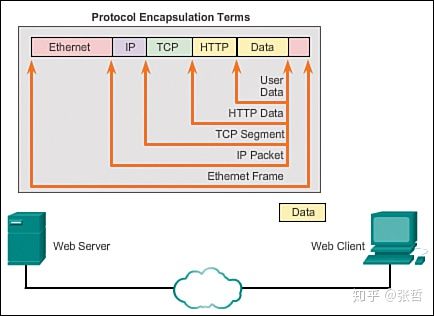
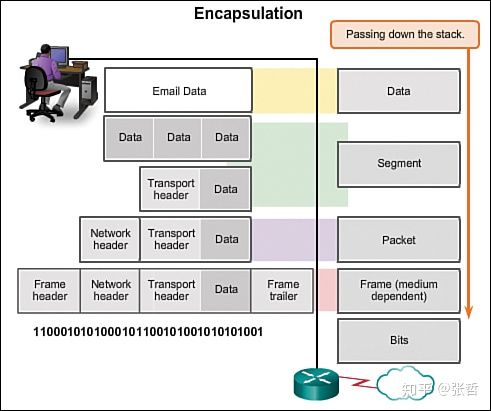
# .服务器->客户端 数据传输过程示例



1. 需要传送的数据是网络服务器的HTML页面。
2. 应用协议HTTP报文头添加到HTML数据之前。报文头信息包括：服务器所使用的HTTP版本，以及表明它包含发给网络客户端信息的状态编码。
3. HTTP应用层协议将HTML格式的网页数据发送给传输层。TCP传输层用于管理网络服务器和客户端之间的会话。
4. IP信息添加到TCP信息之前。IP指定适当的源和目的IP地址。这些信息就构成了IP报文。
5. 以太网协议添加到IP报文的两端之后，就形成了数据链路帧。上述帧发送至通向网络客户端的路径上的最近一个路由器。路由器移除以太网信息，观察IP报文，判定最佳路径，将报文插入一个新的帧，并发送至目标路径上下一个相邻路由器。每一个路由器在转发之前都移除并添加新的数据链路层信息。
6. 数据通过互联网络传输，互联网络包含媒介和中间设备。
7. 客户端接收到包含数据的数据链路帧，处理各层协议头，之后以添加时相反的顺序移除协议头。首先处理并移除以太网信息，之后是IP协议信息，接下来TCP信息，最后是HTTP信息。
8. 之后，将网页信息传递给客户端网页浏览器软件。

## 协议数据单元（Protocol Data Unit, PDU）



PDU：数据片段在各层网络结构中采用的形式就称为协议数据单元。

数据封装：应用层数据在传输过程中沿着协议栈传递，每一层协议都会向其中添加信息。这就是封装的过程。

数据（Data）：应用层PDU的常用术语

分段（Segment）：传输层PDU

包（Packet）：网络层PDU

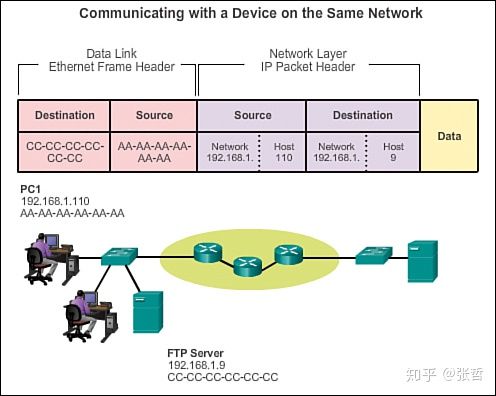
帧（Frame）：链路层PDU

比特（Bits）：在介质上物理传输数据所使用的PDU。

## 网络地址

IP地址包含两个部分：网络前缀和主机。

路由器使用网络前缀部分将报文转发给适当的网络。最后一个路由器使用主机部分将报文发送给目标设备。同一本地网络中，网络前缀部分是相同的，只有主机设备地址部分不同。[IP地址与子网](#_IP地址与子网)



## 数据链路地址

数据链路地址的目的是在同一网络中将数据链路帧从一个网络接口发送至另一个网络接口。

MAC地址是物理植入网卡的48比特地址。

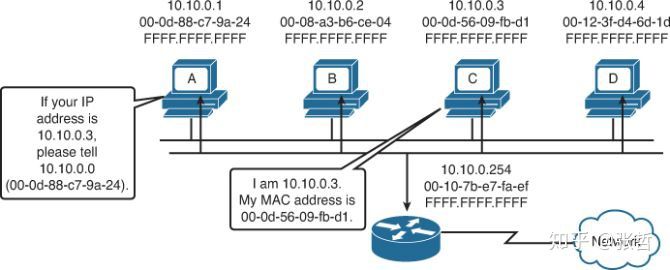
当IP报文的发送方和接收方位于同一网络，数据链路帧直接发送到接收设备。

### 获取ip地址

DNS或手动输入

### 获取mac地址（LAN）

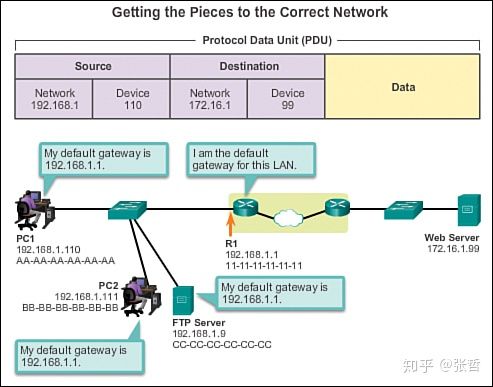
**地址解析协议（Address Resolution Protocol, ARP）**

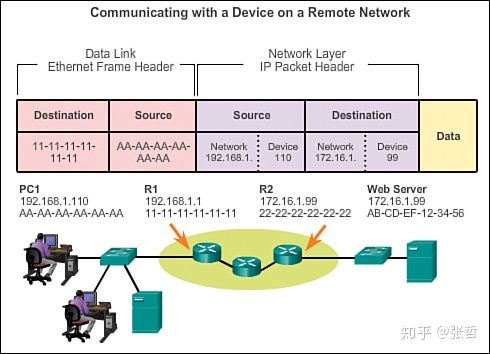


发送方主机使用地址解析协议（Address Resolution Protocol, ARP）以检测本地网络的所有MAC地址。如下图所示，发送主机在整个LAN发送ARP请求消息，这是一条广播消息。ARP请求包含目标设备的IP地址，LAN上的每一个设备都会检查该ARP请求，看看是否包含它自身的IP地址。只有符合该IP地址的设备才会发送ARP响应。ARP响应包含ARP请求中IP地址相对应的MAC地址。

### 访问远程资源

当主机发送消息到远端网络，必须使用路由器，也称为默认网关。





# TCP

## TCP三次握手和四次挥手

### TCP服务模型

1. 一个TCP连接由一个4元组构成，分别是两个IP地址和两个端口号

（其实是客户端和服务器的内存里保存的一份关于对方的信息）

2、TCP的确认机制（ACK）：

1）当TCP接收到另一端的数据时，它会发送一个确认，但这个确认不会立即发送，一般会延迟一会儿。

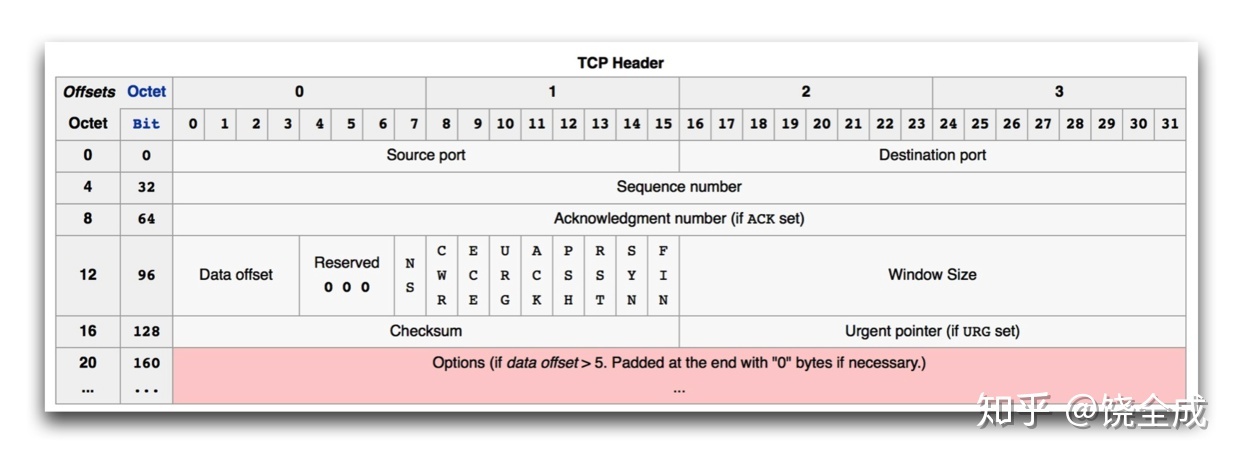
2）ACK是累积的，N表示N以前的都被正确接受了

3、TCP是全双工的

4、序列号：用于求其重复到达的数据，将乱序到达的数据排序后交给上层。

（TCP接收端会先保留大序列号的数据不交给上层应用程序，直到缺失的小序列号的报文段被填满。）

### TCP头部



头部最大为60字节，最小为20字节。

源端口和目的端口在TCP层确定双方进程，序列号表示的是报文段数据中的第一个字节号，ACK表示确认号，该确认号的发送方期待接收的下一个序列号。

data offset:TCP头的长度

URG——紧急指针是否有效。

ACK —— 确认，使得确认号有效。

PSH——提示接收端，从TCP接收缓冲区中读走数据

RST —— 重置连接（经常看到的reset by peer）就是此字段搞的鬼。

SYN —— 用于初如化一个连接的序列号。

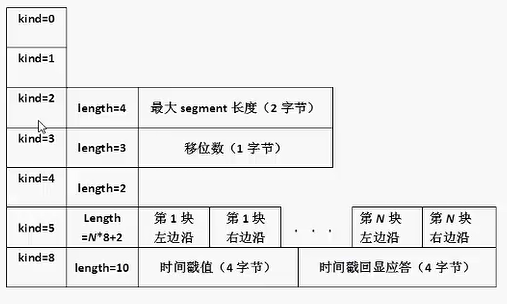
FIN —— 该报文段的发送方已经结束向对方发送数据。

当一个连接被建立或被终止时，交换的报文段只包含TCP头部，而没有数据。

window size:窗口大小，表示想收到的TCP数据段的大小

checksum：校验和：发送端填充，用CRC算法。

**TCP头部选项字段**：



Kind=0 选项表结束选项

Kind=1 空操作选项

Kind=2 最大报文段长度选项

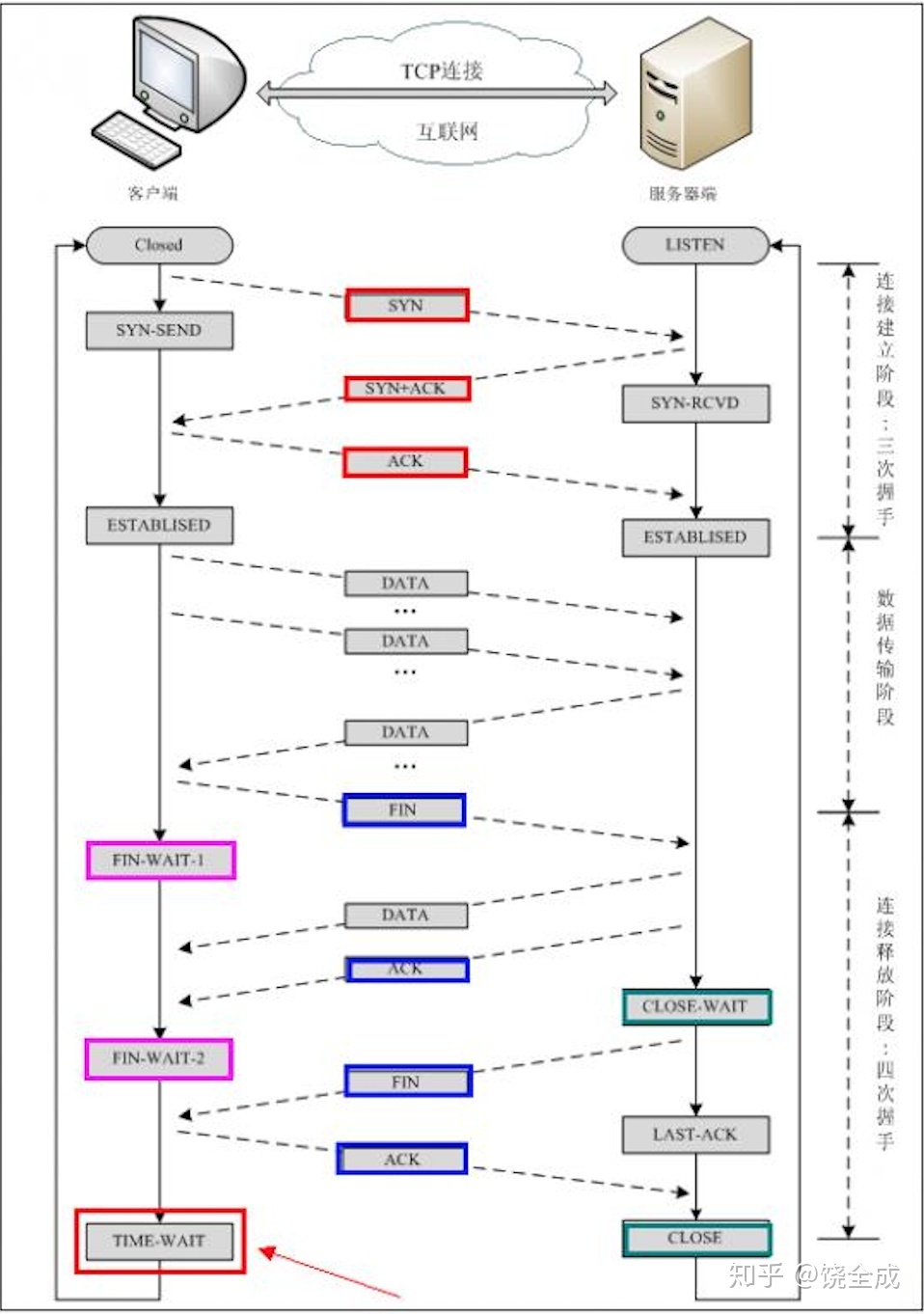
Kind=3 窗口扩大因子选项 TCP连接初始化时，通讯双方使用该选项来协商接受通告窗口的扩大因子 只能出现在同步报文段中

**Kind=4 选择确认选项（SACK） TCP通信中，某个报文段丢失，TCP模块会重发最后确认的TCP报文段的后续所有报文段，这可能导致重复，SACK技术直接发送丢失的报文段就好。**

Kind=5 SACK实际工作选项 该选项参数告诉发送方本端已经收到并缓存不连续的数据块，发送端根据此选项检查并重发丢失的数据块

Kind=8 时间戳选项 比较准确的计算通讯双方的回路时间的方法

### 状态转换



#### 三次握手

三次握手”的作用就是

1）双方都能明确自己和对方的收、发能力是正常的

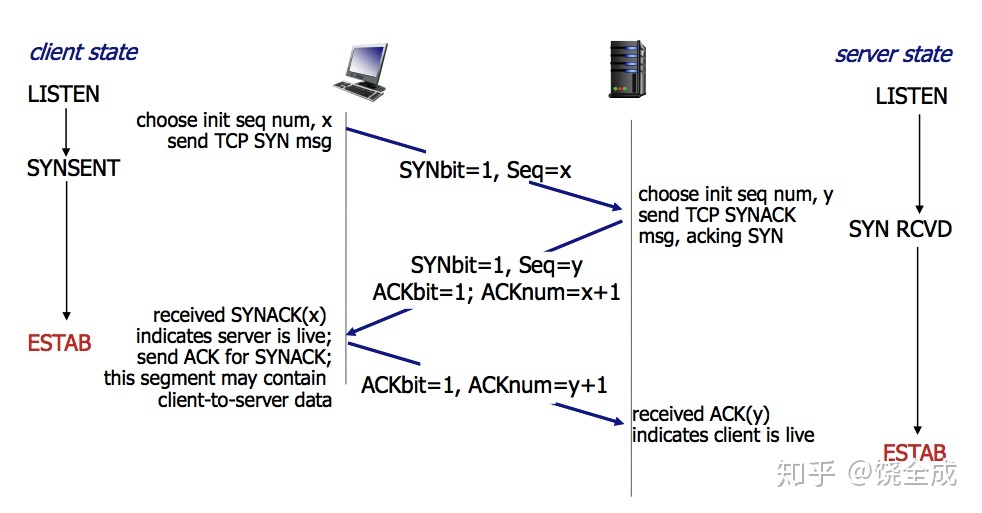
2）利用数据包的选项来传输特殊的信息，交换初始序列号ISN。

第一次握手：客户端发送网络包，服务端收到了。这样服务端就能得出结论：客户端的发送能力、服务端的接收能力是正常的。

第二次握手：服务端发包，客户端收到了。这样客户端就能得出结论：服务端的接收、发送能力，客户端的接收、发送能力是正常的。（下一次的“响应”是由于第一次的“请求”触发，因此每次握手其实是可以得到额外的结论的）

第三次握手：客户端发包，服务端收到了。这样服务端就能得出结论：客户端的接收、发送能力，服务端的发送、接收能力是正常的。

两次达不到让双方都得出自己、对方的接收、发送能力都正常的结论！



* **ISN**

三次握手的一个重要功能是客户端和服务端交换ISN(Initial Sequence Number), 以便让对方知道接下来接收数据的时候如何按序列号组装数据。

如果ISN是固定的，攻击者很容易猜出后续的确认

ISN = M + F(localhost, localport, remotehost, remoteport)

M是一个计时器，每隔4毫秒加1。 F是一个Hash算法

* **序列号回绕**

因为ISN是随机的，所以序列号容易就会超过2^31-1. 而tcp对于丢包和乱序等问题的判断都是依赖于序列号大小比较的。此时就出现了所谓的tcp序列号回绕（sequence wraparound）问题。

*/\**

*\* The next routines deal with comparing 32 bit unsigned ints*

*\* and worry about wraparound (automatic with unsigned arithmetic).*

*\*/*

**static** **inline** **int** **before**(\_\_u32 seq1, \_\_u32 seq2)

{

**return** (\_\_s32)(seq1**-**seq2) **<** 0;

}

**#define after(seq2, seq1) before(seq1, seq2)**

假设seq1=255， seq2=1（发生了回绕）。

seq1 = 1111 1111 seq2 = 0000 0001

我们希望比较结果是

seq1 - seq2=

1111 1111

-0000 0001

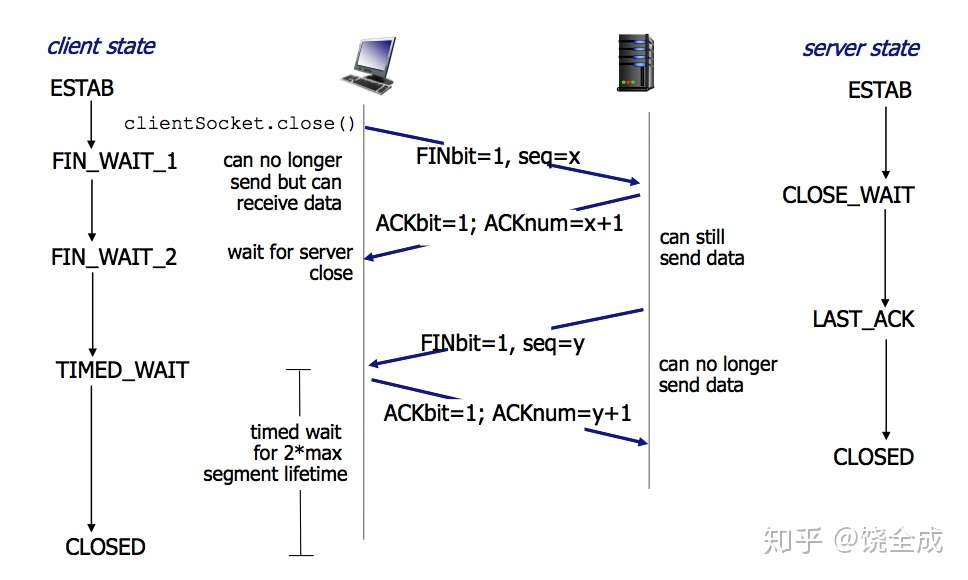
-----------

1111 1110

由于我们将结果转化成了有符号数，由于最高位是1，因此结果是一个负数

#### 四次挥手

TCP连接是双向传输的对等的模式，就是说双方都可以同时向对方发送或接收数据。当有一方要关闭连接时，会发送指令告知对方，我要关闭连接了。这时对方会回一个ACK，此时一个方向的连接关闭。但是另一个方向仍然可以继续传输数据，等到发送完了所有的数据后，会发送一个FIN段来关闭此方向上的连接。接收方发送ACK确认关闭连接。



当T C P执行一个主动关闭，并发回最后一个A CK,该连接必须在TIME\_WAIT状态停留的时间为2倍的MSL。这样可让TCP再次发送最后的ACK以防这个ACK丢失（另一端超时并重发最后的FIN)。

存在这个规则导致一个后果就是在这个2MSL的时间内，该地址上的链接（客户端地址、端口和服务器端的地址、端口）不能被使用。

#### syn flood攻击

如果恶意的向某个服务器端口发送大量的SYN包，则可以使服务器打开大量的半开连接，分配TCB（Transmission Control Block：**存储了每一个连接中的一些重要信息：TCP连接表，到发送和接收缓存的指针，到重传队列的指针，当前的发送和接收序号等等**）, 从而消耗大量的服务器资源，同时也使得正常的连接请求无法被响应。

当开放了一个TCP端口后，该端口就处于Listening状态，不停地监视发到该端口的Syn报文，一 旦接收到Client发来的Syn报文，就需要为该请求分配一个TCB，通常一个TCB至少需要280个字节，在某些操作系统中TCB甚至需要1300个字节，并返回一个SYN ACK命令，立即转为SYN-RECEIVED即半开连接状态。系统会为此耗尽资源。

* **解决办法**

1、无效连接的监视释放

监视系统的半开连接和不活动连接，当达到一定阈值时拆除这些连接，从而释放系统资源。这种方法对于所有的连接一视同仁，而且由于SYN Flood造成的半开连接数量很大，正常连接请求也被淹没在其中被这种方式误释放掉，因此这种方法属于入门级的SYN Flood方法。

2、延缓TCB分配方法

消耗服务器资源主要是因为当SYN数据报文一到达，系统立即分配TCB，从而占用了资源。而SYN Flood由于很难建立起正常连接，因此，当正常连接建立起来后再分配TCB则可以有效地减轻服务器资源的消耗。常见的方法是使用Syn Cache和Syn Cookie技术。

Syn Cache技术

系统在收到一个SYN报文时，在一个专用HASH表中保存这种半连接信息，直到收到正确的回应ACK报文再分配TCB。这个开销远小于TCB的开销。当然还需要保存序列号。

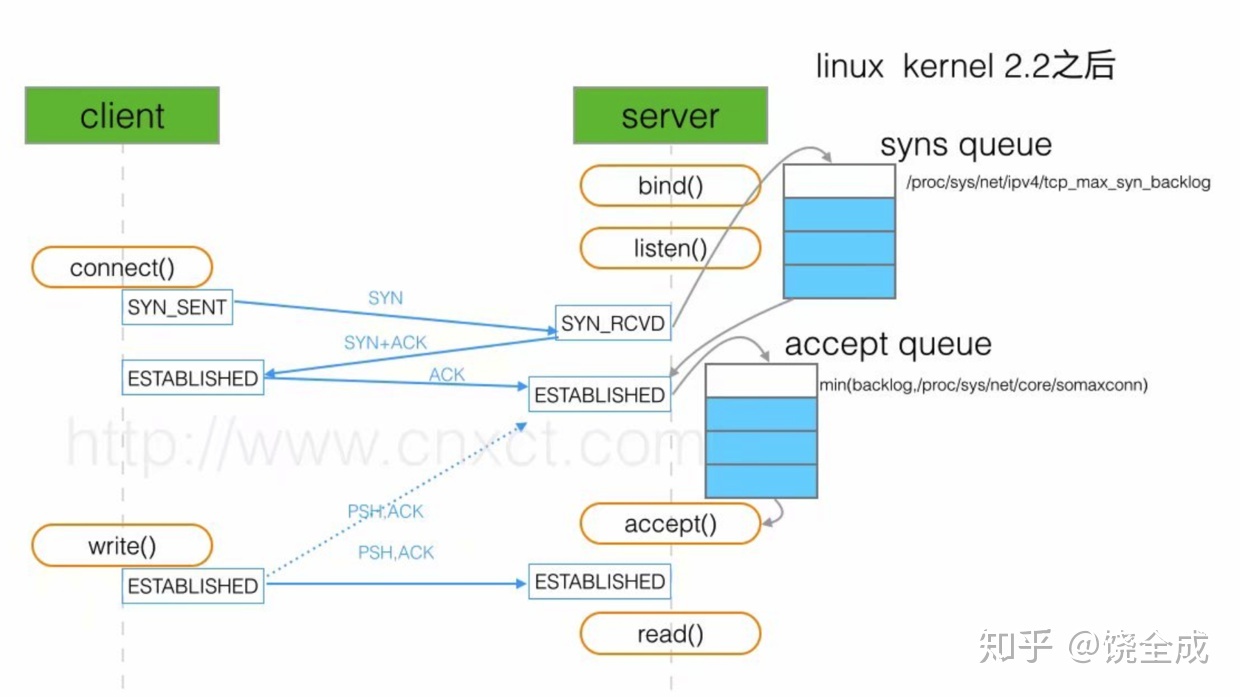
Syn Cookie技术

Syn Cookie技术则完全不使用任何存储资源，这种方法比较巧妙，它使用一种特殊的算法生成Sequence Number，这种算法考虑到了对方的IP、端口、己方IP、端口的固定信息，以及对方无法知道而己方比较固定的一些信息，如MSS(Maximum Segment Size，最大报文段大小，指的是TCP报文的最大数据报长度，其中不包括TCP首部长度。)、时间等，在收到对方 的ACK报文后，重新计算一遍，看其是否与对方回应报文中的（Sequence Number-1）相同，从而决定是否分配TCB资源。

**3、使用SYN Proxy防火墙**

防火墙确定了连接的安全后，会发出一个safe reset命令，client会进行重新连接，这时出现的syn报文会直接放行。但是，client需要发起两次握手过程，因此建立连接的时间将会延长。

**4、连接队列**



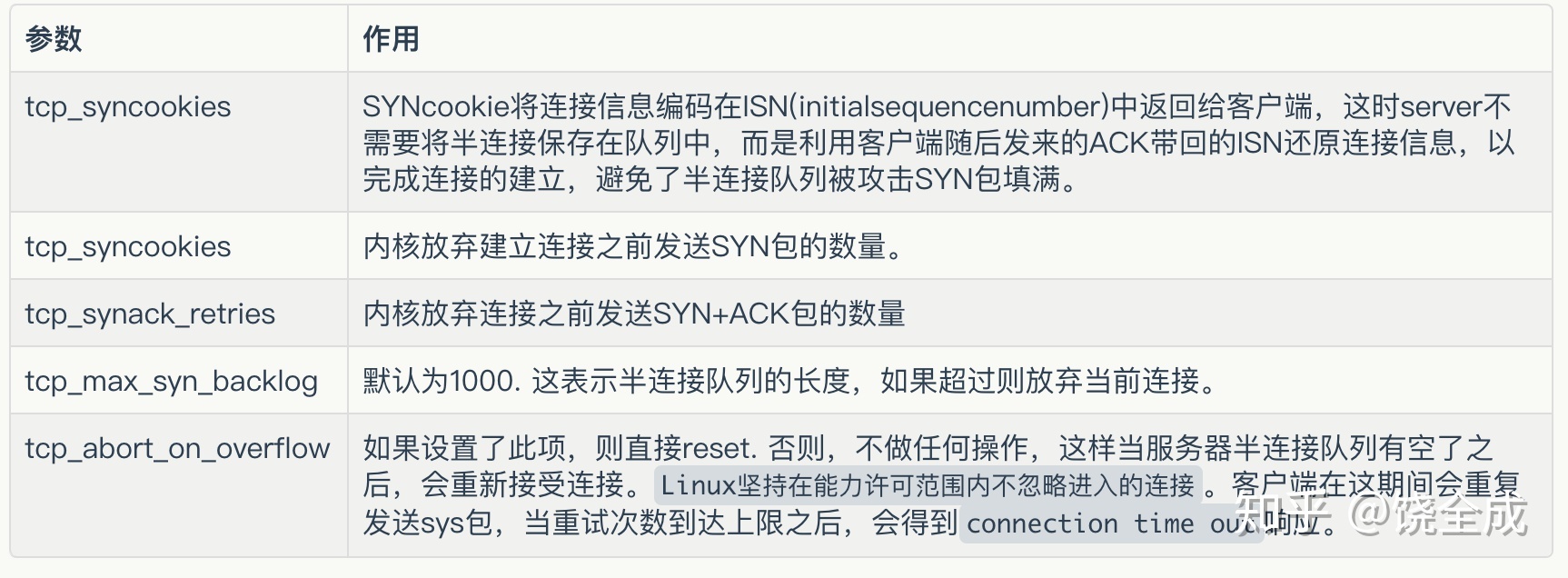
**处于SYN\_RCVD状态的半连接队列：**

该队列为每个客户端的SYN包开设一个条目(服务端在接收到SYN包的时候，就已经创建了request\_sock结构，存储在半连接队列中)，该条目表明服务器已收到SYN包，并向客户发出确认，正在等待客户的确认包。这些条目所标识的连接在服务器处于Syn\_RECV状态，当服务器收到客户的确认包时，删除该条目，服务器进入ESTABLISHED状态。

**处于ESTABLISHED状态但仍未被应用程序accept的为全连接队列：**

* 半连接队列满

Linux下默认会进行5次重发SYN-ACK包，重试的间隔时间从1s开始，下次的重试间隔时间是前一次的双倍，5次的重试时间间隔为1s, 2s, 4s, 8s, 16s, 总共31s, 称为指数退避，第5次发出后还要等32s才知道第5次也超时了，所以，总共需要 1s + 2s + 4s+ 8s+ 16s + 32s = 63s, TCP才会把断开这个连接。



* 全连接队列满

当第三次握手时，当server接收到ACK包之后，会进入一个新的叫 accept 的队列。

当accept队列满了之后，即使client继续向server发送ACK的包，也会不被响应，此时ListenOverflows+1，同时server通过tcp\_abort\_on\_overflow来决定如何返回，0表示直接丢弃该ACK，1表示发送RST通知client；相应的，client则会分别返回read timeout 或者 connection reset by peer。另外，tcp\_abort\_on\_overflow是0的话，server过一段时间再次发送syn+ack给client（也就是重新走握手的第二步），如果client超时等待比较短，就很容易异常了。而客户端收到多个 SYN ACK 包，则会认为之前的 ACK 丢包了。于是促使客户端再次发送 ACK ，在 accept队列有空闲的时候最终完成连接。若 accept队列始终满员，则最终客户端收到 RST 包（此时服务端发送syn+ack的次数超出了tcp\_synack\_retries）。

* **小结**

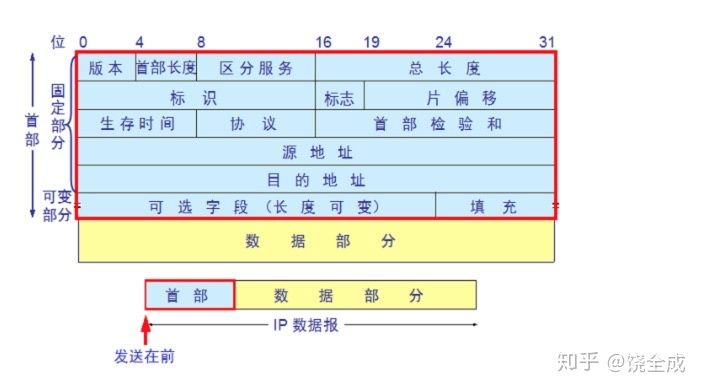
当外部连接请求到来时，TCP模块会首先查看max\_syn\_backlog，如果处于SYN\_RCVD状态的连接数目超过这一阈值，进入的连接会被拒绝。根据tcp\_abort\_on\_overflow字段来决定是直接丢弃，还是直接reset.

从服务端来说，三次握手中，第一步server接受到client的syn后，把相关信息放到半连接队列中，同时回复syn+ack给client. 第三步当收到客户端的ack, 将连接加入到全连接队列。

一般，全连接队列比较小，会先满，此时半连接队列还没满。如果这时收到syn报文，则会进入半连接队列，没有问题。但是如果收到了三次握手中的第3步(ACK)，则会根据tcp\_abort\_on\_overflow字段来决定是直接丢弃，还是直接reset.此时，客户端发送了ACK, 那么客户端认为三次握手完成，它认为服务端已经准备好了接收数据的准备。但此时服务端可能因为全连接队列满了而无法将连接放入，会重新发送第2步的syn+ack, 如果这时有数据到来，服务器TCP模块会将数据存入队列中。一段时间后，client端没收到回复，超时，连接异常，client会主动关闭连接。

#### 抓包分析

IP报文：固定长度20B+可变长度构成。

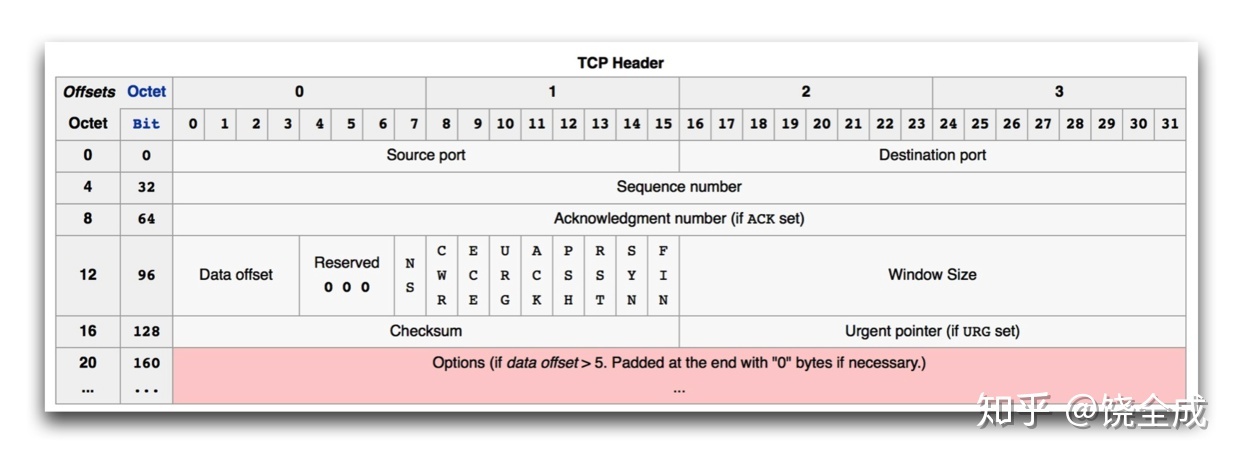


0x0000: 4500 003c 08cf 4000 3606 14a5 0ab3 b561

0x0010: 0a60 5cd4 989e 18eb f65a ebff 0000 0000

0x0020: a002 7210 872f 0000 0204 05b4 0402 080a

0x0030: b062 e330 0000 0000 0103 0307

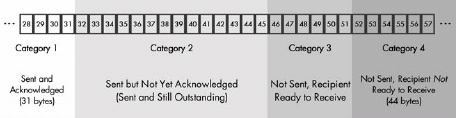


## TCP滑动窗口（2.2-2.5都是讲流量控制）

TCP的主要任务是很简单：打包和发送数据。

TCP通过滑动窗口防止发送方发的太快，耗尽接收方的资源，从而使接收方来不及处理。

### TCP数据流的类别



**1.** **已发送已确认** 数据流中最早的字节已经发送并得到确认。这些数据是站在发送设备的角度来看的。如下图所示，31个字节已经发送并确认。

**2.** **已发送但尚未确认** 已发送但尚未得到确认的字节。发送方在确认之前，不认为这些数据已经被处理。下图所示14字节为第2类。

**3.** **未发送而接收方已Ready** 设备尚未将数据发出，但接收方根据最近一次关于发送方一次要发送多少字节确认自己有足够空间。发送方会立即尝试发送。如图，第3类有6字节。

**4.** **未发送而接收方Not Ready** 由于接收方not ready，还不允许将这部分数据发出。

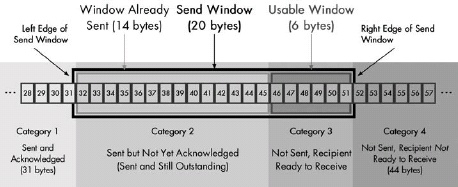
接收方机制类似：

已接收并已确认、未接受但准备好接收，以及尚未接收并尚未准备好接收的数据。

### 发送窗口与可用窗口

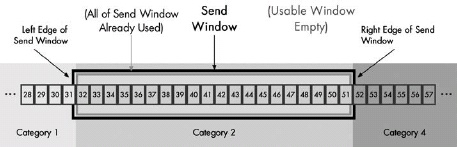
**发送窗口**：接收方允许发送方一次能容纳的未确认的字节数，该窗口决定了发送方允许传送的字节数，也是2类和3类的字节数之和。

**可用窗口**：发送方仍被允许发送的数据量，即第3类的大小。



### 窗口滑动过程

1、初始化状态：



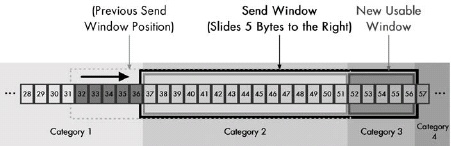
2、传输过程：

假设**已发送未确认（第二类）**字节（32-45）分为4个segment传输，32-34,35-36，37-41,42-45，第1,2,4已经到达，第3段没有到达。

此时发送方只会收到ack=37。

**3、调整窗口：**

当发送设备接收到确认信息，则会将一部分第2类字节转移到第1类，因为它们已经得到了确认。



由于5个字节已被确认，窗口大小没有改变，允许发送方多发5个字节。结果，窗口向右滑动5个字节。同时5个字节从第二类移动到第1类，5个字节从第4类移动至第3类，为接下来的传输创建了新的可用窗口。

4、处理丢失确认信息：

发送设备可以将新的字节添加到第3类之后，即52-56。但是发送设备之后会停止发送，窗口停留在37-41。

等到TCP重传机制触发，重新发送数据（可能不仅发37-41,42-45也会被重发，因为TCP不会对每一个片段分别确认。）

## TCP重传机制

### TCP片段重传计时器以及重传队列

1、**放置于重传队列中，计时器开始**

 包含数据的片段一经发送，片段的一份副本就放在名为**重传队列**的数据结构中，此时**启动重传计时器**。因此，在某些时间点，每一个片段都会放在队列里。队列按照重传计时器的剩余时间来排列，因此TCP软件可追踪那几个计时器在最短时间内超时。

2、**确认处理**

如果在计时器超时之前收到了确认信息，则该片段从重传队列中移除。

3、**超时重传**

重传之后的片段依旧保留在重传队列。

重启计时器

再失败、再重复（不会一直重传）

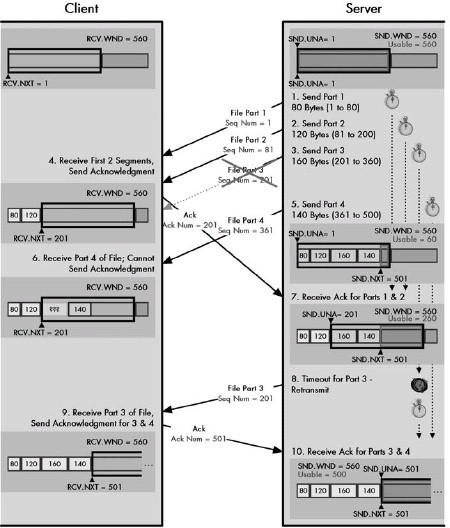
例子：

**片段1** 序列号字段是1片段长度80。所以片段1中最后一个序列号是80。

**片段2** 序列号是81片段长度是120。片段2中最后一个序列号是200。

**片段3** 序列号是201片段长度是160。片段3中最后一个序列号是360。

**片段4** 序列号是361片段长度是140。片段3中最后一个序列号是500。



需要通过一个称为自适应重传的过程来动态调整超时值。

## TCP确认机制

当片段3丢失后，发送方的滑动窗口不会滑动。

接收方无法告知发送方后续片段的ack。

由此看出**累计确认的缺点**：假设第一个丢失了，后面的即使被接收方收到了，也无法返回ack。

**解决方法：**

**1、重传超时的片段**

最坏情况下，所有20个片段都丢失了。我们需要等片段1超时并重传。这一片段也许会得到确认，但之后我们需要等待片段2超时并重传。这一过程会重复多次。

**2、重传所有片段**

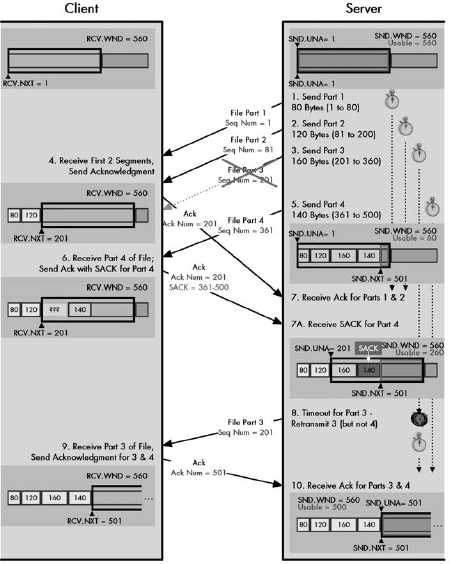
很多重传是不必要的。如果第一个片段丢失而其他19个实际上接收到了，则浪费网络资源

3、**选择确认**

解决方式是对TCP滑动窗口算法进行扩展，添加允许设备**分别确认非连续片段**的功能。

1）连接时使用的SYN片段来协商是否允许SACK。

2）SACK选项。这一选项包含一个关于已接收但未确认片段数据sequence number范围的列表，由于它们是非连续的。各设备对重传队列进行修改，如果该片段已被选择确认过，则该片段中的SACK比特位置为1。一个片段重传之后，之后所有片段也会重传，**除非SACK比特位为1**。



## TCP滑动窗口调整与流量控制

客户端与服务器的连接中，客户端告知服务器它一次希望从服务器接收多少字节数据，这是客户端的接收窗口，即服务器的发送窗口。类似地，服务器告知客户端一次希望从客户端接收多少字节数据，也就是服务器的接收窗口和客户端的发送窗口。

它代表了**设备对于特定连接的接收缓存大小**。即，窗口大小代表一个设备一次能够从对端处理多少数据，之后再传递给应用层处理。

* **操作**

服务器从客户端接受数据后进行两个操作：

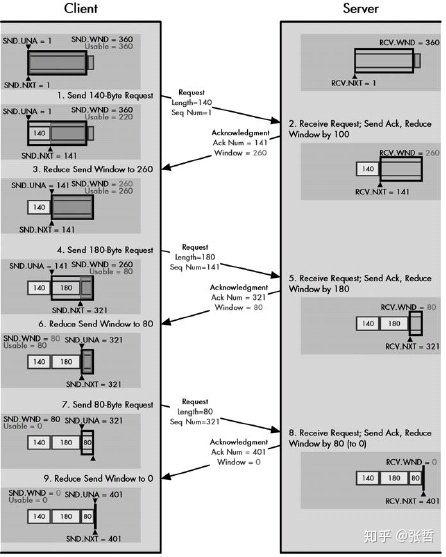
操作1：确认->回复ack

操作2：传输->传递给应用程序

**数据被服务器确认了，并不意味着已经从缓冲区取走了。**当服务器的接收速率>处理速率，会把缓冲区填满。此时需要增加或缩小窗口大小，让服务器和客户端能够确保对端发送数据的速度等同于处理速度。

### 减小窗口大小以降低发送速率

假设我们接收到140字节，但只能**发送40字节给应用程序，缓存中剩下100字节。**当发送140字节的确认信息，**服务器将发送窗口缩小100字节，至260字节。**当客户端从服务器接收到这一片段，它将会看到140字节的确认信息并将窗口向右滑动140字节。在滑动过程中，将大小缩减至260字节。可以认为将**窗口左端滑动140字节，但右端仅滑动40字节。**新的稍小一些的窗口保证服务器从客户端接收最多260字节数据，以适应接收缓存中的剩余空间.



## 拥塞控制

* 拥塞控制与流量控制的区别：

**拥塞控制**：TCP模块为了防止过多的数据注入网络，使**网络中的路由器或链路不致于过载**，以此提高网络利用率。拥塞问题是一个全局性的问题,涉及到所有的主机、所有的路由器、以及与降低网络传输性能有关的所有因素。

**流量控制：**流量控制往往指的是**点对点**通信量的控制，是个**端到端的**问题。流量控制所要做的就是控制**发送端**发送数据的速率，以便使**接收端**来得及接受。

### 慢启动

发送端向网络一次连续写入的数据量，我们称为SWND（Send Window**，发送窗口**）

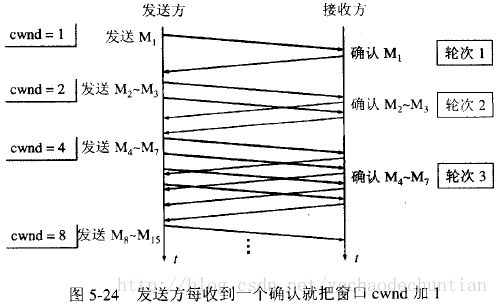
发送端需要合理的选择SWND的大小，如果SWND太小，会引起明显的网络延迟；反之，如果SWND太大，则容易导致网络拥塞。所以还需要引入一个称为**拥塞窗口**（Congestion Window,CWND）的状态变量

发送方维持一个拥塞窗口CWND的状态变量。它的大小取决于网络的拥塞程度，并且在动态的变化，发送方会让自己的发送窗口等于这个拥塞窗口。

发送方控制拥塞窗口的原则是：

（1）只要网络没有出现拥塞，拥塞窗口就再增大一些，以便把更多的分组发送出去。

（2）但只要网络出现拥塞，拥塞窗口就减小一些，以减少注入到网络中的分组数。



慢开始的“慢”并不是指cwnd的增长速率慢，而是指在TCP开始发送报文段时先设置cwnd=1，使得发送方在开始时只发送一个报文段（目的是试探一下网络的拥塞情况），然后再逐渐增大cwnd。

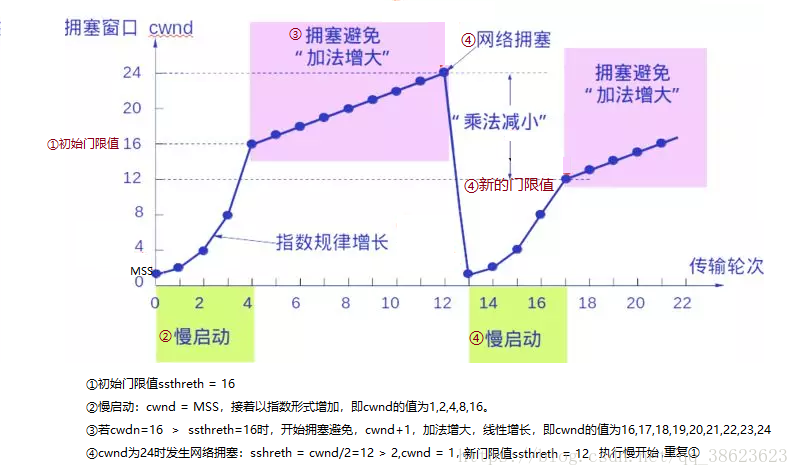
当 cwnd < ssthresh 时，使用上述的慢开始算法。

    当 cwnd > ssthresh 时，停止使用慢开始算法而改用拥塞避免算法。

    当 cwnd = ssthresh 时，既可使用慢开始算法，也可使用拥塞控制避免算法。

### 拥塞避免

让拥塞窗口cwnd缓慢地增大，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1，而不是加倍。这样拥塞窗口cwnd按线性规律缓慢增长，比慢开始算法的拥塞窗口增长速率缓慢得多。



只要发送方判断出网络拥塞，不论是在慢开始还是拥塞控制阶段，都要把慢开始门限值设置为出现拥塞时发送端窗口大小的一半，但不能小于2。然后把cwnd重新置为1，执行慢开始算法。

门限值减半，cwnd重置为1，做目的是减少发送到网络中的分组数，使得发生拥塞的路由器能够有时间能把队列中积压的分组处理掉。

发送端判断网络拥塞的依据：

1. 传送超时，即TCP重传定时器溢出

②收到重复的确认报文

### 快重传

快重传算法要求接收方每收到一个失序的报文段后就立即发出重复确认，而不要等到自己发送数据时才进行捎带确认。发送方只要一连收到3个同样的确认报文就应当立即重传数据报，不必等待报文段的重传计时器到期。

### 快恢复

把慢开始门限减半，“乘法减小”，将cwnd设置为新的慢开始门限值，继续执行拥塞避免算法，“加法增大”。

# IP地址与子网

## IP地址类型

A类网络：高位必须是“0”， 地址范围从1.0.0.0 到126.0.0.0。

对应子网掩码：255.0.0.0(8位)

B类网络 ：最高位必须是“10”， 地址范围从128.0.0.0到191.255.255.255

对应子网掩码：255.255.0.0（16位）

C类网络：的最高位必须是“110”， 范围从192.0.0.0到223.255.255.255。

对应子网掩码：255.255.255.0（24位）

全零（“0．0．0．0”）地址对应于当前主机。全“1”的IP地址（“255．255．255．255”）是当前子网的广播地址。

D类地址用于多点广播（Multicast）：IP地址第一个字节以“lll0”开始

E类地址以“llll0”开始，为将来使用保留

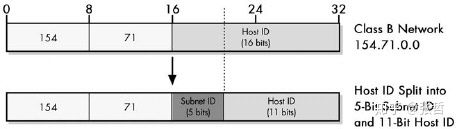
## 子网掩码作用：

IP地址未划分子网：网络号+主机号

IP地址划分子网：网络号+（子网号+子网主机号）

1. 将大的IP网络划分成多个子网络
2. 判断该ip地址是否在局域网上

### 例子1：

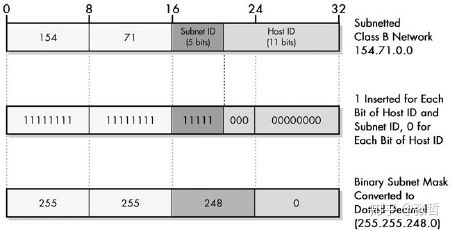


B类网络154.71.0.0，网络ID占16位（154.71），主机ID占16位。

没有子网的情况下一共可容纳65，534台主机

选取1位做子网，则得到两个子网（该位位0或1），每个子网有32768台主机（2的15次方）

以5位作为子网ID，11位作为主机ID为例：



### 例子2：判断两台主机是否在同一个局域网

IP1：210.73.140.5==》主机标识为5=>**000**00101

IP2：210.73.140.16==》主机标识为16=>**000**10000

子网掩码：255.255.255.224（1111 1111, 1111 1111, 1111 1111, **111**0 0000）

IP1与IP2在同一个子网内，他们交换数据不需要经过路由器。

IP3：210.73.60.1的[主机](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA)标识为1=>**000**00001

IP4：210.73.60.252的主机标识为252=>**111**11100

子网掩码：255.255.255.224（1111 1111, 1111 1111, 1111 1111, **111**0 0000）

IP3与IP4不在同一个网络，他们的主机号分别为1和252，他们交换数据经过路由器。

## 子网掩码的表示

1． 通过与IP地址格式相同的[点分十进制](https://baike.baidu.com/item/%E7%82%B9%E5%88%86%E5%8D%81%E8%BF%9B%E5%88%B6)表示

如：255.0.0.0 或255.255.255.128

2． 在IP地址后加上"/"符号以及1-32的数字，其中1-32的数字表示子网掩码中[网络标识](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E6%A0%87%E8%AF%86)位的长度

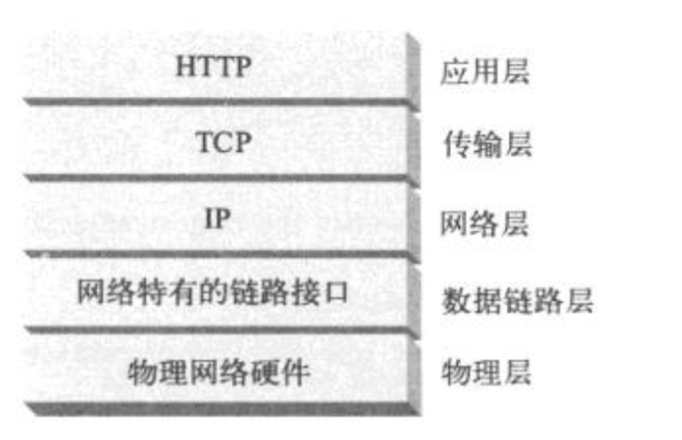
如：192.168.1.1/24 的子网掩码也可以表示为255.255.255.0

子网掩码一般为255.255.255.0

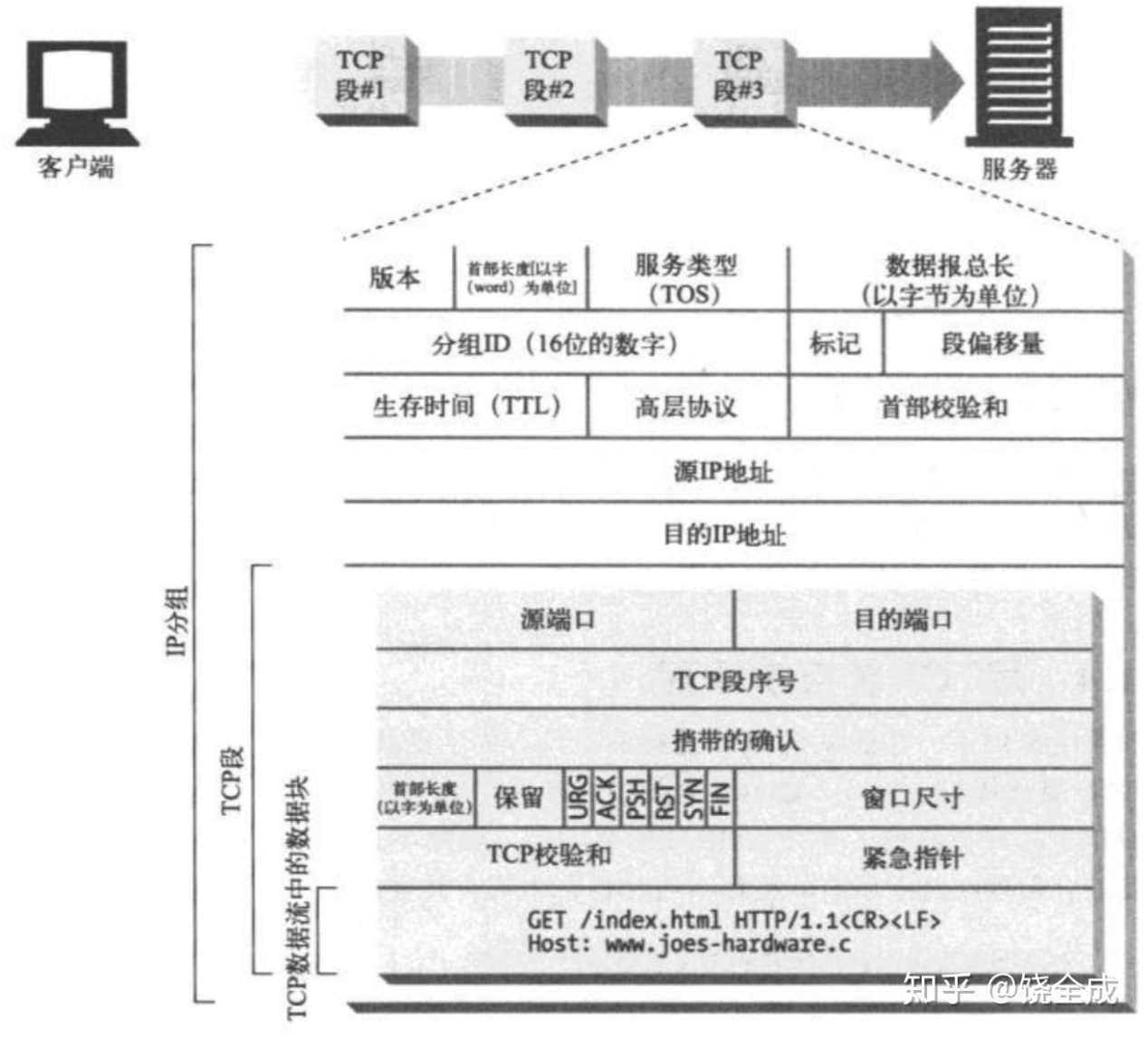
# http

## 整体介绍

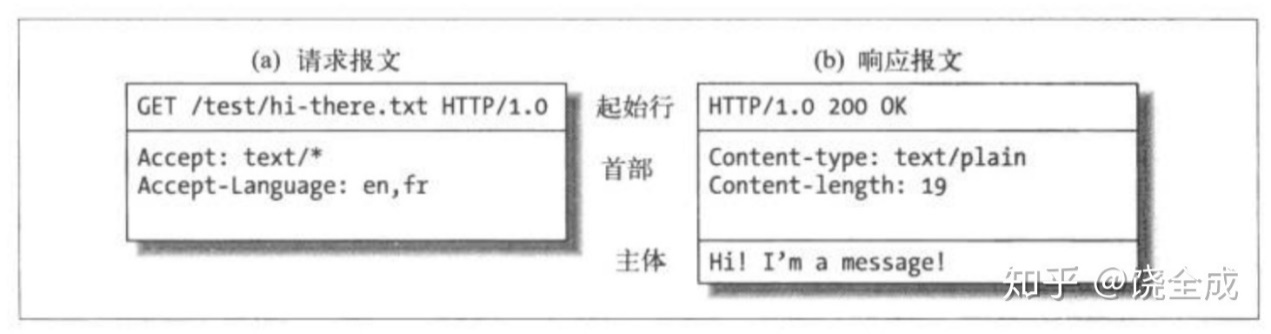
HTTP(Hypertext Transfer Protocol)超文本传输协议，著名的应用是用在浏览器的服务器间的通信。

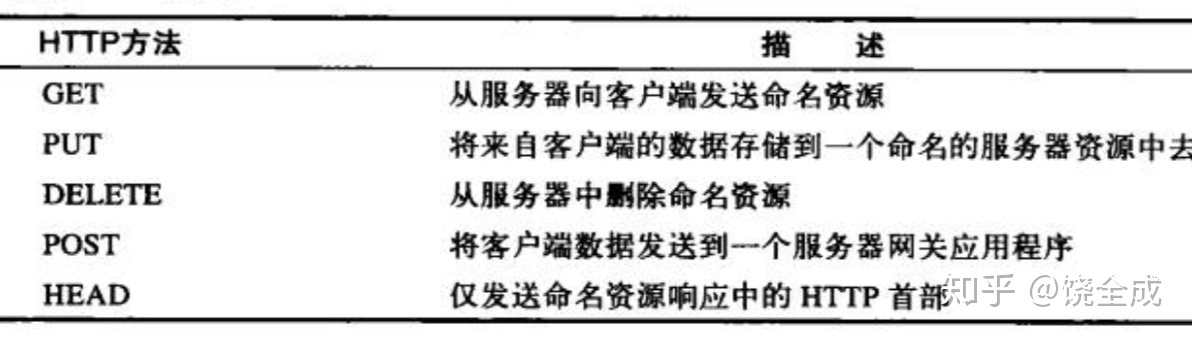


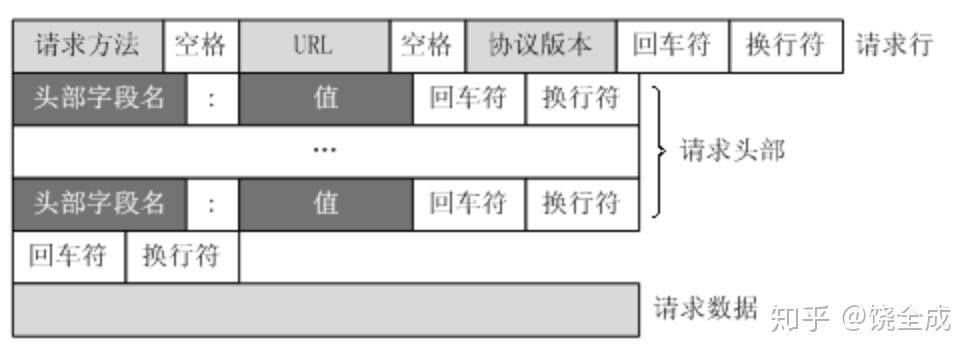
HTTP在传输一段报文时，会以流的形式将报文数据的内容通过一条打开的TCP连接按序传输。TCP接到上层应用交给它的数据流之后，会按序将数据流打散成一个个的分段。再交到IP层，通过网络进行传输。另一端的接收方则相反，它们将接收到的分段按序组装好，交给上层HTTP协议进行处理。

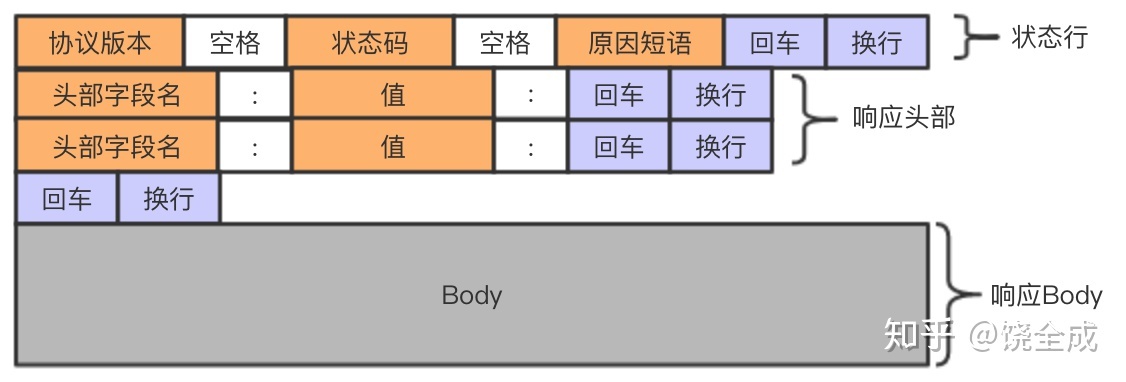


## 报文格式









请求报文、响应报文的起始行和响应头部里的字段都是文本化、结构化的。而请求body却可以包含任意二进制数据（如图片、视频、软件等），当然也可以包含文本。

## 编码

原始的url值

/data?cmd=Fence2Area&meta={"caller":"test","TraceId":"test"}&request={"fence":[{"lng":10.2,"lat":10.2}, {"lng":10.2,"lat":8.2}, {"lng":8.2,"lat":8.2}, {"lng":8.2,"lat":10.2}],"coordtype":2}

编码后的url值：

/data?cmd=Fence2Area&meta={%22caller%22:%22test%22,%22TraceId%22:%22test%22}&request={%22fence%22:[{%22lng%22:10.2,%22lat%22:10.2},%20{%22lng%22:10.2,%22lat%22:8.2},%20{%22lng%22:8.2,%22lat%22:8.2},%20{%22lng%22:8.2,%22lat%22:10.2}],%22coordtype%22:2}

0x22是单引号"的ascii值

## 状态码



200("OK") 一切正常。

404("Not Found")  服务器端不知道客户端要请求哪个资源的情况

410("Gone") 服务器端知道客户端所请求的资源曾经存在，但现在已经不存在了的情况

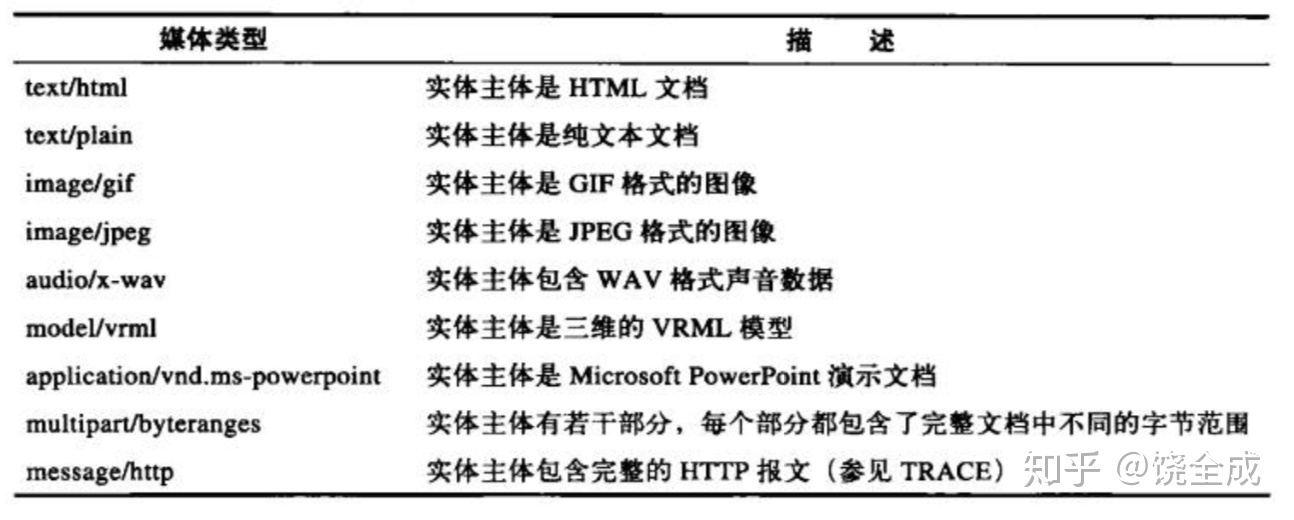
500("Internal Server Error") 服务器方面的问题。

400("Bad Request") 客户端方面的问题。

## 多用途因特网邮件扩展（Multipurpose Internet Media Extension ，MIME）

Content-Type: text/plain; charset=utf-8

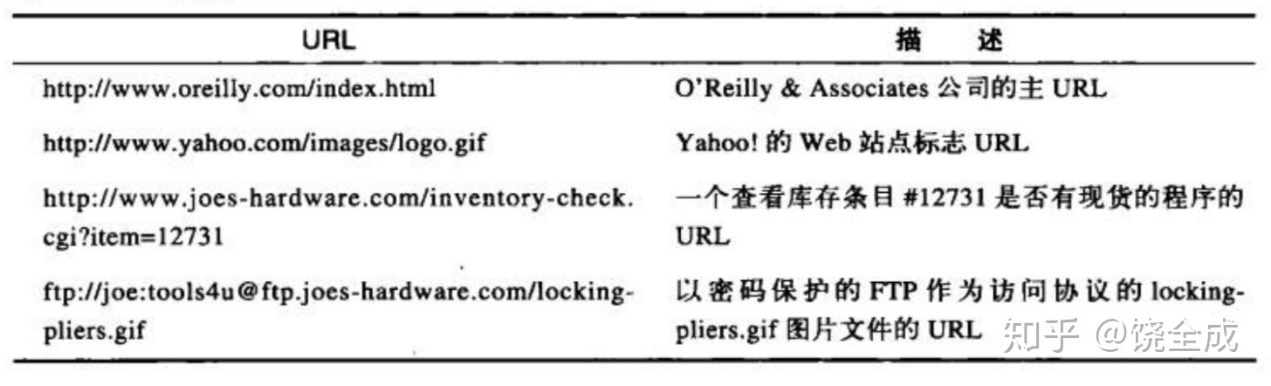
浏览器从服务器上取回了一个对象时，会去查看MIME类型，从而得知如何处理这种对象，是该展示图片，还是调用声卡播放声音



## URI/URL/URN

URI(Uniform Resource Identifier, 统一资源标识符)表示服务器资源，URL(Uniform Resource Locator, 统一资源定位符)和URN(Uniform Resource Name, 统一资源名)是URI的具体实现。URI是一个通用的概念，由两个主要的子集URL和URN构成，URL通过位置、URN通过名字来标识资源。

URL定义了资源的位置，表示资源的实际地址，在使用URL的过程中，如果URL背后的资源发生了位置移动，访问者就找不到它了。这个时候就要用到URN了，它给定资源一个名字，无论它移动到哪里，都可以通过这个名字来访问到它。URL通常的格式是：

协议方案+服务器地址+具体的资源路径

# NAT（以下简要介绍）

## 作用

1、私有地址->共有地址

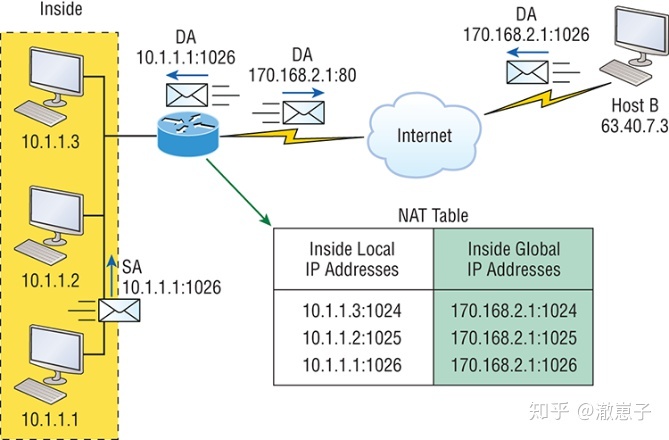
2、屏蔽网络内部的地址

## 分类

1、静态NAT：此类NAT在本地和全局地址之间做一到一的永久映射。须注意静态NAT要求用户对每一台主机都有一个真实的Internet IP地址。

2、动态NAT：允许用户将一个未登记的IP地址映射到一个登记的IP地址池中的一个。采用动态分配的方法将外部合法地址映射到内部网络，无需像静态NAT那样，通过对路由器进行静态配置来将内部地址映射到外部地址，但是必须有足够的真正的IP地址来进行收发包。

3、端口NAT（PAT）：最为流行的NAT配置类型。通过多个源端口，将多个未登记的IP地址映射到一个合法IP地址（多到一）。使用PAT能够使上千个用户仅使用一个全局IP地址连接到Internet。



PAT方式中，所有内部主机都转换为一个IP地址；

端口号帮助路由器识别哪一台主机应当收到返回数据；

PAT允许我们在传输层识别主机，从而理论上一个真实IP地址可被65，000台主机共享。

# 路由

对于LAN，通过**交换机**即可完成以太网帧在链路层上的数据转发。

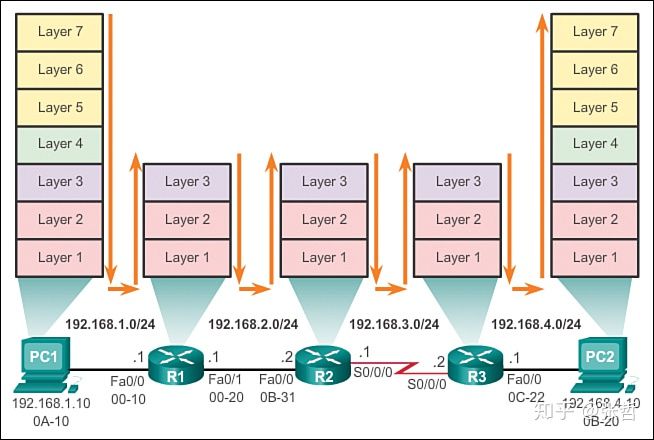
当源和目的IP地址位于不同网络时，以太网帧必须发送给路由器。

**路由器主要执行以下三个步骤：**

1. 将链路层的帧头和帧尾移除，解析出网路层报文。

2. 检查IP报文的目的IP地址，在路由表中查找最佳路由。

3. 如果路由器找到一条最佳路径，则将网络层报文封装到新的链路帧中，并将帧转发到输出端口。



1. 发送报文：

PC 1发送报文给PC 2时，首先必须确定目的IPv4地址是否位于同一网络。

方法：源IP地址与子网掩码按位与，目的IP地址与子网掩码按位与。

同一网络（LAN）：通过查找ARP缓存或ARP广播

不同网络（WAN）：将数据帧发送给默认网关

1. 转发至下一跳

R1从PC 1接收到以太网帧后执行以下步骤：

1. R1检查目的MAC地址，与接收端口FastEthernet 0/0相匹配，因此，将帧复制到buffer。

2. R1识别以太网类型为0x800，意味着以太网帧的数据部分包含IPv4报文。

3. R1解封装该以太网帧。

4. 由于目的IPv4地址与R1直连的任何网络都不相符，R1在路由表中查找包含该目的IPv4地址主机的网络地址。本例中，路由表中有192.168.4.0/24网络的路由。目的IPv4地址为192.168.4.10，即该网络上的主机IPv4地址。

1. 到达目的地

1. R3将数据链路帧复制到它的buffer。

2. R3解封装该数据链路帧。

3. R3在路由表中查找该目的IPv4地址。R3路由表中有直接连接到该网络的路由。这表示报文可直接发送到目的设备而无需发送至路由器

## 路由表

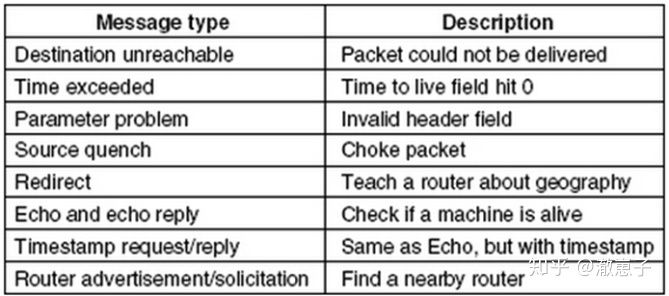
路由表是存储在RAM中的一份数据文件，用于存储直连以及远端网络的路由信息。路由表中包含网络或下一跳地址的信息。这些信息告知路由器可以通过将报文发送至代表下一跳地址的路由器以最佳路径到达目的地址。

# ICMP

ICMP是（Internet Control Message Protocol）Internet控制[报文](https://baike.baidu.com/item/%E6%8A%A5%E6%96%87/3164352)协议

**当路由器端处理报文时如有意外发生，事件通过ICMP报告给发送端。**

ICMP信息封装在IP报文中，最重要的一部分如下表所列：



**DESTINATION UNREACHABLE**消息用于当路由器无法找到目标地址

**TIME EXCEEDED**消息是由于报文TTL（Time to live）计数器到达0时。该事件是报文在回环，或计数器值设置过低的迹象。对于这一错误信息的聪明的应用是**traceroute**工具，traceroute发现从主机到目的IP地址路径上的路由器。它向目的地发送IP包，第一次的时候，将TTL设置为1，引发第一个路由器的Time Exceeded错误。这样，第一个路由器回复ICMP包，从而让出发主机知道途径的第一个路由器的信息。随后TTL被设置为2、3、4，...，直到到达目的主机。这样，沿途的每个路由器都会向出发主机发送ICMP包来汇报错误。traceroute将ICMP包的信息打印在屏幕上，就是接力路径的信息了。这并不是TIME EXCEEDED信息的本意，但却是非常有用的故障排查工具

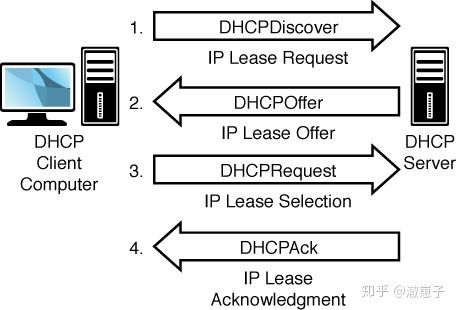
# DHCP

动态主机设置协议（Dynamic Host Configuration Protocol, DHCP）是一个[局域网](http://link.zhihu.com/?target=http%3A//zh.wikipedia.org/wiki/%25E5%25B1%2580%25E5%259F%259F%25E7%25BD%2591)的[网络协议](http://link.zhihu.com/?target=http%3A//zh.wikipedia.org/wiki/%25E7%25BD%2591%25E7%25BB%259C%25E5%258D%258F%25E8%25AE%25AE)，使用[UDP](http://link.zhihu.com/?target=http%3A//zh.wikipedia.org/wiki/UDP)协议工作，主要有两个用途：

1、给内部网络或网络服务供应商自动分配[IP](http://link.zhihu.com/?target=http%3A//zh.wikipedia.org/wiki/IP)地址给用户

2、给内部网络管理员作为对所有电脑作中央管理的手段

## 工作原理

DHCP从一个IP地址池中提供IP地址，该池有DHCP服务器数据库定义，称为scope。如果客户端接受这一地址，则它可在一个预定义的期限内使用该地址，称为租约。如果客户端无法从DHCP服务器获取IP地址，它就无法正常初始化TCP/IP。  


**1、客户机请求IP地址（DHCPDISCOVER）:**

当一个IPv4客户机启动时监测到需要IP地址，它会初始化一个TCP/IP的限制版本，之后广播一个报文请求寻找DHCP服务器的地址。该广播报文告知监听服务器客户端需要IP地址信息。DHCP客户端发送的报文这一阶段包括租约请求，客户端源地址，0.0.0.0，目的地址，即广播地址255.255.255.255。报文也包括客户端硬件MAC地址和机器名，该信息也指明了向DHCP服务器发起请求的设备。

客户端向DHCP服务器发送请求IP地址的真实报文称为DHCPDISCOVER报文。网络上每一台安装了TCP/IP协议的主机都会接收到这种广播信息，但只有DHCP服务器才会做出响应。

**2、服务器提供IP地址（DHCPOFFER）:**

所有拥有有效IP地址的DHCP服务器都会向DHCP客户端提供IP地址信息。它响应以地址池中一个未分配的IP地址供请求主机使用。要能够响应DHCPDISCOVER报文，DHCP服务器必须拥有客户端的有效IP配置信息。DHCP服务器回复的DHCPOFFER报文包含以下信息：

客户端的硬件地址

提供的IP地址

合适的子网掩码

租约有效期

服务器ID，即DHCP服务器的IP地址

**3、客户机选择IP地址（DHCPREQUEST）：**

DHCP客户端选择它所接收到的第一个DHCPOFFER报文提供的IP地址。之后，它把这一信息广播至网络。该报文中，客户端请求服务器提供给它的IP地址。这是因为客户端可能收到不止一个DHCP服务器发送的offer。通过广播这一请求，客户端告知其他DHCP服务器不会再接受其他offer。为了进一步确保客户端接受的服务器offer没有疑义，DHCPREQUEST报文中还包含以下信息：

提供所接受offer的服务器IP地址

客户端硬件地址

客户端接受的IP地址

**4、服务器确认IP租约（DHCPACK）：**

DHCP服务器对客户端作出响应，将IP地址分配给客户端。之后，它发送DHCPACK确认信息给客户端。该信息包含IP地址的有效租约以及其他配置信息。

有时，在客户端接收服务器提供的租约后，DHCP租约请求仍可能不成功。可能有以下几种情况：

由于客户端移动至其他子网，IP地址无效

客户端尝试租约它之前的IP地址但该IP地址不再可用

在上述情况下，服务器会发送一条不成功信息DHCPNACK。收到DHCPNACK的客户端必须重新开始整个DHCP初始化进程。也就是说，它必须发送另一个DHCPDISCOVER报文查找新的IP地址。