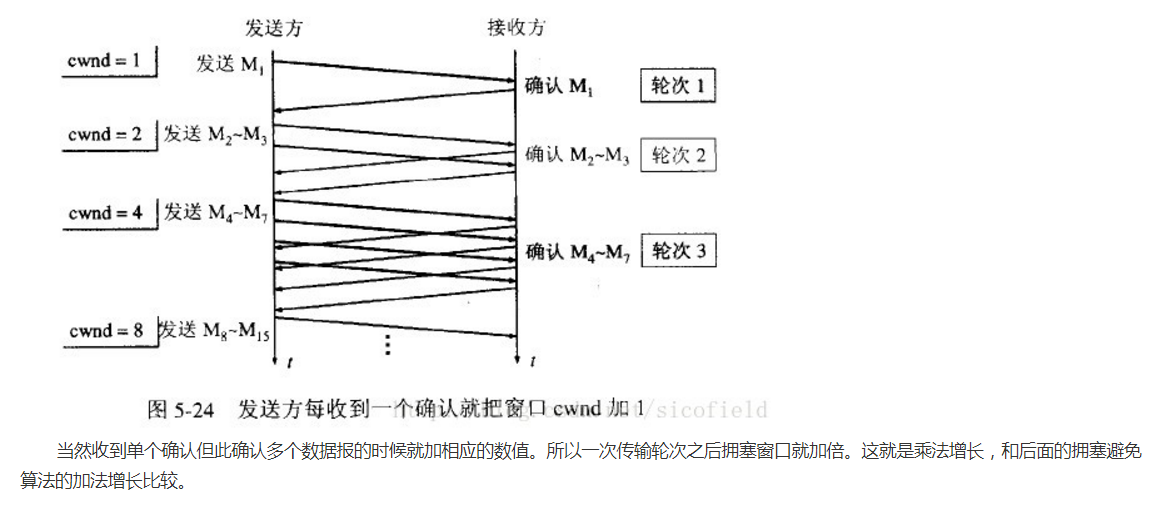
HTTP使用TCP而不是UDP的原因在于（打开）一个网页必须传送很多数据，而TCP协议提供传输控制，按顺序组织数据，和错误纠正。HTTP协议的瓶颈及其优化技巧都是基于TCP协议本身的特性。如TCP建立连接时三次握手有1.5个RTT（round-trip time）的延迟，为了避免每次请求的都经历握手带来的延迟，应用层会选择不同策略的http长链接方案。又如TCP在建立连接的初期有慢启动（slow start）的特性，所以连接的重用总是比新建连接性能要好。

拥塞控制就是防止过多的数据注入网络中，这样可以使网络中的路由器或链路不致过载。

发送方维持一个叫做**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，另外考虑到接受方的接收能力，发送窗口可能小于拥塞窗口。

       慢开始算法的思路就是，不要一开始就发送大量的数据，先探测一下网络的拥塞程度，也就是说由小到大逐渐增加拥塞窗口的大小。

       这里用报文段的个数的拥塞窗口大小举例说明慢开始算法，实时拥塞窗口大小是以字节为单位的。如下图：



当然收到单个确认但此确认多个数据报的时候就加相应的数值。所以一次传输轮次之后拥塞窗口就加倍。这就是乘法增长，和后面的拥塞避免算法的加法增长比较。

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

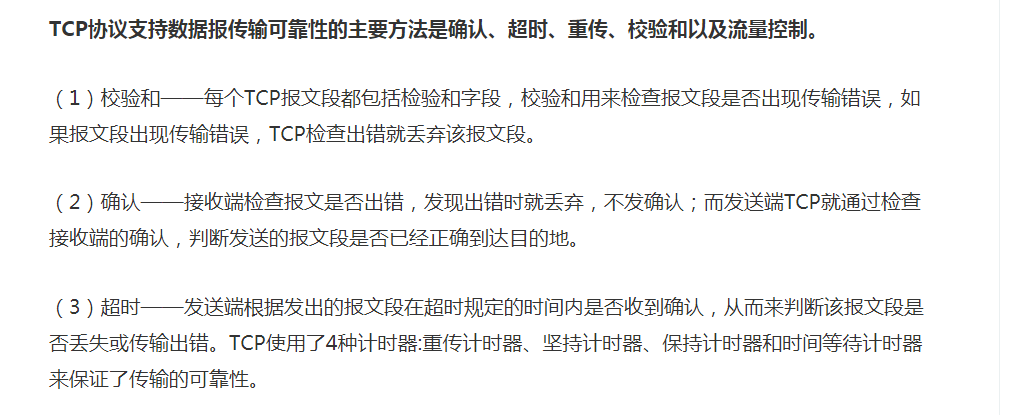
**当cwnd<ssthresh时，使用慢开始算法。**

**当cwnd>ssthresh时，改用拥塞避免算法。**

**当cwnd=ssthresh时，慢开始与拥塞避免算法任意。**

       拥塞避免算法让拥塞窗口缓慢增长，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。这样拥塞窗口按线性规律缓慢增长。

       无论是在**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，虽然没有收到确认可能是其他原因的分组丢失，但是因为无法判定，所以都当做拥塞来处理），就把慢开始门限设置为出现拥塞时的发送窗口大小的一半。然后把拥塞窗口设置为1，执行慢开始算法。



## HTTP1.0

HTTP 协议老的标准是HTTP/1.0，为了提高系统的效率，HTTP 1.0规定浏览器与服务器只保持短暂的连接，浏览器的每次请求都需要与服务器建立一个TCP连接，服务器完成请求处理后立即断开TCP连接，服务器不跟踪每个客户也不记录过去的请求。

## HTTP1.1

为了克服HTTP 1.0的这个缺陷，HTTP 1.1支持持久连接（HTTP/1.1的默认模式使用带流水线的持久连接），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。在http1.1，request和reponse头中都有可能出现一个connection的头，此header的含义是当client和server通信时对于长链接如何进行处理。  
在http1.1中，client和server都是默认对方支持长链接的， 如果client使用http1.1协议，但又不希望使用长链接，则需要在header中指明connection的值为close，使用的话，指明为：keep-alive  
  
HTTP 1.1通过增加更多的请求头和响应头来改进和扩充HTTP 1.0的功能。如，HTTP 1.0不支持Host请求头字段，WEB浏览器无法使用主机头名来明确表示要访问服务器上的哪个WEB站点，这样就无法使用WEB服务器在同一个IP地址和端口号上配置多个虚拟WEB站点。在HTTP 1.1中增加Host请求头字段后，WEB浏览器可以使用主机头名来明确表示要访问服务器上的哪个WEB站点，这才实现了在一台WEB服务器上可以在同一个IP地址和端口号上使用不同的主机名来创建多个虚拟WEB站点。

GET POST**方**法：在浏览器的地址栏中输入网址的方式访问网页时，浏览器采用GET方法向服务器获取资源，POST方法要求被请求服务器接受附在请求后面的数据，常用于提交表单。**GET是用于获取数据的，POST一般用于将数据发给服务器之用**。

**多路复用 (Multiplexing)**

多路复用允许同时通过单一的 HTTP/2 连接发起多重的请求-响应消息。在 HTTP/1.1 协议中浏览器客户端在同一时间，针对同一域名下的请求有一定数量限制。超过限制数目的请求会被阻塞。这也是为何一些站点会有多个静态资源 CDN 域名的原因之一，拿 Twitter 为例，http://twimg.com，目的就是变相的解决浏览器针对同一域名的请求限制阻塞问题。而 HTTP/2 的多路复用(Multiplexing) 则允许同时通过单一的 HTTP/2 连接发起多重的请求-响应消息。因此 HTTP/2 可以很容易的去实现多流并行而不用依赖建立多个 TCP 连接，HTTP/2 把 HTTP 协议通信的基本单位缩小为一个一个的帧，这些帧对应着逻辑流中的消息。并行地在同一个 TCP 连接上双向交换消息。

**首部压缩（Header Compression）**

HTTP/1.1并不支持 HTTP 首部压缩，为此 SPDY 和 HTTP/2 应运而生， SPDY 使用的是通用的DEFLATE 算法，而 HTTP/2 则使用了专门为首部压缩而设计的 HPACK 算法。随着 Web 功能越来越复杂，每个页面产生的请求数也越来越多，根据 HTTP Archive 的统计，当前平均每个页面都会产生上百个请求。越来越多的请求导致消耗在头部的流量越来越多，尤其是每次都要传输 UserAgent、Cookie 这类不会频繁变动的内容，完全是一种浪费。

**服务端推送（Server Push）**

服务端推送是一种在客户端请求之前发送数据的机制。在 HTTP/2 中，服务器可以对客户端的一个请求发送多个响应。Server Push 让 HTTP1.x 时代使用内嵌资源的优化手段变得没有意义；如果一个请求是由你的主页发起的，服务器很可能会响应主页内容、logo 以及样式表，因为它知道客户端会用到这些东西。这相当于在一个 HTML 文档内集合了所有的资源，不过与之相比，服务器推送还有一个很大的优势：可以缓存！也让在遵循同源的情况下，不同页面之间可以共享缓存资源成为可能。

#### HTTPS

HTTP协议传输的数据都是未加密的，也就是明文的，因此使用HTTP协议传输隐私信息非常不安全。为了保证这些隐私数据能加密传输，于是网景公司设计了SSL（Secure Sockets Layer）协议用于对HTTP协议传输的数据进行加密，从而就诞生了HTTPS。现在的HTTPS都是用的TLS协议，但是由于SSL出现的时间比较早，并且依旧被现在浏览器所支持，因此SSL依然是HTTPS的代名词。

HTTPS在传输数据之前需要客户端（浏览器）与服务端（网站）之间进行一次握手，在握手过程中将确立双方加密传输数据的密码信息。TLS/SSL协议不仅仅是一套加密传输的协议，TLS/SSL中使用了非对称加密，对称加密以及HASH算法。握手过程的简单描述如下：

1.浏览器将自己支持的一套加密规则发送给网站。

2.网站从中选出一组加密算法与HASH算法，并将自己的身份信息以证书的形式发回给浏览器。证书里面包含了网站地址，加密公钥，以及证书的颁发机构等信息。

3.获得网站证书之后浏览器要做以下工作：  
a) 验证证书的合法性（颁发证书的机构是否合法，证书中包含的网站地址是否与正在访问的地址一致等），如果证书受信任，则浏览器栏里面会显示一个小锁头，否则会给出证书不受信的提示。  
b) 如果证书受信任，或者是用户接受了不受信的证书，浏览器会生成一串随机数的密码，并用证书中提供的公钥加密。  
c) 使用约定好的HASH计算握手消息，并使用生成的随机数对消息进行加密，最后将之前生成的所有信息发送给网站。

4.网站接收浏览器发来的数据之后要做以下的操作：  
a) 使用自己的私钥将信息解密取出密码，使用密码解密浏览器发来的握手消息，并验证HASH是否与浏览器发来的一致。  
b) 使用密码加密一段握手消息，发送给浏览器。

5.浏览器解密并计算握手消息的HASH，如果与服务端发来的HASH一致，此时握手过程结束，之后所有的通信数据将由之前浏览器生成的随机密码并利用对称加密算法进行加密。

这里浏览器与网站互相发送加密的握手消息并验证，目的是为了保证双方都获得了一致的密码，并且可以正常的加密解密数据。其中非对称加密算法用于在握手过程中加密生成的密码，对称加密算法用于对真正传输的数据进行加密，而HASH算法用于验证数据的完整性。由于浏览器生成的密码是整个数据加密的关键，因此在传输的时候使用了非对称加密算法对其加密。非对称加密算法会生成公钥和私钥，公钥只能用于加密数据，因此可以随意传输，而网站的私钥用于对数据进行解密，所以网站都会非常小心的保管自己的私钥，防止泄漏。

TLS握手过程中如果有任何错误，都会使加密连接断开，从而阻止了隐私信息的传输。正是由于HTTPS非常的安全，攻击者无法从中找到下手的地方，于是更多的是采用了假证书的手法来欺骗客户端，从而获取明文的信息。默认HTTP的端口号为80，HTTPS的端口号为443。

1xx（临时响应）   
表示临时响应并需要请求者继续执行操作的状态代码。

代码   说明   
100   （继续） 请求者应当继续提出请求。 服务器返回此代码表示已收到请求的第一部分，正在等待其余部分。    
101   （切换协议） 请求者已要求服务器切换协议，服务器已确认并准备切换。

2xx （成功）   
表示成功处理了请求的状态代码。

代码   说明   
200   （成功）  服务器已成功处理了请求。 通常，这表示服务器提供了请求的网页。   
201   （已创建）  请求成功并且服务器创建了新的资源。   
202   （已接受）  服务器已接受请求，但尚未处理。   
203   （非授权信息）  服务器已成功处理了请求，但返回的信息可能来自另一来源。   
204   （无内容）  服务器成功处理了请求，但没有返回任何内容。   
205   （重置内容） 服务器成功处理了请求，但没有返回任何内容。   
206   （部分内容）  服务器成功处理了部分 GET 请求。

3xx （重定向）   
表示要完成请求，需要进一步操作。 通常，这些状态代码用来重定向。

代码   说明   
300   （多种选择）  针对请求，服务器可执行多种操作。 服务器可根据请求者 (user agent) 选择一项操作，或提供操作列表供请求者选择。   
301   （永久移动）  请求的网页已永久移动到新位置。 服务器返回此响应（对 GET 或 HEAD 请求的响应）时，会自动将请求者转到新位置。   
302   （临时移动）  服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。   
303   （查看其他位置） 请求者应当对不同的位置使用单独的 GET 请求来检索响应时，服务器返回此代码。   
304   （未修改） 自从上次请求后，请求的网页未修改过。 服务器返回此响应时，不会返回网页内容。   
305   （使用代理） 请求者只能使用代理访问请求的网页。 如果服务器返回此响应，还表示请求者应使用代理。   
307   （临时重定向）  服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。

4xx（请求错误）   
这些状态代码表示请求可能出错，妨碍了服务器的处理。

代码   说明   
400   （错误请求） 服务器不理解请求的语法。   
401   （未授权） 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   （禁止） 服务器拒绝请求。   
404   （未找到） 服务器找不到请求的网页。   
405   （方法禁用） 禁用请求中指定的方法。   
406   （不接受） 无法使用请求的内容特性响应请求的网页。   
407   （需要代理授权） 此状态代码与 401（未授权）类似，但指定请求者应当授权使用代理。   
408   （请求超时）  服务器等候请求时发生超时。   
409   （冲突）  服务器在完成请求时发生冲突。 服务器必须在响应中包含有关冲突的信息。   
410   （已删除）  如果请求的资源已永久删除，服务器就会返回此响应。   
411   （需要有效长度） 服务器不接受不含有效内容长度标头字段的请求。   
412   （未满足前提条件） 服务器未满足请求者在请求中设置的其中一个前提条件。   
413   （请求实体过大） 服务器无法处理请求，因为请求实体过大，超出服务器的处理能力。   
414   （请求的 URI 过长） 请求的 URI（通常为网址）过长，服务器无法处理。   
415   （不支持的媒体类型） 请求的格式不受请求页面的支持。   
416   （请求范围不符合要求） 如果页面无法提供请求的范围，则服务器会返回此状态代码。   
417   （未满足期望值） 服务器未满足”期望”请求标头字段的要求。

5xx（服务器错误）   
这些状态代码表示服务器在尝试处理请求时发生内部错误。 这些错误可能是服务器本身的错误，而不是请求出错。

代码   说明   
500   （服务器内部错误）  服务器遇到错误，无法完成请求。   
501   （尚未实施） 服务器不具备完成请求的功能。 例如，服务器无法识别请求方法时可能会返回此代码。   
502   （错误网关） 服务器作为网关或代理，从上游服务器收到无效响应。   
503   （服务不可用） 服务器目前无法使用（由于超载或停机维护）。 通常，这只是暂时状态。   
504   （网关超时）  服务器作为网关或代理，但是没有及时从上游服务器收到请求。   
505   （HTTP 版本不受支持） 服务器不支持请求中所用的 HTTP 协议版本。