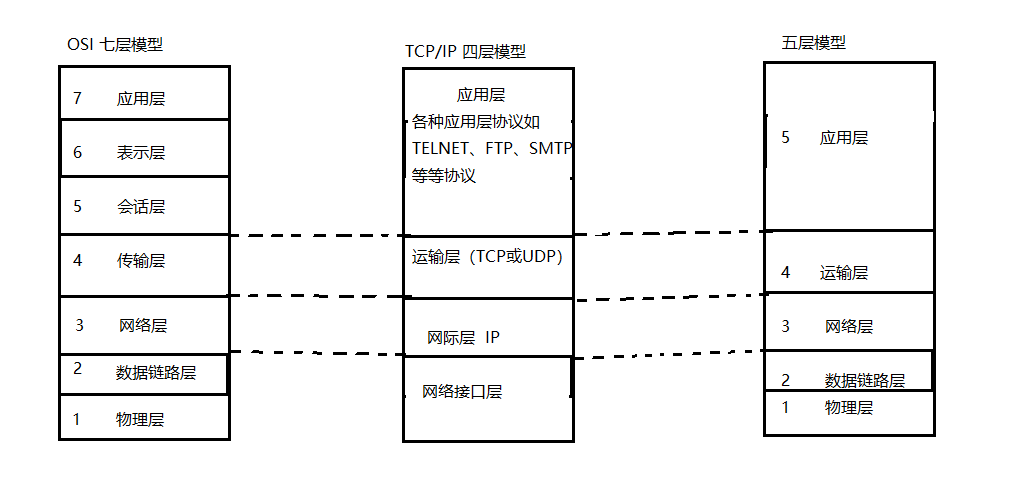
一、网络分层



OSI 七层模型：

1、 物理层：主要定义物理标准，如网线的接口类型、光纤的接口类型、各种介质的传输速率等。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，

到达目的地后在转化为 1、0，也就是我们常说的数模转换与模数转换）。这一层的数据叫比特。

2、数据链路层：定义了如何让格式化数据以数据帧为单位进行传输，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。

（即 封装成帧、透明传输、差错检测）。（将比特组装成帧和点到点的传递）

3、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个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。即 网络层负责数据包从源到宿的传递和网际互连（主机之间的通信）

4、传输层：定义了一些传输数据的协议和端口(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，到达目的地址后再进行重组。常常把这一层数据叫做段。传输层，提供端到端的可靠报文传递和错误恢复（应用进程之间的通信）

5、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收会话请求。（设备之间需要相互认识可以是 IP 也

可以是 MAC 或者是主机名）。即，会话层：建立、管理和终止会话（会话协议数据单元SPDU）

6、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 程序员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。即，**表示**

**层**：对数据进行翻译、加密和压缩（表示协议数据单元PPDU）

7、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。

二、典型协议

**2.1 常见协议**

应用层： 常见的协议有 HTTP，FTP 协议、SMTP

传输层： 常见的协议有 TCP/UDP 协议

网络层： 常见的协议有 IP 协议，ICMP 协议，IGMP 协议、ARP（地址解析）协议

（1）**TCP 传输控制协议**（TransmissionControl 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。

TCP通过校验和进行差错校验，通过序列号、确认和超时重传机制实现可靠传输，通过滑动窗口实现流量控制，通过慢开始和拥塞避免、快重传和快恢复实现拥塞控制。

一般用于文件传输（FTP HTTP 对数据准确性要求高，速度可以相对慢），发送或接收邮件（POP IMAP SMTP 对数据准确性要求高，非紧急应用），远程登录（TELNET SSH 对数据准确性有一定要求，有连接的概念）等等。

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

传输数据之前源端和终端不建立连接，当它想传送时就简单地去抓取来自应用程序的数据，并尽可能快地把它扔到网络上。在发送端，UDP传送数据的速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制；在接收端，UDP把每个消息段放在队列中，应用程序每次从队列中读一个消息段。

UDP一般用于即时通信（QQ聊天 对数据准确性和丢包要求比较低，但速度必须快），在线视频（RTSP 速度一定要快，保证视频连续，但是偶尔花了一个图像帧，人们还是能接受的），网络语音电话（VoIP 语音数据包一般比较小，需要高速发送，偶尔断音或串音也没有问题），NFS(网络文件系统)、SNMP(简单网络管理协议)、DNS(主域名称系统)、TFTP(通用文件传输协议)等等。

（3）**HTTP 超文本传输协议**（HyperText Transfer Protocol）是互联网上应用最为广泛的一种协议。



HTTP协议永远都是客户端发起请求，服务器回送响应。HTTP协议是一个无状态的协议，同一个客户端的这次请求和上次请求没有对应关系。

一次HTTP操作称为一个事务，其工作过程可分为四步：

1）首先客户机与服务器需要建立连接（建立的是TCP连接，不是HTTP连接，因为HTTP是无连接的）。（只要单击某个超级链接，HTTP的工作开始）具体来说如下：

① 地址解析

如客户端浏览器请求这个页面：[http://localhost.com:8080/index.htm](http://localhost:8080/simple.htm)l

从中分解出协议名、主机名、端口、对象路径等部分，对于我们的这个地址，解析得到的结果如下：

协议名：http

主机名：localhost.com

端口：8080

对象路径：/index.htm

在这一步，需要域名解析系统DNS解析域名 localhost.com 得到主机的IP地址。

域名解析的过程：首先搜索浏览器自身的DNS缓存，如果自身的缓存中有对应的条目，而且没有过期，则解析到此结束；如果浏览器自身的缓存里面没有找到对应的条目，那么浏览器会搜索操作系统自身的DNS缓存；如果在Windows系统的DNS缓存也没有找到，那么尝试读取hosts文件看看这里面有没有该域名对应的IP地址，如果有则解析成功；如果在hosts文件中也没有找到对应的条目，浏览器就会发起一个DNS的系统调用，就会向本地配置的首选DNS服务器发起域名解析请求。

② 封装HTTP请求数据包

把以上部分结合本机自己的信息，封装成一个HTTP请求数据包。

报文的组成：

（1）开始行：在请求报文中叫请求行（包括方法、请求资源的URL、HTTP版本），在响应报文中叫状态行（HTTP版本、状态码、状态码的简单短语）。

其中http版本有：

**http 0.9**：HTTP 0.9是第一个版本的HTTP协议，已过时。它的组成极其简单，只允许客户端发送GET这一种请求，且不支持请求头。由于没有协议头，造成了HTTP 0.9协议只支持一种内容，即纯文本。

**http 1.0**：HTTP协议的第二个版本，第一个在通讯中指定版本号的HTTP协议版本，至今仍被广泛采用。

**http 1.1** ：HTTP协议的第三个版本是HTTP 1.1，是目前使用最广泛的协议版本 。（和1.0 的主要区别就是是否是持久连接，1.0一个tcp连接只传输一个Web对象；1.1默认使用持久连接，当然也可以配置成使用非持久连接）

**http 2.0**：HTTP 2.0是下一代 HTTP 协议，目前应用还非常少。

HTTP 2.0最大的特点: 不会改动HTTP 的语义，HTTP 方法、状态码、URI 及首部字段这些核心概念没变，却能致力于突破上一代标准的性能限制，改进传输性能，实现低延迟和高吞吐量。而之所以叫2.0，是在于新增的二进制分帧层。同时还可以让服务器将响应主动“推送”到客户端缓存中，以此来避免往返的延迟。

HTTP 2.0 会将所有传输的信息分割为更小的消息和帧,并对它们采用二进制格式的编码，这些帧可以乱序发送，然后再根据每个帧首部的流标识符重新组装。

（2）首部行，说明浏览器，服务器或者报文主体的一些信息。

（3）实体主体，一般不使用。

③ 封装成TCP包，建立TCP连接（HTTP请求报文作为三次握手的第三个报文的数据发送给服务器）。

2）TCP连接建立后，客户机发送一个http请求给服务器，请求方式的格式为：统一资源标识符（URL）、协议版本号，后边是MIME信息包括请求修饰符、客户机信息和可能的内容。

3）服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是MIME信息包括服务器信息、实体信息和可能的内容。

4）客户端接收服务器所返回的信息通过浏览器显示在用户的显示屏上，然后客户机与服务器断开连接。

如果在以上过程中的某一步出现错误，那么产生错误的信息将返回到客户端，由显示屏输出。对于用户来说，这些过程是由HTTP自己完成的，用户只要用鼠标点击，等待信息显示就可以了。

**（4）ip 地址的分类和划分子网**

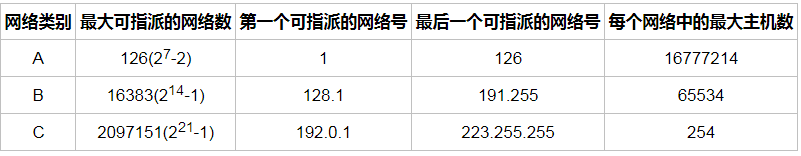
分类的IP地址，是一两级的IP地址（32位），其格式为：

IP地址 = {<网络号>，<主机号>}

**A类地址**网络号字段占一个字节，只有7位可用（第一位已固定为0），故A类网络有（2^7 - 2）126个（全0表示本网络，全1（即0111 1111 ）即网络号为127保留为本地软件环回测试本主机进程之间的通信）。每一个A类网络中最大主机数为：2^24 - 2 【1：全0的主机号表示本主机；2：全1的主机号表示**本网络内全部主机】。**

**B类地址**网络号字段有2个字节，前两位固定为10，不存在全0或全1，**然而，实际中128.0.0.0是不指派的。**所以B类地址网络号数：2^14-1=16383，同样的主机数2^16-2 =65534。

**C类地址**，前面固定110，同样的实际192.0.0.0是不指派的，2^21-1 = 2097151，主机数2^8-2 =254。



**划分子网：**

子网的划分就是将两级IP增加到三级IP，它是将IP地址的主机号借用作为子网号，这样增加了网络数，同时也减少了主机数。

子网掩码是用来和IP地址进行与运算计算得出子网地址。

如果一个网络不划分子网，那么该网络就使用默认的子网掩码。默认子网掩码中1的位置和IP地址中的网络号对应。划分子网后的子网掩码中1的位置和网络号+子网号的位置对应。

**（5）二进制协议 VS 文本协议**

二进制协议一般消息头固定和消息体变长 ，每个字段固定了含义 ，其特点如下：

* 1. 解析效率超高，几乎没有解析代价（优点）；
  2. 没有冗余字段，体积小（优点）；
  3. 可读性差，难于调试（缺点）；
  4. 扩展性不好 ，如果要扩展字段，旧版协议就不兼容了（缺点）；



文本协议（一般是由一串ACSII字符组成的数据）：

* 1. 可读性好，便于调试；
  2. 扩展性也好，方便兼容旧协议；
  3. 解析效率一般，需要进行字符串比对；
  4. 存在冗余字段，体积大；

http 2.0之前的是文本协议，2.0采用的二进制协议。

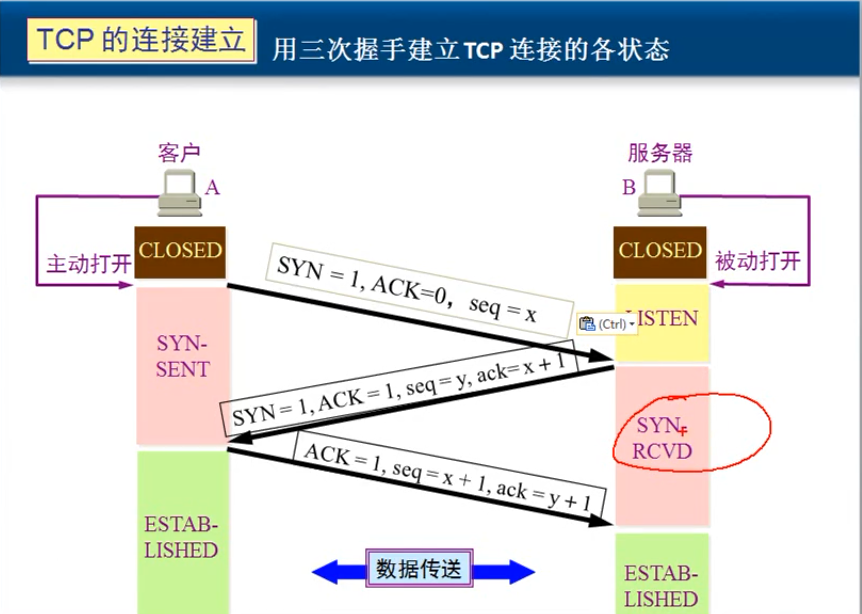
**2.2 TCP协议原理**

TCP每发送一个报文段，就启动一个定时器，如果在定时器超时之后还没有收到ACK确认，就重传该报文。 如图所示，数据包由A的缓冲区发往B，B在收到数据包以后，回发一个ACK确认包给A，之后A将该数据包从缓冲区释放。因此，该数据包会一直缓存在A的缓冲区，直到一个ACK确认为止。



在TCP/IP协议中，TCP协议提供可靠的面向连接的服务；三次握手（建立连接）和四次挥手（关闭连接）；使用**滑动窗口机制**进行流量控制；

**2.2.1 三次握手**



TCP首部的字段：

确认**ACK**：仅当ACK=1时确认号（ack）才有用，当ACK=0时确认号无效。TCP规定，连接建立后的ACK都置为1。

同步**SYN**：连接建立时用来同步序号。当SYN=1，ACK=0时表明这是一个请求报文段。响应的报文段中SYN=1，ACK=1。SYN=1表明这是一个连接请求或者连接接受阶段。

终止**FIN**：释放一个连接。当FIN=1时表明此报文段的数据已经发送完毕，要求释放运输连接。

（1）第一次握手：建立连接时，客户端A发送SYN包[SYN=1,ACK=0,seq=x]（随机选择一个初始序号seq=x）到服务器B（SYN=1的报文段不携带数据，但消耗掉一个序号），并进入SYN\_SENT状态（同步已发送），等待服务器B确认。

（2）第二次握手：服务器B收到SYN包，必须确认客户A的SYN，同时自己也发送一个SYN包，即SYN+ACK包[SYN=1,ACK=1,seq=y,ack=x+1]，此时服务器B进入SYN\_RECV状态（同步收到）。

（3）第三次握手：客户端A收到服务器B的SYN＋ACK包，向服务器B发送确认包ACK[ACK=1,seq=x+1,ack=y+1]，此包发送完毕，客户端A和服务器B进入ESTABLISHED状态，完成三次握手。

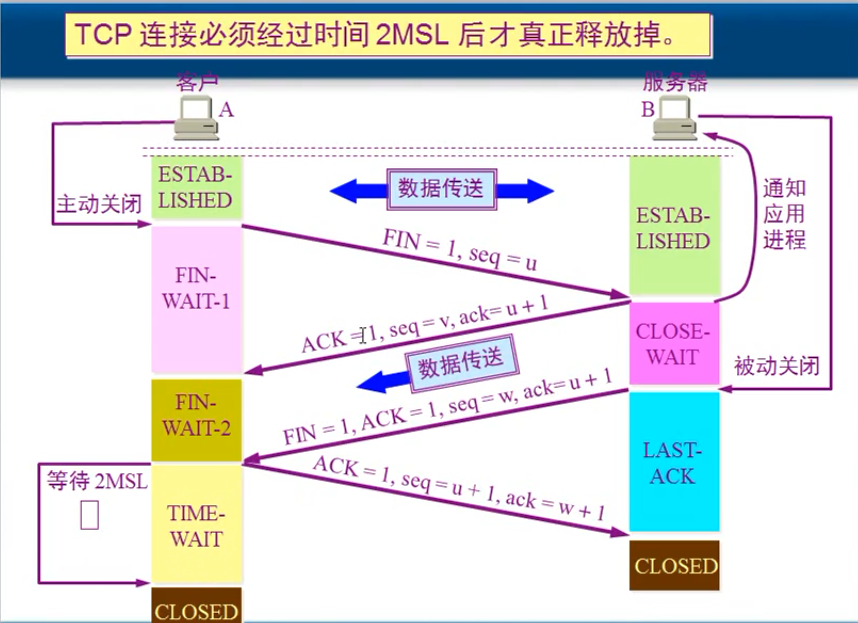
完成三次握手，客户端与服务器开始传送数据。在socket编程中，**客户端执行connect()时，将会触发三次握手**。

**为什么需要三次握手呢？**

主要是为了防止已经失效的连接请求报文段突然又到达了B，因而产生错误。

B认为A要建立连接，所以就向A发送确认报文段建立了连接，但是A没有建立连接的请求，因此不会理睬B的确认，也不会向B发送数据，但是B却一直在等待，造成资源的浪费。

**2.2.2 四次挥手**



客户端或服务器均可主动发起挥手动作，在socket编程中，任何一方执行close()操作即可产生挥手操作。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。这个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。TCP要保证在所有可能的情况下使得所有的数据都能够被投递，当你关闭一个socket时，主动关闭一端的socket将进入TIME\_WAIT状态，而被动关闭一方则转入CLOSED状态 ，这能够保证所有的数据都被传输。

（1）首先A B端的TCP进程都处于established状态， 当A的应用程序传送完报文段，就会去主动关闭连接。A会停止发送报文段（但是还会接收），并向B发送[FIN = 1,seq=u]（u等于前面已经发送过的数据的最后一个字节加一）数据，之后进入FIN-WAIT-1状态。

（2）B接收到A发送的请求之后，会通知应用进程，A已经不再发送数据，同时B会向A发送ACK确认数据[ACK=1,seq=v,ack=u+1 ]，B进入CLOSE-WAIT状态（关闭等待），A接收到B发送的数据之后，A进入FIN-WAIT-2状态；此时A到B方的连接已经关闭了（即半连接状态）。

（3）当B的应用进程发现自己也没有数据需要传送，B应用进程就会发出被动关闭的请求，B此时向A发送[FIN=1,ACK=1,seq=w,ack=u+1]数据，并且进入LAST-ACK状态（最后确认）。

（4）A接收到B发送的数据之后，向B发送ACK确认数据[ACK =1,seq=u+1,ack=w+1]，进入TIME-WAIT状态，等待2MSL（Maximum Segment Lifetime，报文最大生存时间）之后正常关闭连接进入CLOSED状态；B接收到A发送的确认之后进入CLOSED状态。B到A方的连接关闭！至此，TCP连接才真正全部关闭！

**2.2.3 time\_wait 状态存在2MSL的原因**

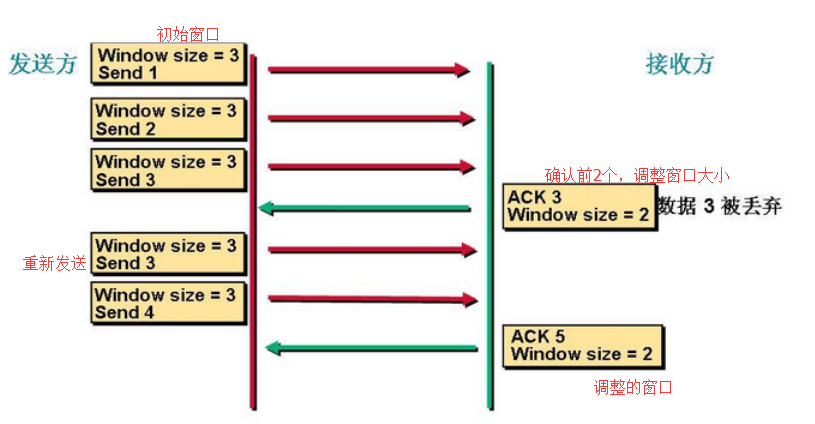
（1）为实现TCP这种全双工（full-duplex）连接的可靠释放

主动关闭连接发送的ACK（4次交互的最后一个包）在网络中丢失，那么由于TCP的重传机制，执行passiveclose的一方需要重发其FIN，在该FIN到达client（client是active close发起方）之前，client必须维护这条连接的状态（尽管它已调用过close），具体而言，就是这条TCP连接对应的（local\_ip, local\_port）资源不能被立即释放或重新分配。直到被动关闭的一方重发的FIN达到，client也重发ACK后，该TCP连接才能恢复初始的CLOSED状态。如果activeclose方不进入TIME\_WAIT以维护其连接状态，则当passive close方重发的FIN达到时，active close方的TCP传输层会以RST包响应对方，这会被对方认为有错误发生（而事实上，这是正常的关闭连接过程，并非异常）。

（2）为使旧的数据包在网络因过期而消失

主动关闭的一方主动调用close后，此时的TCP连接进入TIME\_WAIT状态，处于该状态下的TCP连接不能立即以同样的四元组（源IP，源端口，目的IP，目的端口）建立新连接，即发起active close的那方占用的local port在TIME\_WAIT期间不能再被重新分配。由于TIME\_WAIT状态持续时间为2MSL，这样保证了旧TCP连接双工链路中的旧数据包均因过期而消失，此后，就可以用相同的四元组建立一条新连接而不会发生前后两次连接数据错乱的情况。

**2.2.4 流量控制（滑动窗口实现）**



流量控制就是让发送方发送速率不要太快，让接收方来得及接收。这是**点对点**的通信量的控制。

滑动窗口协议，是TCP使用的一种流量控制方法。发送方的发送窗口不能超过接收方给出的接受窗口的大小。

**2.2.5 拥塞控制**

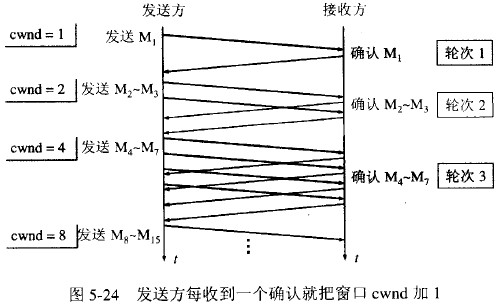
拥塞控制就是防止过多的数据注入到网络中，可以使网络中的路由器和链路不致过载。这是一个**全局性**的过程，涉及到所有的主机。路由器等等。

**（1）慢开始和拥塞避免**

发送方维持一个拥塞窗口 cwnd ( congestion window )的状态变量。拥塞窗口的大小取决于网络的拥塞程度，并且动态地在变化。发送方让自己的发送窗口等于拥塞窗口。

发送方控制拥塞窗口的原则是：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。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。

**慢开始算法**：当主机开始发送数据时，如果立即把大量数据字节注入到网络，那么就有可能引起网络拥塞，因为现在并不清楚网络的负荷情况。因此，较好的方法是 先探测一下，即由小到大逐渐增大发送窗口，也就是说，由小到大逐渐增大拥塞窗口数值。通常在刚刚开始发送报文段时，先把拥塞窗口 cwnd 设置为一个最大报文段MSS的数值。而在每收到一个对新的报文段的确认后（一个传输轮次），把拥塞窗口增加一倍。用这样的方法逐步增大发送方的拥塞窗口 cwnd ，可以使分组注入到网络的速率更加合理。



为了防止拥塞窗口cwnd增长过大引起网络拥塞，还需要设置一个慢开始门限 ssthresh 状态变量。慢开始门限ssthresh的用法如下：

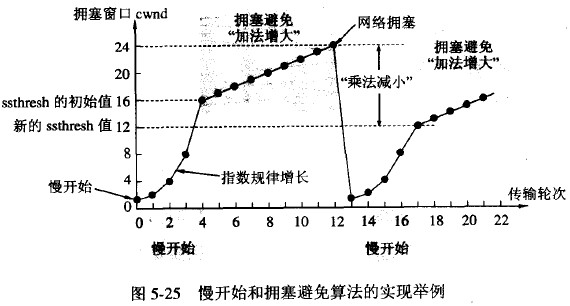
当 cwnd < ssthresh 时，使用上述的慢开始算法。

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。

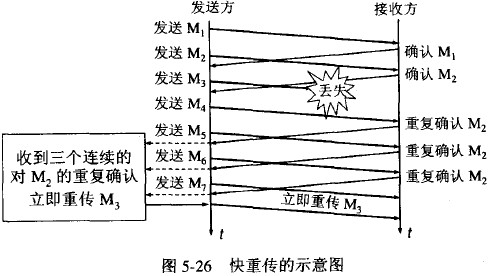
**拥塞避免算法**：让拥塞窗口cwnd缓慢地增大，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1，而不是加倍。这样拥塞窗口cwnd按线性规律缓慢增长，比慢开始算法的拥塞窗口增长速率缓慢得多。

无论在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞（其根据就是没有收到确认），就要把慢开始门限ssthresh设置为出现拥塞时的发送方窗口值的一半（但不能小于2）。然后把拥塞窗口cwnd重新设置为1，执行慢开始算法。这样做的目的就是要迅速减少主机发送到网络中的分组数，使得发生拥塞的路由器有足够时间把队列中积压的分组处理完毕。



**（2）快重传和快恢复**

**快重传：**算法首先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（为的是使发送方及早知道有报文段没有到达对方）而不要等到自己发送数据时才进行捎带确认。

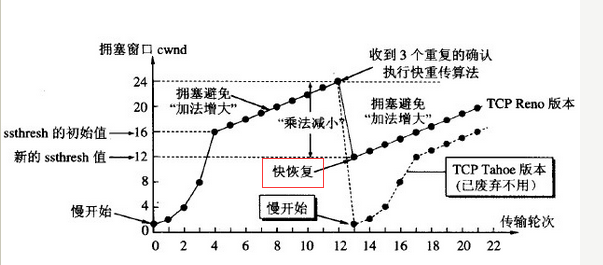


快重传算法还规定，发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段M3，而不必继续等待M3设置的重传计时器到期。由于发送方尽早重传未被确认的报文段，因此采用快重传后可以使整个网络吞吐量提高约20%。

**快恢复算法**，其过程有以下两个要点：

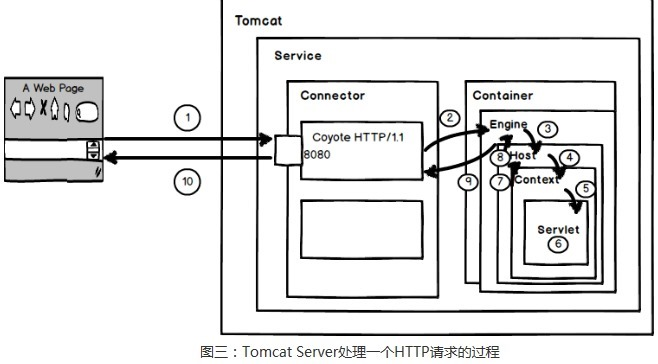
（1）当发送方连续收到三个重复确认，就执行“乘法减小”算法，把慢开始门限ssthresh减半。这是为了预防网络发生拥塞。请注意：接下去不执行慢开始算法。

（2） 由于发送方现在认为网络很可能没有发生拥塞，因此与慢开始不同之处是现在不执行慢开始算法（即拥塞窗口cwnd现在不设置为1），而是把cwnd值设置为 慢开始门限ssthresh减半后的数值，然后开始执行拥塞避免算法（“加法增大”），使拥塞窗口缓慢地线性增大。



三、应用

**1、Tomcat 处理一个HTTP请求的过程**



(1) browser 请求 url 被发送到本机端口默认为80，被在那里侦听的Coyote HTTP/1.1 Connector获得。

(2) Connector 把该请求交给它所在的 Service 的 Engine来处理，并等待来自Engine的回应 。

(3) Engine获得请求url，匹配它所拥有的所有虚拟主机Host。

(4) Engine匹配到名为severName的Host（即使匹配不到也把请求交给localhost Host处理，因为该Host被定义为该Engine的默认主机）

(5) Host 获得请求 url，匹配它所拥有的所有Context。

(6) Host 匹配到路径前缀相同的Context（如果匹配不到就把该请求交给路径名为”“的Context去处理）

(7)Context 获得请求 url，在它的mapping table中寻找对应的servlet

(8) Context匹配到URL PATTERN为\*.jsp的servlet，对应于JspServlet类

(9) 构造HttpServletRequest对象和HttpServletResponse对象，作为参数调用Servlet的doGet或doPost方法

(10)Context把执行完了之后的HttpServletResponse对象返回给Host

(11)Host把HttpServletResponse对象返回给Engine

(12)Engine把HttpServletResponse对象返回给Connector

(13)Connector把HttpServletResponse对象返回给客户browser

**2. HTTP/1.1 协议中的八种请求方法**

**2.1 get 和 post 的区别**

GET和POST是HTTP协议中的两种发送请求的方法，而HTTP的底层是TCP/IP，所以GET和POST的底层也是TCP/IP。GET和POST本质上就是TCP链接，并无差别。但是由于HTTP的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同。

|  |  |  |
| --- | --- | --- |
|  | GET | POST |
| 数据位置 | 参数通过URL传递 | 放在Request body中 |
| 缓存 | 请求可被主动缓存 | 请求不会被缓存（除非主动设置） |
| 数据长度 | GET方式提交的数据最多只能是1024字节 | 无限制 |
| 可见性/安全性 | 数据在 URL 中对所有人都是可见的 | 数据不会显示在 URL 中 |
| 数据类型 | 只允许 ASCII 字符 | 没有限制，也允许二进制数据 |
| 数据包的个数 | GET产生一个TCP数据包（浏览器会把http header和data一并发送出去 ） | POST产生两个TCP数据包（浏览器先发送header，服务器响应，浏览器再发送data ） |

**2.2 其他方法**

（1）PUT

向指定资源位置上传其最新内容。

（2）DELETE

请求服务器删除 request-url所标识的资源。

（3）OPTIONS

返回服务器针对特定资源所支持的HTTP请求方法。

（4）HEAD

向服务器索要与get请求相一致的响应，但是响应体将不会被返回。

（5）TRACE

回显服务器收到的请求，主要用于测试和诊断。

（6）CONNECT

预留给能够将连接改为管道方式的代理服务器。

**3. TCP和UDP的区别**

（1）[TCP](https://www.baidu.com/s?wd=TCP&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)提供的是面向连接的、可靠的数据流传输；[UDP](https://www.baidu.com/s?wd=UDP&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)提供的是非面向连接的、不可靠的数据流传输。

（2）[TCP](https://www.baidu.com/s?wd=TCP&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)提供可靠的服务，通过TCP连接传送的数据，无差错、不丢失，不重复，按序到达；[UDP](https://www.baidu.com/s?wd=UDP&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)尽最大努力交付，即不保证可靠交付。

（3）TCP面向字节流；UDP面向报文。

（4）TCP连接只能是点到点的；UDP支持一对一、一对多、多对一和多对多的交互通信。

（5）TCP首部开销20字节；UDP的首部开销小，只有8个字节。

（6）TCP的逻辑通信信道是全双工的可靠信道；UDP的逻辑通信信道是不可靠信道。

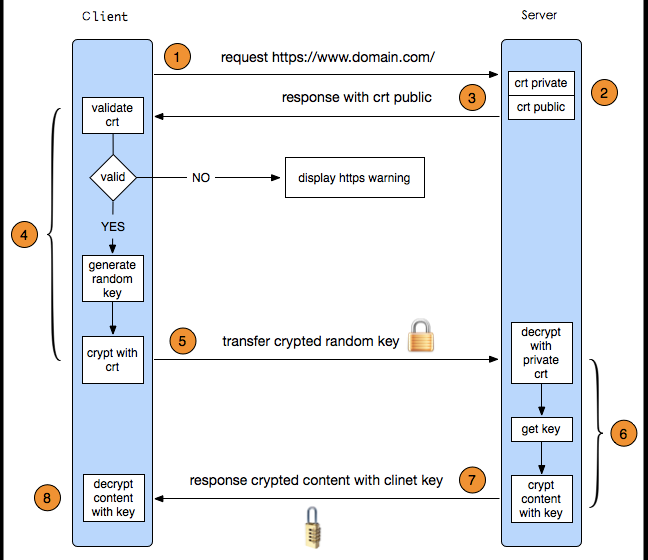


**4. https过程详解**

**4.1 加密的过程**

https简单的说就是安全版的http，因为http协议的数据都是明文进行传输的，所以对于一些敏感信息的传输就很不安全，为了安全传输敏感数据，在http的基础上添加了一个安全传输层，对所有的数据都加密后再进行传输，客户端和服务器端收到加密数据后按照之前约定好的秘钥解密。

https要使客户端与服务器端的通信过程得到安全保证，必须使用对称加密算法，但是协商对称加密算法的过程，需要使用非对称加密算法来保证安全，然而直接使用非对称加密的过程本身也不安全，会有中间人篡改公钥的可能性，所以客户端与服务器不直接使用公钥，而是使用数字证书签发机构颁发的证书来保证非对称加密过程本身的安全。这样通过这些机制协商出一个对称加密算法，就此双方使用该算法进行加密解密，从而解决了客户端与服务器端之间的通信安全问题。



（1）客户端先向服务器发出加密通信的请求（协议版本、random1、加密方法、压缩方法）。

（2）服务器收到请求,然后响应（协议版本、random2、加密方法、服务器证书）。

（3）客户端收到证书之后会首先会进行验证，如果没有验证通过，就会显示不安全的提示。验证通过之后，客户端会生成一个随机数 pre-master secret，然后使用证书中的公钥进行加密，然后传递给服务器端。

（4）服务器收到使用公钥加密的内容，在服务器端使用私钥解密（https中公钥和私钥是互为加密解密的。公钥是大家都知道的，你发送的数据是用公钥加密的，如果公钥能够解密的话，https也就不具备安全性了）之后获得随机数pre-master secret，然后根据radom1、radom2、pre-master secret通过一定的算法得出session Key和MAC算法秘钥，作为后面交互过程中使用对称秘钥。同时客户端也会使用radom1、radom2、pre-master secret，和同样的算法生成session Key和MAC算法的秘钥。

（5）然后在后续的交互中就使用session Key和MAC算法的秘钥对传输的内容进行加密和解密。

**4.2 抓包的数据安全性**

https抓包的原理就是抓包程序将服务器返回的证书截获，然后给客户端返回一个它自己的证书，客户端发送的数据抓包程序用自己的证书解密，然后再用截获的证书加密，再发给服务器，所以抓包程序看到的是明文。

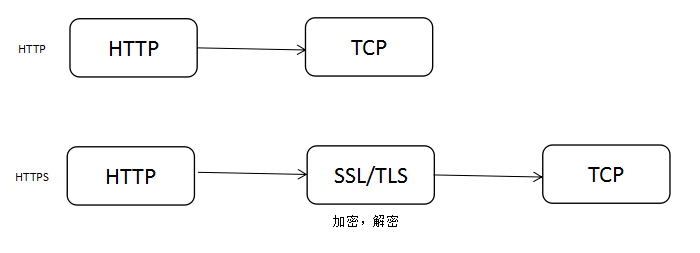
**4.3 HTTPS 与 HTTP 的一些区别**

（1）HTTPS协议需要到CA申请证书，一般免费证书很少，需要交费。

（2）HTTP协议运行在TCP之上，所有传输的内容都是明文，HTTPS运行在SSL/TLS之上，SSL/TLS运行在TCP之上，所有传输的内容都经过加密的。

（3）HTTP和HTTPS使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

（4）HTTPS可以有效的防止运营商劫持，解决了防劫持的一个大问题。



**5. cookie 和 session 的区别和联系**

理论上一个用户的所有请求操作都应该属于同一个会话，Web应用程序是使用HTTP协议进行传输数据的，HTTP协议是无状态的协议。一旦数据交换完毕，客户端与服务器端的连接就会关闭，再次交换数据需要建立新的连接，这就意味着服务器无法从连接上跟踪会话，为了弥补HTTP协议无状态的不足，引入了cookie机制进行会话跟踪。

客户端请求服务器，如果服务器需要记录该用户状态，就使用response向客户端浏览器颁发一个Cookie。客户端浏览器会把Cookie保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务器。服务器检查该Cookie，以此来辨认用户状态。

**cookie里面保存的信息：**

名称：一个唯一确定cookie的名称。cookie的名称必须是经过URL编码的。

值：储存在cookie中的字符串值。值必须被URL编码。

域：cookie对于哪个域是有效的。

路径：对于这个指定域中的路径才会向服务器发送cookie。

失效时间：表示cookie何时应该被删除的时间（何时应该停止向服务器发送这个cookie）。默认情况下，浏览器会话结束时即将所有cookie删除；不过也可以自己设置删除时间。

安全标志：指定后，cookie只有在使用SSL连接的时候才发送到服务器。

Cookie 与 Session 都可以进行会话跟踪，主要作用都是在整个会话过程中，实现数据的跨页面访问，但是实现的原理不太一样。

**区别：**

（1）cookie在客户端记录信息确定用户身份，session在服务器端记录信息确定用户身份。

（2）Cookie是服务器发给客户端的特殊信息，cookie是以文本的方式保存在客户端，每次请求时都带上它。

（3）Session是服务器端使用的一种记录客户端状态的机制，使用上比Cookie简单一些，相应的也增加了服务器的存储压力。。

（4）cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗，考虑到安全应当使用session。

（5）session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能，考虑到减轻服务器性能方面，应当使用COOKIE。

**联系：**

（1）cookie和session都是用来跟踪浏览器用户身份的会话方式。

（2）session 的运行依赖 session id，而 session id 是存在 cookie 中的，如果浏览器禁用了 cookie ，同时 session 也会失效（但是可以通过其它方式实现，比如在 url 中传递 session\_id）。

Session是保存在服务器端的，每个用户都会产生一个Session。如果并发访问的用户非常多，会产生非常多的Session，消耗大量的内存。 而Cookie保存在客户端，不占用服务器资源。如果并发浏览的用户非常多，Cookie是很好的选择。

**6. http常见的状态码**

**1xx——通知信息**，如请求收到了或正在处理。

**2xx——成功**，成功被服务器接收、理解、并接受。

200 服务器成功返回网页

201 请求成功并且服务器创建了新的资源。

202 服务器已接受请求，但尚未处理

**3xx——重定向**，客户端需要采取进一步的操作才能完成请求。

301状态码：被请求的资源已永久移动到新位置，服务器返回此响应时会自动将请求者转到新位置。

302状态码：请求的资源临时从不同的URI响应请求，但请求者应继续使用原有位置来进行以后的请求。

304自从上次请求后，请求的网页未修改过。

**4xx——请求错误**。

401状态码：请求要求身份验证。 对于需要登录的网页，服务器可能返回此响应。

403状态码：服务器已经理解请求，但是拒绝执行它。

404状态码：请求失败，请求所希望得到的资源未被在服务器上发现。

**5xx——服务器的差错**，服务器在处理请求的过程中有错误或者异常状态发生。

500状态码：服务器遇到了一个未曾预料的状况，导致了它无法完成对请求的处理。

503状态码：由于临时的服务器维护或者过载，服务器当前无法处理请求。通常，这个是暂时状态，一段时间会恢复。

**7. HTTP协议的区别**

**7.1 HTTP1.0和HTTP1.1的一些区别**

**（1）长连接**

HTTP 1.1支持长连接（PersistentConnection）和请求的流水线处理，在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟，一定程度上弥补了HTTP 1.0每次请求都要创建连接的缺点。

**（2）错误通知的管理**

在HTTP1.1中新增了24个错误状态响应码，如409（Conflict）表示请求的资源与资源的当前状态发生冲突；410（Gone）表示服务器上的某个资源被永久性的删除。

**（3）带宽优化及网络连接的使用**

HTTP1.0中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能，HTTP1.1则在请求头引入了range头域，它允许只请求资源的某个部分，即返回码是206（Partial Content），这样就方便了开发者自由的选择以便于充分利用带宽和连接。

**（4）Host头处理**

在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名（hostname）。但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机，并且它们共享一个IP地址。HTTP1.1的请求消息和响应消息都应支持Host头域，且请求消息中如果没有Host头域会报告一个错误（400 Bad Request）。

**（5）缓存处理**

在HTTP1.0中主要使用header里的If-Modified-Since,Expires来做为缓存判断的标准，HTTP1.1则引入了更多的缓存控制策略例如Entity tag，If-Unmodified-Since, If-Match, If-None-Match等更多可供选择的缓存头来控制缓存策略。

**7.2 HTTP2.0 和 HTTP1.X 相比的新特性**

HTTP 2.0最大的特点: 不会改动HTTP 的语义，HTTP 方法、状态码、URI 及首部字段这些核心概念没变，却能致力于突破上一代标准的性能限制，改进传输性能，实现低延迟和高吞吐量。

**（1）新的二进制格式**（Binary Format）

而之所以叫2.0，是在于新增的二进制分帧层。HTTP 2.0 会将所有传输的信息分割为更小的消息和帧，并对它们采用二进制格式的编码，这些帧可以乱序发送，然后再根据每个帧首部的流标识符重新组装。

**（2）多路复用**（MultiPlexing）

即连接共享，一个request对应一个id，这样一个连接上可以有多个request，每个连接的request可以随机的混杂在一起，接收方可以根据request的 id将request再归属到各自不同的服务端请求里面。

**（3）header压缩**

HTTP1.x的header带有大量信息，而且每次都要重复发送，HTTP2.0使用encoder来减少需要传输的header大小，通讯双方各自cache一份header fields表，既避免了重复header的传输，又减小了需要传输的大小。

**（4）服务端推送**（server push）

同时还可以让服务器将响应主动“推送”到客户端缓存中，以此来避免往返的延迟。