包含从序号1001开始的20个字节数据。
- 服务器发出段5，确认序号为1021，对序号为1001-1020的数据表示确认收到，同时请求发送序号1021开始的数据，服务器在应答的同时也向客户端发送从序号8001开始的10个字节数据，这称为piggyback。
- 客户端发出段6，对服务器发来的序号为8001-8010的数据表示确认收到，请求发送序号8011开始的数据。

在数据传输过程中，ACK和确认序号是非常重要的，应用程序交给TCP协议发送的数据会暂存在TCP层的发送缓冲区中，发出数据包给对方之后，只有收到对方应答的ACK段才知道该数据包确实发到了对方，可以从发送缓冲区中释放掉了，如果因为网络故障丢失了数据包或者丢失了对方发回的ACK段，经过等待超时后TCP协议自动将发送缓冲区中的数据包重发。

关闭连接（四次握手）的过程：

由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。这原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接。收到一个FIN只意味着这一方向上没有数据流动，一个TCP连接在收到一个FIN后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。

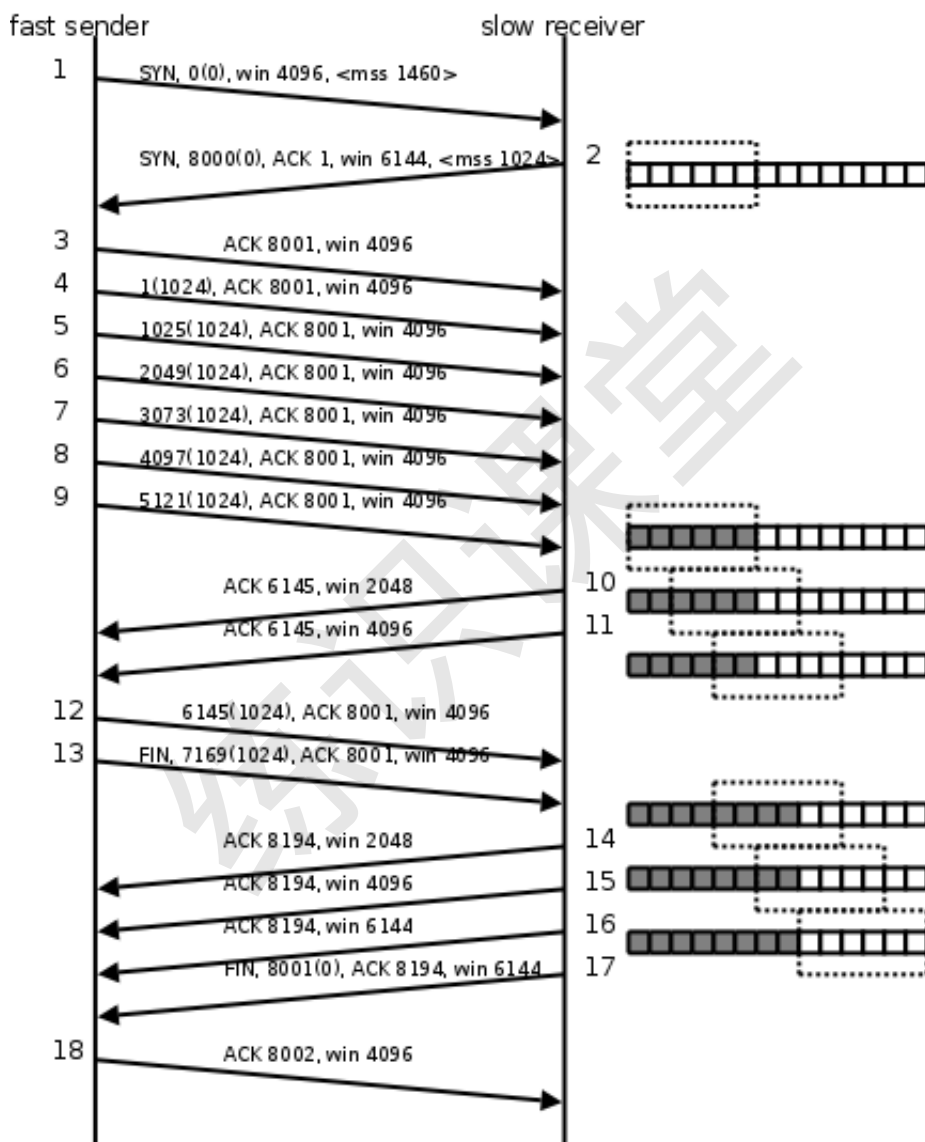
- 客户端发出段7，FIN位表示关闭连接的请求。
- 服务器发出段8，应答客户端的关闭连接请求。
- 服务器发出段9，其中也包含FIN位，向客户端发送关闭连接请求。

- 客户端发出段10，应答服务器的关闭连接请求。

建立连接的过程是三方握手，而关闭连接通常需要4个段，服务器的应答和关闭连接请求通常不合并在一个段中，因为有连接半关闭的情况，这种情况下客户端关闭连接之后就不能再发送数据给服务器了，但是服务器还可以发送数据给客户端，直到服务器也关闭连接为止。

## 4.2 滑动窗口 (TCP流量控制)

介绍UDP时描述了这样的问题：如果发送端发送的速度较快，接收端接收到数据后处理的速度较慢，而接收缓冲区的大小是固定的，就会丢失数据。TCP协议通过“滑动窗口 (Sliding Window)”机制解决这一问题。看下图的通讯过程：



滑动窗口

- 发送端发起连接，声明最大段尺寸是1460，初始序号是0，窗口大小是4K，表示“我的接收缓冲区还有4K字节空闲，你发的数据不要超过4K”。接收端应答连接请求，声明最大段尺寸是1024，初始序号是8000，窗口大小是6K。发送端应答，三方握手结束。
- 发送端发出段4-9，每个段带1K的数据，发送端根据窗口大小知道接收端的缓冲区满了，因此停止发送数据。
- 接收端的应用程序提走2K数据，接收缓冲区又有了2K空闲，接收端发出段10，在应答已收到6K数据的同时声明窗口大小为2K。



- 接收端的应用程序又提走2K数据，接收缓冲区有4K空闲，接收端发出段11，重新声明窗口大小为4K。
- 发送端发出段12-13，每个段带2K数据，段13同时还包含FIN位。
- 接收端应答接收到的2K数据（6145-8192），再加上FIN位占一个序号8193，因此应答序号是8194，连接处于半关闭状态，接收端同时声明窗口大小为2K。
- 接收端的应用程序提走2K数据，接收端重新声明窗口大小为4K。
- 接收端的应用程序提走剩下的2K数据，接收缓冲区全空，接收端重新声明窗口大小为6K。
- 接收端的应用程序在提走全部数据后，决定关闭连接，发出段17包含FIN位，发送端应答，连接完全关闭。

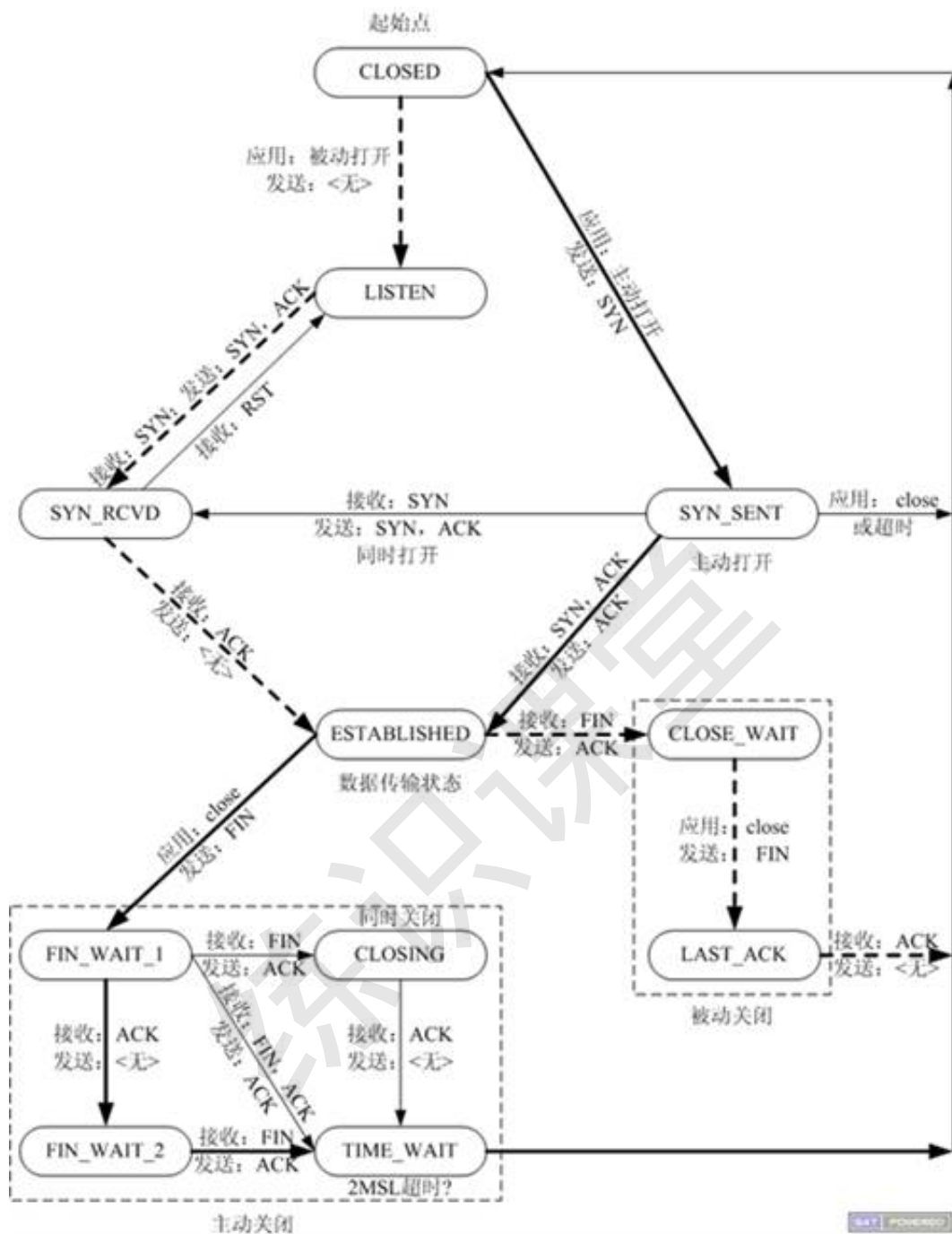
上图在接收端用小方块表示1K数据，实心的小方块表示已接收到的数据，虚线框表示接收缓冲区，因此套在虚线框中的空心小方块表示窗口大小，从图中可以看出，随着应用程序提走数据，虚线框是向右滑动的，因此称为滑动窗口。

从这个例子还可以看出，发送端是一K—K地发送数据，而接收端的应用程序可以两K两K地提走数据，当然也有可能一次提走3K或6K数据，或者一次只提走几个字节的数据。也就是说，应用程序所看到的数据是一个整体，或说是一个流（stream），在底层通讯中这些数据可能被拆成很多数据包来发送，但是一个数据包有多少字节对应用程序是不可见的，因此TCP协议是面向流的协议。而UDP是面向消息的协议，每个UDP段都是一条消息，应用程序必须以消息为单位提取数据，不能一次提取任意字节的数据，这一点和TCP是很不同的。

## 4.3 TCP状态转换

这个图N多人都知道，它排除和定位网络或系统故障时大有帮助，但是怎样牢牢地将这张图刻在脑中呢？那么你就一定要对这张图的每一个状态，及转换的过程有深刻的认识，不能只停留在一知半解之中。下面对这张图的11种状态详细解析一下，以便加强记忆！不过在这之前，先回顾一下TCP建立连接的三次握手过程，以及关闭连接的四次握手过程。

# TCP 状态转换图



TCP状态转换图

**CLOSED**：表示初始状态。

**LISTEN**：该状态表示服务器端的某个SOCKET处于监听状态，可以接受连接。

**SYN\_SENT**：这个状态与SYN\_RCVD遥相呼应，当客户端SOCKET执行CONNECT连接时，它首先发送SYN报文，随即进入到了SYN\_SENT状态，并等待服务端的发送三次握手中的第2个报文。SYN\_SENT状态表示客户端已发送SYN报文。

**SYN\_RCVD**：该状态表示接收到SYN报文，在正常情况下，这个状态是服务器端的SOCKET在建立TCP连接时的三次握手会话过程中的一个中间状态，很短暂。此种状态时，当收到客户端的ACK报文后，会进入到ESTABLISHED状态。

**ESTABLISHED:** 表示连接已经建立。

**FIN\_WAIT\_1:** FIN\_WAIT\_1和FIN\_WAIT\_2状态的真正含义都是表示等待对方的FIN报文。区别是：

FIN\_WAIT\_1状态是当socket在ESTABLISHED状态时，想主动关闭连接，向对方发送了FIN报文，此时该socket进入到FIN\_WAIT\_1状态。

FIN\_WAIT\_2状态是当对方回应ACK后，该socket进入到FIN\_WAIT\_2状态，正常情况下，对方应马上回应ACK报文，所以FIN\_WAIT\_1状态一般较难见到，而FIN\_WAIT\_2状态可用netstat看到。

**FIN\_WAIT\_2:** 主动关闭链接的一方，发出FIN收到ACK以后进入该状态。称之为半连接或半关闭状态。该状态下的socket只能接收数据，不能发。

**TIME\_WAIT:** 表示收到了对方的FIN报文，并发送出了ACK报文，等2MSL后即可回到CLOSED可用状态。如果FIN\_WAIT\_1状态下，收到对方同时带FIN标志和ACK标志的报文时，可以直接进入到TIME\_WAIT状态，而无须经过FIN\_WAIT\_2状态。

**CLOSING:** 这种状态较特殊，属于一种较罕见的状态。正常情况下，当你发送FIN报文后，按理来说是应该先收到（或同时收到）对方的ACK报文，再收到对方的FIN报文。但是CLOSING状态表示你发送FIN报文后，并没有收到对方的ACK报文，反而却也收到了对方的FIN报文。什么情况下会出现此种情况呢？如果双方几乎在同时close一个SOCKET的话，那么就出现了双方同时发送FIN报文的情况，也即会出现CLOSING状态，表示双方都正在关闭SOCKET连接。

**CLOSE\_WAIT:** 此种状态表示在等待关闭。当对方关闭一个SOCKET后发送FIN报文给自己，系统会回应一个ACK报文给对方，此时则进入到CLOSE\_WAIT状态。接下来呢，察看是否还有数据发送给对方，如果没有可以close这个SOCKET，发送FIN报文给对方，即关闭连接。所以在CLOSE\_WAIT状态下，需要关闭连接。

**LAST\_ACK:** 该状态是被动关闭一方在发送FIN报文后，最后等待对方的ACK报文。当收到ACK报文后，即可以进入到CLOSED可用状态。

## 4.4 半关闭

当TCP链接中A发送FIN请求关闭，B端回应ACK后（A端进入FIN\_WAIT\_2状态），B没有立即发送FIN给A时，A方处在半链接状态，此时A可以接收B发送的数据，但是A已不能再向B发送数据。

## 4.5 2MSL

2MSL (Maximum Segment Lifetime) TIME\_WAIT状态的存在有两个理由：

（1）让4次握手关闭流程更加可靠；4次握手的最后一个ACK是由主动关闭方发送出去的，若这个ACK丢失，被动关闭方会再次发一个FIN过来。若主动关闭方能够保持一个2MSL的TIME\_WAIT状态，则有更大的机会让丢失的ACK被再次发送出去。

（2）防止lost duplicate对后续新建正常链接的传输造成破坏。lost duplicate在实际的网络中非常常见，经常是由于路由器产生故障，路径无法收敛，导致一个packet在路由器A，B，C之间做类似死循环的跳转。IP头部有个TTL，限制了一个包在网络中的最大跳数，因此这个包有两种命运，要么最后TTL变为0，在网络中消失；要么TTL在变为0之前路由器路径收敛，它凭借剩余的TTL跳数终于到达目的地。但非常可惜的是TCP通过超时重传机制在早些时候发送了一个跟它一模一样的包，并先于它达到了目的地，因此它的命运也就注定被TCP协议栈抛弃。

另外一个概念叫做incarnation connection，指跟上次socket pair一模一样的新连接，叫做incarnation of previous connection。lost uplicate加上incarnation connection，则会对我们的传输造成致命的错误。

TCP是流式的，所有包到达的顺序是不一致的，依靠序列号由TCP协议栈做顺序的拼接；假设一个incarnation connection这时收到的seq=1000，来了一个lost duplicate为seq=1000，len=1000，则TCP认为这个lost duplicate合法，并存放入了receive buffer，导致传输出现错误。通过一个2MSL TIME\_WAIT状态，确保所有的lost duplicate都会消失掉，避免对新连接造成错误。

该状态为什么设计在主动关闭这一方：

(1) 发最后ACK的是主动关闭一方。

(2) 只要有一方保持TIME\_WAIT状态，就能起到避免incarnation connection在2MSL内的重新建立，不需要两方都有。

如何正确对待2MSL TIME\_WAIT？

RFC要求socket pair在处于TIME\_WAIT时，不能再起一个incarnation connection。但绝大部分TCP实现，强加了更为严格的限制。在2MSL等待期间，socket中使用的本地端口在默认情况下不能再被使用。

若A 10.234.5.5 : 1234和B 10.55.55.60 : 6666建立了连接，A主动关闭，那么在A端只要port为1234，无论对方的port和ip是什么，都不允许再起服务。这甚至比RFC限制更为严格，RFC仅仅是要求socket pair不一致，而实现当中只要这个port处于TIME\_WAIT，就不允许起连接。这个限制对主动打开方来说是无所谓的，因为一般用的是临时端口；但对于被动打开方，一般是server，就悲剧了，因为server一般是熟知端口。比如http，一般端口是80，不可能允许这个服务在2MSL内不能起来。

解决方案是给服务器的socket设置SO\_REUSEADDR选项，这样的话就算熟知端口处于TIME\_WAIT状态，在这个端口上依旧可以将服务启动。当然，虽然有了SO\_REUSEADDR选项，但sockt pair这个限制依旧存在。比如上面的例子，A通过SO\_REUSEADDR选项依旧在1234端口上起了监听，但这时我们若是从B通过6666端口去连它，TCP协议会告诉我们连接失败，原因为Address already in use.

RFC 793中规定MSL为2分钟，实际应用中常用的是30秒，1分钟和2分钟等。

RFC (Request For Comments)，是一系列以编号排定的文件。收集了有关因特网相关资讯，以及UNIX和因特网社群的软件文件。



微信搜一搜



练识课堂