目录

[1. OSI七层模型和TCP/IP四层模型 1](#_Toc460360431)

[ TCP/IP的四层模型 1](#_Toc460360432)

[ OSI(Open System Interconnection)七层模型 1](#_Toc460360433)

[2. UDP和TCP的区别 2](#_Toc460360434)

[3. TCP三次握手 2](#_Toc460360435)

[4. TCP三次握手有哪些漏洞，有没有被攻击的可能，怎么被攻击? 2](#_Toc460360436)

[5. 三次握手，为什么需要最后一次握手呢？ 3](#_Toc460360437)

[6. 连接释放时：为什么是两次请求呢？四次握手 3](#_Toc460360438)

[7. A为什么等待2MSL，从TIME\_WAIT到CLOSE？ 4](#_Toc460360439)

[8. ARP是什么？ 4](#_Toc460360440)

[9. 各种协议 5](#_Toc460360441)

[10. RARP 5](#_Toc460360442)

[11. HTTP长连接和短连接 5](#_Toc460360443)

[长连接和短连接 5](#_Toc460360444)

[什么时候用长连接和短连接？ 6](#_Toc460360445)

[12. 从输入网址到获得页面的过程 6](#_Toc460360446)

[13. 基本的Http流程 7](#_Toc460360447)

[14. Socket编程 7](#_Toc460360448)

[15. Http与Https的区别 7](#_Toc460360449)

1. OSI七层模型和TCP/IP四层模型

OSI：Open System Interconnection 开放系统互连。七层网络模型称为开放式网络互连参考模型；

TCP/IP：Transmission Control Protocol / Internet Protocol 传输控制协议/因特网互连协议。

是最基本的Internet协议，由网络层的IP和传输层的TCP构成。

|  |  |  |
| --- | --- | --- |
| OSI七层模型 | TCP/IP四层模型 | 对应网络协议 |
| 应用层 | 应用层 | TFTP、FTP、NFS、WAIS |
| 表示层 | Telnet、SNMP |
| 会话层 |  | SMTP、DNS |
| 传输层 | 传输层 | TCP、UDP |
| 网络层 | 网际层 | IP、ICMP、ARP、RIP |
| 数据链路层 | 网络接口层 | FDDI、Ethernet |
| 物理层 | IEEE802.1、IEEE802.2到802.11 |

* TCP/IP的四层模型

     应用层：应用程序间沟通的层，如简单电子邮件传输（SMTP）、文件传输协议（FTP）、网络远程访问协议（Telnet）等。

     传输层：在此层中，它提供了节点间的数据传送服务，如传输控制协议（TCP）、用户数据报协议（UDP）等，TCP和UDP给数据包加入传输数据并把它传输到下一层中，这一层负责传送、确定数据已被送达并接收。

互连网络层：负责提供基本的数据封包传送功能，让每一块数据包都能够到达目的主机（但不检查是否被正确接收），如网际协议（IP）。

     网络接口层：对实际的网络媒体的管理，定义如何使用实际网络（如Ethernet、Serial Line等）来传送数据。

* OSI(Open System Interconnection)七层模型

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）。这一层的数据叫做比特。

数据链路层：定义了如何让格式化数据以进行传输，以及如何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。

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](http://baike.baidu.com/view/468464.htm)，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，到达目的地址后再进行重组。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。

1. UDP和TCP的区别

UDP(User Datagram Protocol):提供无连接的通信，不能保证数据包被发送到目标地址，典型的即时传输少量数据的应用程序通常使用UDP。

TCP全双工的：面向连接的（三次握手）面向字节流的，有拥塞控制(根据网络的好坏控制报文发送的速率)。面向连接的、可靠的、基于字节流的应用程序提供连接定向和可靠的通信。

① TCP面向连接的传输控制协议，而UDP是没有面向连接的数据报服务；

② TCP高可靠性，确保传输数据的正确性，不会出现丢失或乱序；UDP在传输数据前不建立连接，不对数据报进行检查与修改，无需等待对方的回答，所以会出现分组丢失，重复，乱序，应用程序需要负责传输可靠性方面的所有工作；

③ TCP对系统资源要求多，UDP对系统资源要求少；

④ UDP具有较好的实时性，工作效率较TCP高；

⑤ UDP的段结构比TCP段结构简单，因此网络开销小。

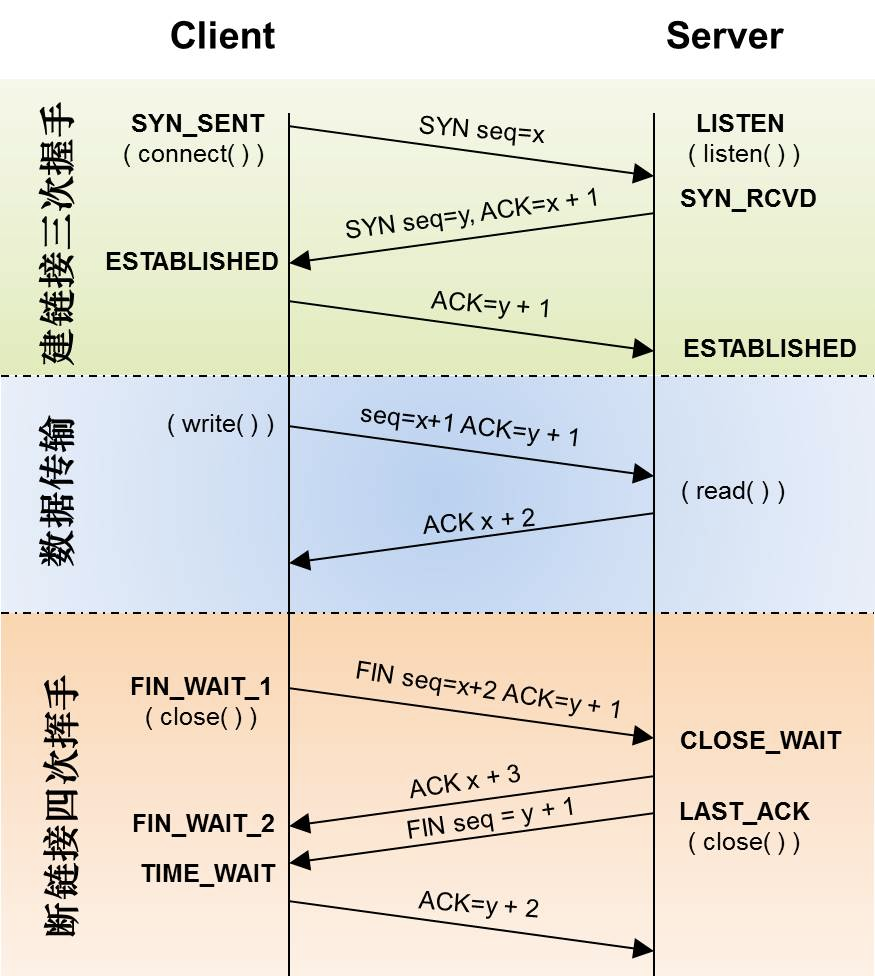
1. TCP三次握手

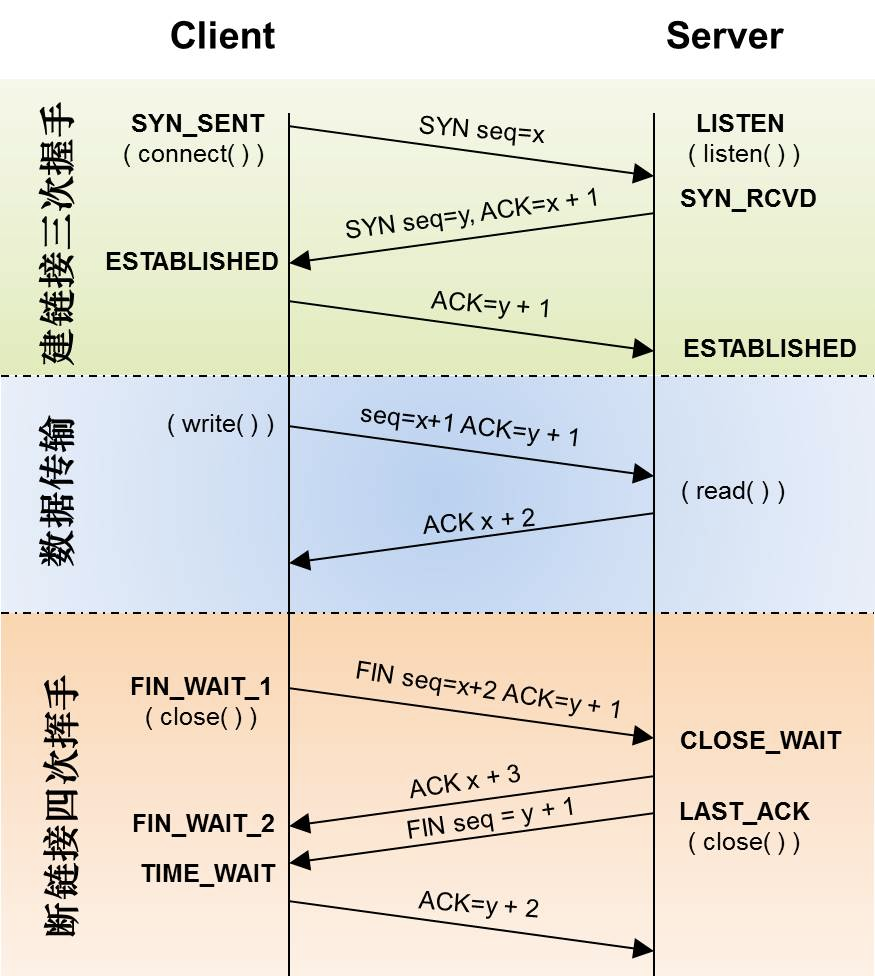
 序列号seq， 序列号seq，确认ACK，同步SYN，终止FIN

第一次握手：主机A发送位码为syn＝1,随机产生seq number=1234567的数据包到服务器，主机B由SYN=1知道，A要求建立联机；

第二次握手：主机B收到请求后要确认联机信息，向A发送ack number=(主机A的seq+1),syn=1,ACK=1,随机产生seq=7654321的包。

第三次握手：主机A收到后检查ack number是否正确，即第一次发送的seq number+1,以及位码ACK是否为1，若正确，主机A会再发送ack number=(主机B的seq+1),ACK=1，主机B收到后确认seq值与ACK=1则连接建立成功。

完成三次握手，主机A与主机B开始传送数据。



1. TCP三次握手有哪些漏洞，有没有被攻击的可能，怎么被攻击?

答：是有漏洞的，主要是如果收到一个恶意攻击的ip一直请求连接，然后服务器会发送ack确认，但是永远等不到回复，就会导致服务器的NIC(网络适配器)，内存，cpu占用率超载，这种攻击方式叫做基于TCP半开回话的洪水攻击。

DDos攻击：

Step1：客户端向服务端发送连接请求数据包；

Step2：服务端向客户端回复连接请求数据包，然后服务器等待客户端发送tcp/ip连接的第三步数据包；

Step3：如果客户端不向服务端发送最后一个数据包，则服务器必须等待30s到2min的时间才能将连接关闭。

当大量的请求只进行到第二步的时候，而不进行第三步，服务器大量的资源都变成了等待第三个数据包，则造成DDos攻击。

预防DDos的方法：（没有根治的方法，除非不用TCP/IP连接）

①严格限制对外开放的服务器的向外访问；

②确保服务器的系统文件是最新版本，并及时更新系统补丁；

③关闭不必要的服务；

④限制同时打开SYN的半连接数目；

⑤缩短SYN半连接的time out时间；

⑥正确设置防火墙；

⑦禁止对主机的非开放服务的访问；

⑧限制特定IP短地址的访问；

⑨启用防火墙的防DDos的属性.

1. 三次握手，为什么需要最后一次握手呢？

TCP三次握手的过程是：假设主机A为客户端，主机B为服务端。

主机A向B发送连接请求，主机B对收到的主机A的报文段进行确认，主机A再次对主机B的确认进行确认。

采用三次握手是为了防止失效的连接请求报文字段突然又传送到主机B，因而产生错误。已失效的连接请求报文段的产生在这样一种情况下：client发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达server。本来这是一个早已失效的报文段。但server收到此失效的连接请求报文段后，就误认为是client再次发出的一个新的连接请求。于是就向client发出确认报文段，同意建立连接。假设不采用三次握手，那么只要server发出确认，新的连接就建立了。由于现在client并没有发出建立连接的请求，因此不会理睬server的确认，也不会向server发送数据。但server却以为新的运输连接已经建立，并一直等待client发来数据。这样，server的很多资源就白白浪费掉了。采用三次握手的办法可以防止上述现象发生。

1. 连接释放时：为什么是两次请求呢？四次握手

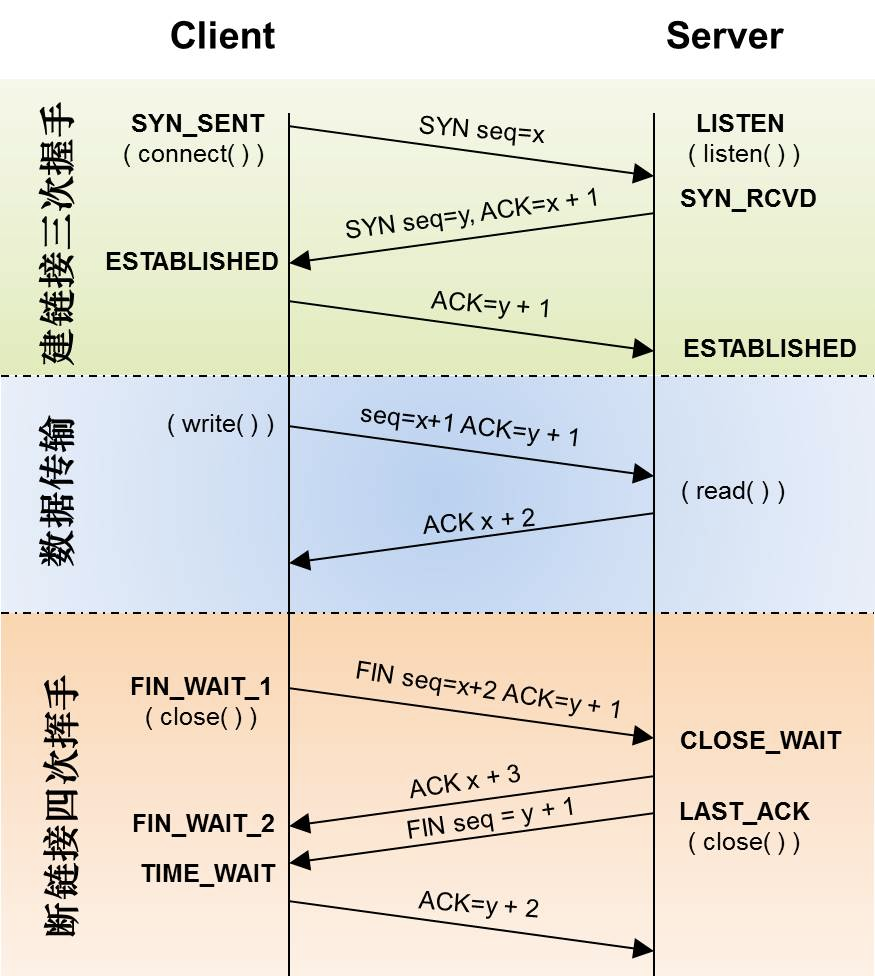
由于TCP连接是全双工的，因此每个方向上都必须单独进行关闭。

主机A发送关闭请求，主机B确认；主机B发送关闭请求，主机A确认。

① 当主机A确认发送完数据且知道B已经接收完了，想要关闭发送数据口（当然确认信号还是可以发），就会发FIN给主机B。

② 主机B收到A发送的FIN，表示收到了，就会发送ACK回复。

③ 但此时B可能还在发送数据，没有想要关闭数据口的意思，所以FIN与ACK不是同时发送的，而是等到B数据发送完了，才会发送FIN给主机A。

④ A收到B发来的FIN，知道B的数据也发送完了，回复ACK， A等待2MSL以后，没有收到B传来的任何消息，知道B已经收到自己的ACK了，A就关闭链接，B也关闭链接了。

对于建链接的3次握手，主要是要初始化Sequence Number 的初始值。通信的双方要互相通知对方自己的初始化的Sequence Number（缩写为ISN：Inital Sequence Number）。所以叫SYN，全称Synchronize Sequence Numbers。也就上图中的 x 和 y。这个号要作为以后的数据通信的序号，以保证应用层接收到的数据不会因为网络上的传输的问题而乱序（TCP会用这个序号来拼接数据）。

1. A为什么等待2MSL，从TIME\_WAIT到CLOSE？

在Client（客户端）发送出最后的ACK回复，但该ACK可能丢失。Server如果没有收到ACK，将不断重复发送FIN片段。所以Client不能立即关闭，它必须确认Server接收到了该ACK。Client会在发送出ACK之后进入到TIME\_WAIT状态。Client会设置一个计时器，等待2MSL的时间。如果在该时间内再次收到FIN，那么Client会重发ACK并再次等待2MSL。所谓的2MSL是两倍的MSL(Maximum Segment Lifetime)。MSL指一个片段在网络中最大的存活时间，2MSL就是一个发送和一个回复所需的最大时间。如果直到2MSL，Client都没有再次收到FIN，那么Client推断ACK已经被成功接收，则结束TCP连接。

1. ARP是什么？

ARP是地址解析协议，简单语言解释一下工作原理。

① 首先，每个主机都会在自己的ARP缓冲区中建立一个ARP列表，以表示IP地址和MAC地址之间的对应关系。

② 源主机要发送数据时，首先检查ARP列表中是否有对应IP地址的目的主机的MAC地址，如果有，则直接发送数据，如果没有，就向本网段的所有主机发送ARP数据包，该数据包包括的内容有：源主机IP地址，源主机MAC地址，目的主机的IP地址。

③ 当本网络的所有主机收到该ARP数据包时，首先检查数据包中的IP地址是否是自己的IP地址，如果不是，则忽略该数据包，如果是，则首先从数据包中取出源主机的IP和MAC地址写入到ARP列表中，如果已经存在，则覆盖，然后将自己的MAC地址写入ARP响应包中，告诉源主机自己是它想要找的MAC地址。

④ 源主机收到ARP响应包后。将目的主机的IP和MAC地址写入ARP列表，并利用此信息发送数据。如果源主机一直没有收到ARP响应数据包，表示ARP查询失败。

广播发送ARP请求，单播发送ARP响应。

1. 各种协议

ICMP协议：因特网控制报文协议。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。

TFTP协议：是TCP/IP协议族中的一个用来在客户机与服务器之间进行简单文件传输的协议，提供不复杂、开销不大的文件传输服务。

HTTP协议：超文本传输协议，是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。

DHCP协议：动态主机配置协议，是一种让系统得以连接到网络上，并获取所需要的配置参数手段。

NAT协议：网络地址转换属接入广域网(WAN)技术，是一种将私有（保留）地址转化为合法IP地址的转换技术，

DHCP协议：一个局域网的网络协议，使用UDP协议工作，用途：给内部网络或网络服务供应商自动分配IP地址，给用户或者内部网络管理员作为对所有计算机作中央管理的手段。

1. RARP

RARP是逆地址解析协议，作用是完成硬件地址MAC到IP地址的映射，主要用于无盘工作站，因为给无盘工作站配置的IP地址不能保存。工作流程：在网络中配置一台RARP服务器，里面保存着IP地址和MAC地址的映射关系，当无盘工作站启动后，就封装一个RARP数据包，里面有其MAC地址，然后广播到网络上去，当服务器收到请求包后，就查找对应的MAC地址的IP地址装入响应报文中发回给请求者。因为需要广播请求报文，因此RARP只能用于具有广播能力的网络。

1. HTTP长连接和短连接

### 长连接和短连接

**在HTTP/1.0中，默认使用的是短连接**。也就是说，浏览器和服务器每进行一次HTTP操作，就建立一次连接，但任务结束就中断连接。如果客户端浏览器访问的某个HTML或其他类型的 Web页中包含有其他的Web资源，如JavaScript文件、图像文件、CSS文件等；当浏览器每遇到这样一个Web资源，就会建立一个HTTP会话。

但从**HTTP/1.1起，默认使用长连接**，用以保持连接特性。使用长连接的HTTP协议，会在响应头有加入这行代码：Connection: Keep-Alive

HTTP协议的长连接和短连接，实质上是TCP协议的长连接和短连接。

长连接的情况，client向server发起连接，server接受client连接，双方建立连接。Client与server完成一次读写之后，它们之间的连接并不会主动关闭，后续的读写操作会继续使用这个连接。

长连接：建立连接——数据传输...（保持连接）...数据传输——关闭连接

TCP短连接的情况，client向server发起连接请求，server接到请求，然后双方建立连接。client向server 发送消息，server回应client，然后一次读写就完成了，这时候双方任何一个都可以发起close操作，不过一般都是client先发起 close操作。

短连接：建立连接——数据传输——关闭连接...建立连接——数据传输——关闭连接

### 什么时候用长连接和短连接？

**长连接**多用于操作频繁，点对点的通讯，而且连接数不能太多情况。每个TCP连接都需要三步握手，这需要时间，如果每个操作都是先连接，再操作的话那么处理速度会降低很多，所以每个操作完后都不断开，处理时直接发送数据包就OK了，不用建立TCP连接。例如：数据库的连接用长连接，如果用短连接频繁的通信会造成socket错误，而且频繁的socket创建也是对资源的浪费。

而像WEB网站的http服务一般都用**短链接**，因为长连接对于服务端来说会耗费一定的资源，而像WEB网站这么频繁的成千上万甚至上亿客户端的连接用短连接会更省一些资源，如果用长连接，而且同时有成千上万的用户，如果每个用户都占用一个连接的话，那可想而知吧。所以并发量大，但每个用户无需频繁操作情况下需用短连好。

1. 从输入网址到获得页面的过程

Step1：根据输入的网址查询DNS(Domain Name System)，获取域名对应的IP地址；

查询DNS的先后顺序为：浏览器搜索自身的DNS缓存，搜索操作系统的DNS缓存，读取本地的HOST文件，发起一个DNS的系统调用(带宽运营服务器查看本身缓存，运营商服务器发起一个迭代DNS解析请求)。

Step2：浏览器获得域名对应的IP地址后，向目标服务器发起三次握手连接请求；

Step3：TCP/IP连接建立起来之后，浏览器就向服务器发送Http请求；

Step4：服务器接收到请求之后，根据路径参数，经过后端的一些处理生成HTML页面代码返回给浏览器；

Step5：浏览器拿到完整的HTML页面代码开始解析和渲染，如果遇到引用的外部JS，CSS，图片等静态资源，它们同样也是一个一个的Http请求，都需要经过上面的步骤；

Step6：浏览器根据拿到的资源对页面进行渲染，最终把一个完整的页面呈现给用户。

1. 基本的Http流程

Http是Hyper Text Transfer Protocol(超文本传输协议)的缩写，主要负责服务器与浏览器之间的通信。Http把客户端浏览器的请求发送到服务器，并把响应的网页内容由服务器返回到客户端浏览器。

Http协议的特点：

① Http是一种无状态协议；

② Http协议在应用层，在服务器每个Http都需要启动独立的线程去处理；

Http会话有四个过程：

① 客户端与服务器建立tcp连接；

② 客户端向服务器发出请求文档；

③ 服务器向客户端发出响应文档；

④ 释放tcp连接。

1. Socket编程

Socket在计算机中提供了一种通信接口，可以通过这个端口与任何一个具有socket接口的计算机通信，应用程序在网络上传输，接收的信息都通过这个socket接口来实现。

Socket通信原理如下：

服务器端实现监听连接🡪客户端实现发送连接请求🡪连接建立🡪发送和接收数据进行通信。

1. Http与Https的区别

协议 明文 端口 证书 是否简单

https是Http协议运行在TCP之上，所有传输的内容都是明文，客户端和服务端都无法验证对方的身份；

https是HTTP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处理，加密采用对称加密，但对称加密的秘钥用服务器方的证书进行了非对称加密。此外客户端可以验证服务端的身份，如果配置了客户端验证，服务器方也可以验证客户端的身份；

https协议需要到ca申请证书，一般免费证书很少，需要交费；

http是超文本传输协议，信息明文传输，https则是具有安全性的ssl加密传输协议；

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；

http的连接很简单，是无状态的；

https协议是由SSL+HTTP协议构建的可进行加密传输，身份认证的网络协议要比http协议更加安全。

1. TCP流量控制和拥塞控制

## 1.流量控制

在确认应答机制中，对每一个发送的数据段，都要给一个ACK确认应答，收到ACK后再发送下一个数据段。这样做有一个比较大的缺点，就是性能较差。尤其是数据往返时间较长的时候。那么我们可不可以一次发送多个数据段呢：滑动窗口。

所谓的流量控制就是让发送方的发送速率不要太快，让接收方来得及接收。利用滑动窗口机制可以很方便的在TCP连接上实现对发送方的流量控制。

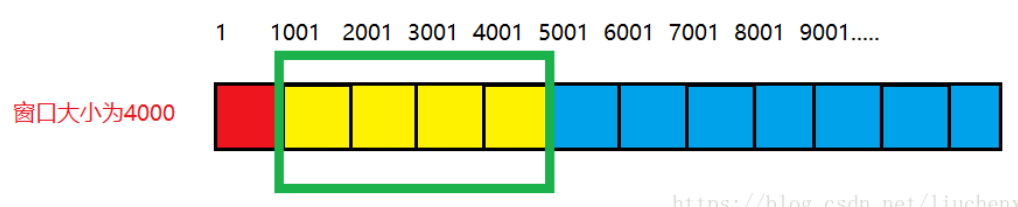
### 1.窗口类型

1.接收端窗口rwnd（recv window）：接收端缓冲区大小。接收端将此窗口值放在TCP报文的首部中的窗口字段，传送给发送端；

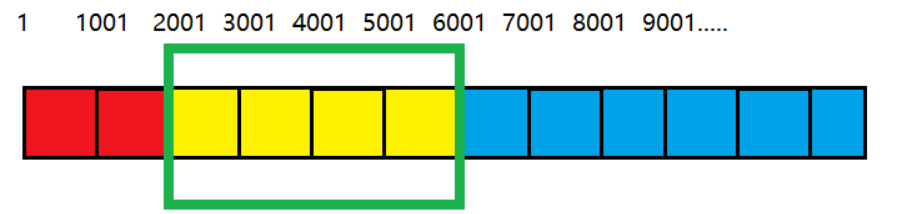
2.拥塞窗口cwnd（congestion window）：发送端缓冲区大小；

3.发送窗口swnd（send window）：发送窗口的上限值 = Min [rwnd, cwnd]。

### 2.过程



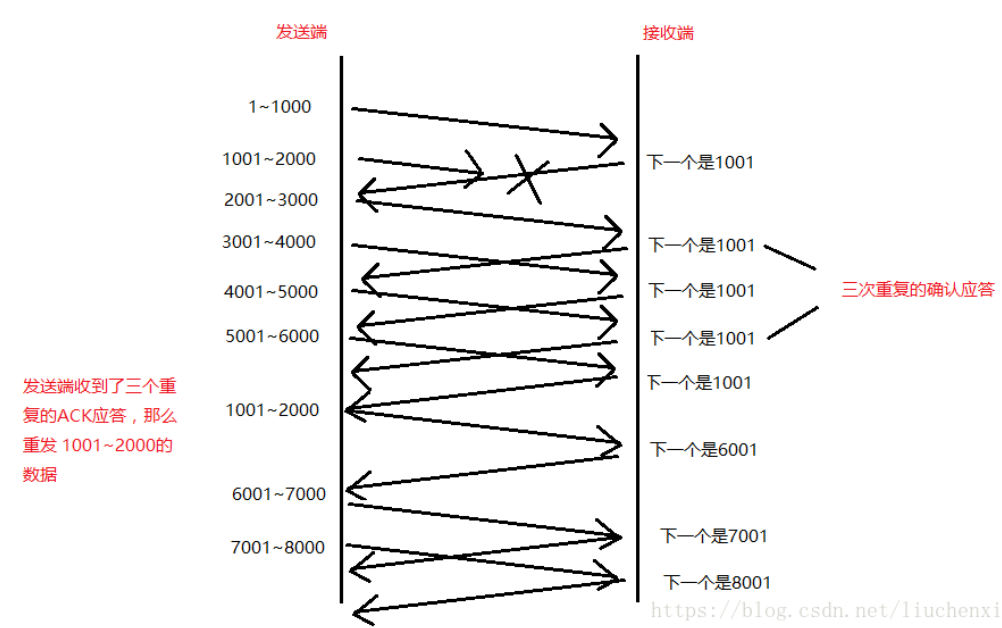
如图，假设窗口大小为4000。那么就是是说图上1001-5001之间的数据是可以一次性全部发送，并且不需要等待ACK响应的。当发送完这4000的数据后，此时等待。等到ACK响应中确认收到1001-2000的数据时，滑动窗口右移。



滑动窗口内部的数据都是已经发送但是没有收到ACK应答的数据，滑动窗口左侧都是已经收到了ACK应答的数据，滑动窗口右侧是未发送的数据。

### 3.丢包处理(快重传机制)

#### 1.发送端发送的数据丢包

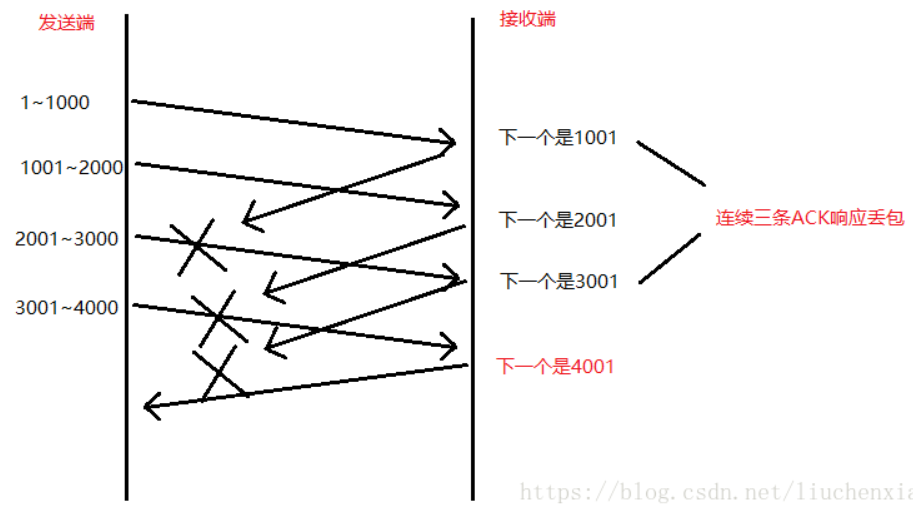


这里1001-2000的数据报文丢失，此时发送端并不知道，继续发送滑动窗口内的报文。这时候接收到了三条重复的ACK响应（TCP协议规定，收到三条相同的ACK响应就出发重传）。此时重传1001-2000的数据报文。重传完毕后，等待接收端的响应。此时接收端响应为下一条是6001的数据报文，也就是说在2001-6000的报文都接收到了，这段报文被放到了接收端的接收缓冲区中。那么此时滑动窗口继续移动，发送端继续发送。

接收到三个重复的ACK响应，就开始重传响应所要求的报文的机制就是快重传速机制。

快速重传机制的触发为什么是三个重复的ACK响应？因为如果连续收到两个ACK，极有可能是乱序问题，如果收到三个ACK，那么很大概率是丢包了，这个值在实际过程中也不一定就是三，可能远大于3，只是一个经验值。

#### 2.接收端ack相应丢包



接连三条的ACK响应都丢包，但是第四条ACK响应到达。此时并不会影响发送端的发送。因为ACK响应的是下一条数据包是4001，这意味着接收端已接收到前面的所以报文数据。所以接收端ACK响应丢包其实对发送端发送的影响并不是那么大，后续的ACK响应能够处理好这个问题。

## 2.拥塞控制

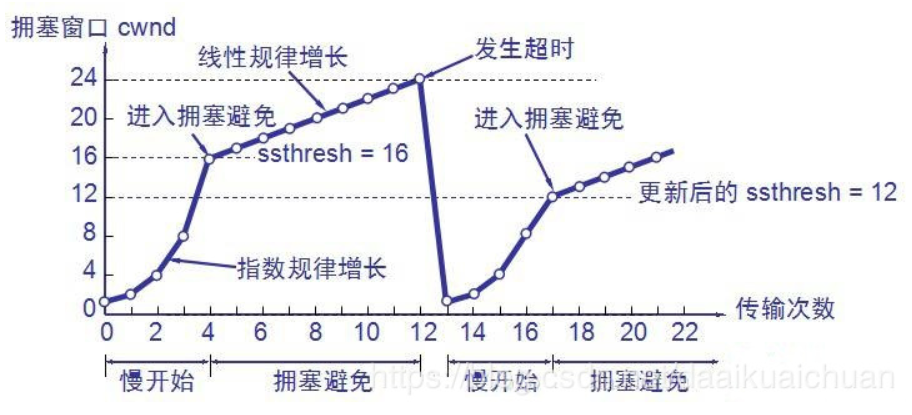
拥塞控制也就是考虑当前的网络环境，动态调整窗口大小，没有发生拥塞情况，则窗口增大，拥塞了窗口减小，如此往复，最终应该接近与接收端的窗口大小。

拥塞控制主要是四个算法：1）慢启动，2）拥塞避免，3）拥塞发生，4）快速恢复。

### 1.慢启动和拥塞避免

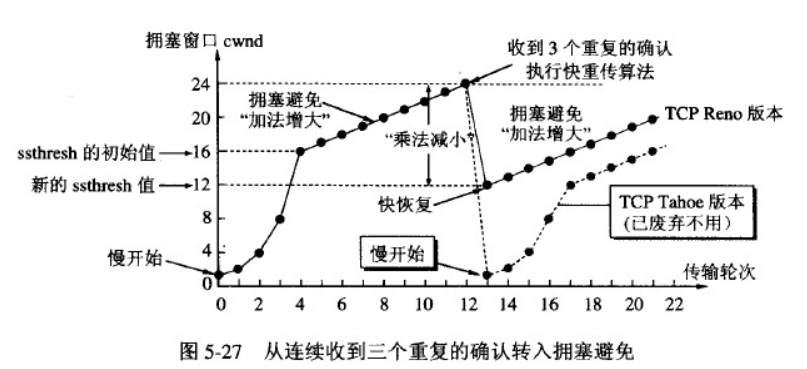
在开始发送信息时，由于不知道具体的网络环境，为避免大量信息造成的拥塞现象，此时的拥塞窗口以最小值（即拥塞窗口和接收端窗口中的较小值）进行数据发送，并设定门限值作为慢启动算法和拥塞避免算法的分割点。慢启动是指以最小的拥塞窗口按照指数形式递增，达到门限值后，以拥塞避免算法，即线性递增方式增大拥塞窗口（这里递增时间间隔为一个往返时间RTT）。

在上述过程中，无论是窗口大小指数递增或者线性递增，当发生拥塞现象，则门限值更新为当前窗口大小的一半，拥塞窗口大小变为最小值，重复上述递增过程（此时属于网络环境限制，所以在接收端和拥塞窗口两个限制条件中选择拥塞窗口作为限制）。



拥塞窗口大小为什么先以指数增加再以线性增加？窗口大小首先以指数递增去探测一下网络的拥塞程度，执行拥塞避免算法后，拥塞窗口线性缓慢增大，防止网络过早出现拥塞。

### 2.快重传和快恢复



TCP在收到乱序到达包时就会立即发送ACK，TCP利用3个相同的ACK来判定数据包的丢失，此时进行快速重传，快速重传做的事情有：

1.把ssthresh（慢启动阈值）设置为cwnd（拥塞窗口）的一半

2.把cwnd再设置为ssthresh的值(具体实现有些为ssthresh+3)

3.重新进入拥塞避免阶段。

后来的“快速恢复”算法是在上述的“快速重传”算法后添加的，当收到3个重复ACK时，TCP最后进入的不是拥塞避免阶段，而是快速恢复阶段。快速重传和快速恢复算法一般同时使用。快速恢复的思想是“数据包守恒”原则，即同一个时刻在网络中的数据包数量是恒定的，只有当“老”数据包离开了网络后，才能向网络中发送一个“新”的数据包，如果发送方收到一个重复的ACK，那么根据TCP的ACK机制就表明有一个数据包离开了网络，于是cwnd加1。如果能够严格按照该原则那么网络中很少会发生拥塞，事实上拥塞控制的目的也就在修正违反该原则的地方。

具体来说快速恢复的主要步骤是：

1.当收到3个重复ACK时，把ssthresh设置为cwnd的一半，把cwnd设置为ssthresh的值加3，然后重传丢失的报文段，加3的原因是因为收到3个重复的ACK，表明有3个“老”的数据包离开了网络。

2.再收到重复的ACK时，拥塞窗口增加1。

3.当收到新的数据包的ACK时，把cwnd设置为第一步中的ssthresh的值。原因是因为该ACK确认了新的数据，说明从重复ACK时的数据都已收到，该恢复过程已经结束，可以回到恢复之前的状态了，也即再次进入拥塞避免状态。

1. IO

**同步、异步**描述的是：**客户端在请求数据的过程中，能否做其他事情。**

**阻塞、非阻塞**描述的是：**客户端与服务端是否从头到尾始终都有一个持续连接，以至于占用了通道，不让其他客户端成功连接。**

1.BIO（同步阻塞）

定义：客户端在请求数据的过程中，**保持一个连接**，**不能做其他事情**。

那么BIO存在两个问题：

1. 由于**连接是双向**的，“始终保持一个连接”，则说明，对于客户端和服务端而言，都需要一个线程来维护这个连接，如果服务端没有数据给客户端，则客户端需要一直等待，该连接也需要一直维持。假设一个连接需要5MB的内存，不考虑多任务的情况下，客户端总是要花费固定的5MB。那么对服务端，1个客户端建立连接则需要花5MB，10个就要50MB，1000个就要5GB。显然，阻塞给服务器带来的性能负担极大。
2. 客户端不能做其他事情，只能等待该请求的完成，其本身的性能没有得到充分的释放，所以等待就是浪费时间。

2.NIO（同步非阻塞）

定义：客户端在请求数据的过程中，**不用保持一个连接**，不能做其他事情。

上面提到BIO，当有很多个客户端同时向服务端请求数据时，其连接所花费的开销就极大。那么NIO就使用了“不用始终保持一个连接”的方式，解决该问题。其过程为：

客户端发送一个请求，并建立一个连接，服务端接收到了。如果服务端没有数据，就告知客户端“没有数据”；如果有数据，则返回数据。客户端接到了服务端回复的“没有数据”就断开连接，过了一段时间后，客户端重新问服务端是否有数据。服务器重复以上步骤。

客户端反复建立连接询问，如果没有数据则断开连接。这个过程称为“轮询”。**NIO用轮询代替了始终保持一个连接。**

那么这样具体会有什么收益呢？

我们考虑以下问题：假如一个轮询连接只持续1s，服务器需要4s来准备一个数据，客户端在接到“没有数据”的回复后，隔1s再轮询一次。

对于BIO，1000个连接就需要5GB，在4s内，服务器内存消耗都是5GB。

对于NIO，在第1s内，服务器接收1000个连接的请求并花费5GB；在第2s内，服务器没有接收任何请求；在第3s内，服务器再次花费5GB接收1000个连接；第4s内没有请求；第5s开始时，处理所有请求返回结果。

整个流程是：为了接收1000个连接的请求，第1和第3s花费5GB，第2和第4s花费0GB，平均下来则是2.5GB。换个角度实际上是，1000个连接需要花费2.5GB，则2000个请求需要花费5GB。

在该例子中，NIO的容纳量比BIO高了一倍（5GB的容纳量从1000变成2000）

所以NIO的收益就是，节约了“始终保持一个连接”的内存消耗。

3.AIO（异步非阻塞）

定义：客户端在请求数据的过程中，不用保持一个连接，**可以做其他事情**。

AIO也不用始终保持一个连接，但是其处理方式和NIO是不同的。并且这个方式让客户端可以做其他事情。

AIO用了一个通知机制，其流程如下：

客户端向服务端请求数据。服务端若有，则返回数据；若无，则告诉客户端“没有数据”。客户端收到“没有数据”的回复后，就做自己的其他事情。服务端有了数据之后，就主动通知客户端，并把数据返回去。

如此一来，整个请求流程中，不仅维持连接的消耗没了，而且客户端可以做别的事情了，节约了客户端的时间。

需要提的是，这里解决了连接的消耗，但是也必然引入了**别的消耗**。这里让客户端能先做别的事情，也肯定会带来**新的麻烦**。

别的消耗是指，服务端需要主动通知客户端，关于“通知”的业务逻辑肯定是需要消耗资源的。新的麻烦是指，客户端本来在做别的事情，突然前面的事情又插过来要做了，必然引入了一个多线程的协调工作。