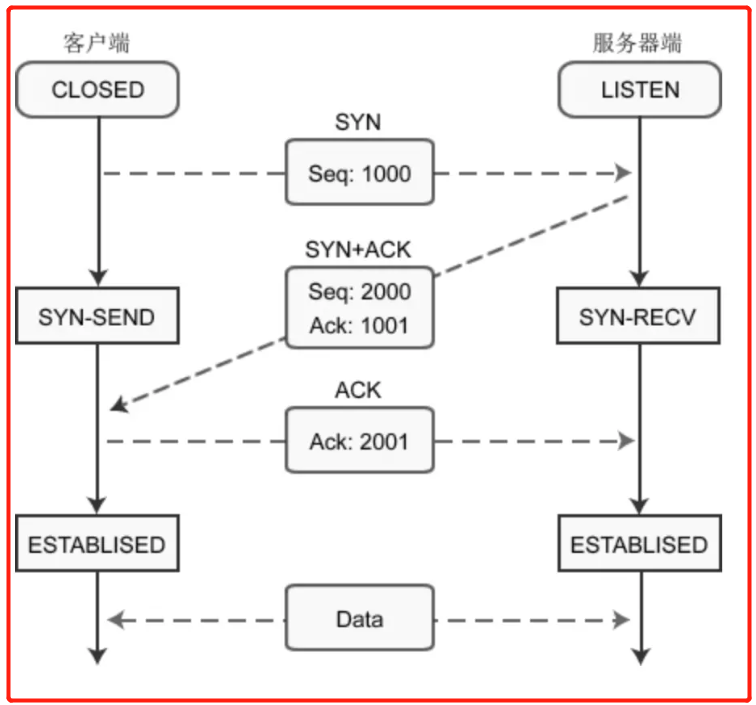
# TCP协议

## 三次握手



首先当服务端处于 **listen** 状态的时候，我们就可以在客户端发起连接了，此时客户端由Cloce状态变为**SYN\_SENT**状态，服务端收到这个消息后会返回一个SYN并且同事ACK客户端的请求，之后服务端便会处于**SYN\_RECEIVED**状态。这个时候客户端收到服务器端的SYN&ACK,就会对服务器端发送ACK,之后便处于ESTABLISHED

经常在面试中可能有人提问：为什么握手要3次，不是2次或者4次呢？

首先说4次握手，其实为了保证消息可靠性，这个握手可以一直循坏下去；但是没有一个终止就没有意义了。所以3次，保证了各方消息有来有回就够了。当然这里可能有一种情况是，客户端发送的ACK在网络中被丢了，那怎么办？

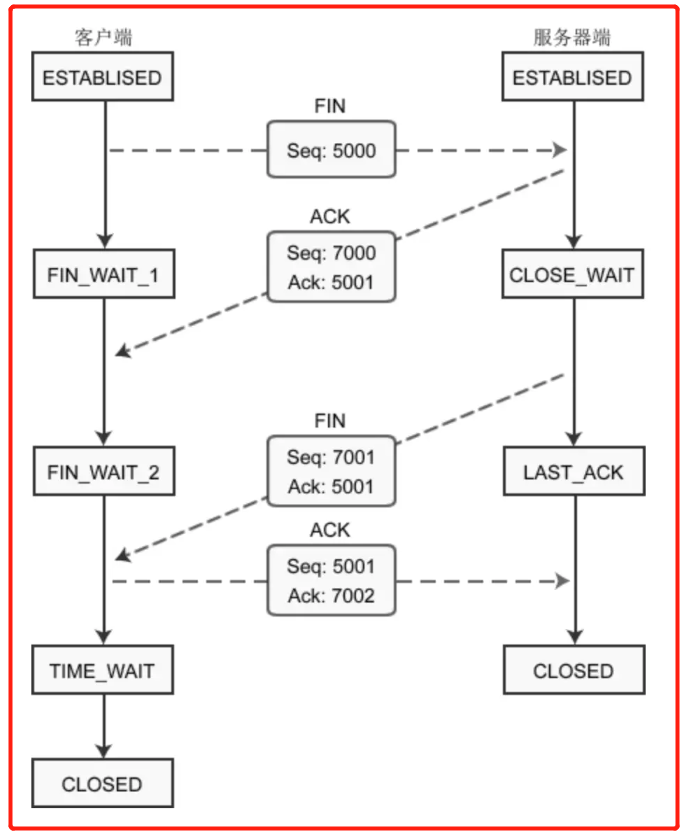
1. 其实大部分时候，我们连接建立完成就会立刻发送数据，所以如果服务器端没有收到ACK没关系，当收到数据就会认为连接已经建立
2. 如果连接建立后不立马传输数据，那么服务端就会认为没有建立成功会周期性重发SYN&ACK直到客户端确认成功。

再说为什么2次握手不行呢？2次握手我们可以想象是没有三次握手最后的 **ACK**, 在实际中确实会出现客户端发送 **ACK** 服务端没有收到的情况（上面的情况一），那么这是否说明两次握手也是可行的呢？看下情况二，2次握手当服务器端发送消息后，就认为建立成功，而恰巧此时又没数据传输。这就会带来一种资源浪费的情况。比如：客户端可能由于延时发送了多个连接情况，当服务器端每收到一个请求回复后就认为连接建立成功，但是这其中很多请求都是延时产生的重复连接，浪费了很多宝贵的资源。

因此综上所述，从资源节省、效率3次握手都是最合适的。话又回来三次握手的真实意义其实就是协商传输数据用的：**序列号与窗口大小**。

**抓包：** sudo tcpdump -n host www.baidu.com -S

## 四次挥手

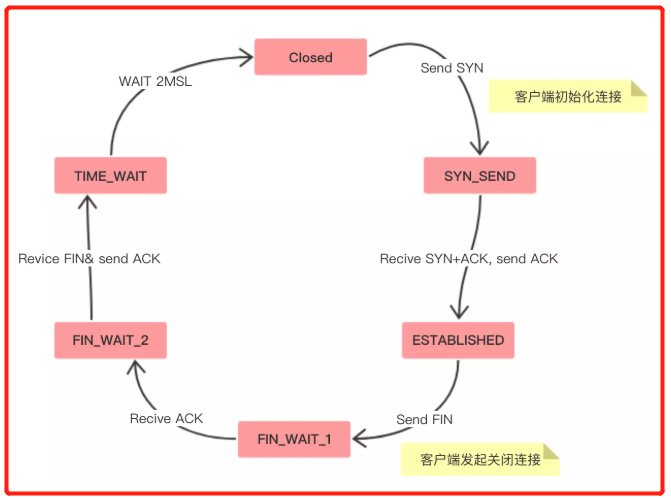


这里有个问题需要注意下，其实客户端、服务端都能够主动发起关闭操作，谁调用 close() 就先发送关闭的请求。当然一般的流程，发起建立连接的一方会主动发起关闭请求（http中）。

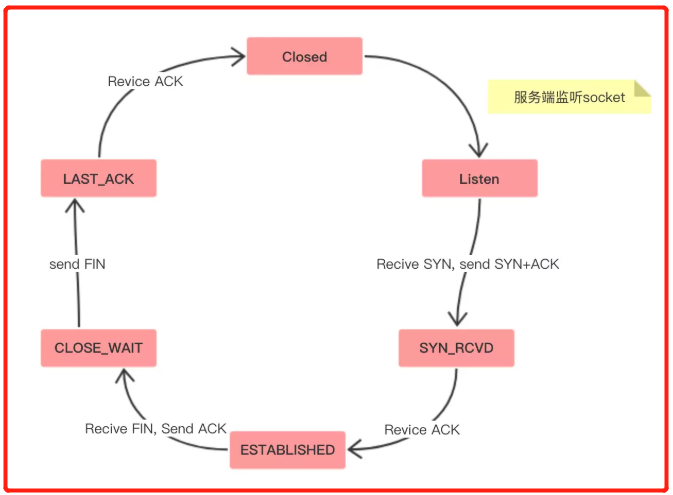
客户端发起FIN请求后（第一次挥手），由ESTABLISHED状态变为FIN\_WAIT\_1状态，服务器端收到FIN请求会回复ACK给客户端（第二次挥手），状态变为CLOSE\_WAIT,而客户端状态会变为FIN\_WAIT\_2，之后服务器端再发送一次FIN给客户端（第三次挥手），服务器端状态由CLOSE变为LAST\_ACK,客户端收到FIN消息后也会给服务器端发送一个ACK(第四次挥手)，客户端状态由FIN\_WAIT\_2变为TIME\_WAIT,最后变为CLOCE,当然这时候服务器端收到ACK后也会变为CLOSE

**close\_wait** 与 **time\_wait** 两个重要的状态

## 客户端状态变更



## 服务端状态变更



## TCP的流量控制与拥塞控制

我们常说TCP是面向连接的，UDP是无连接的。那么TCP这个面向连接主要解决的是什么问题

下面是抓包数据

2 0:33:26.583598 IP 192.168.0.102.58165 > 103.235.46.39.80: Flags [S], seq 621839080, win 65535, options [mss 1460,nop,wscale 6,nop,nop,TS val 1050275400 ecr 0,sackOK,eol], length 0

20:33:26.660754 IP 103.235.46.39.80 > 192.168.0.102.58165: Flags [S.], seq 1754967387, ack 621839081, win 8192, options [mss 1452,nop,wscale 5,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,sackOK,eol], length 0

20:33:26.660819 IP 192.168.0.102.58165 > 103.235.46.39.80: Flags [.], ack 1754967388, win 4096, length 0

上面我们说到 TCP 的三次握手最重要的就是协商传输数据用的序列号。那这个序列号究竟有些什么用呢？这个序号能够帮助后续两端进行确认数据包是否收到、解决顺序、丢包问题；另外我们还可以看到win字段，这个是双方交流的窗口大小，这在每次数据传输过程中也会携带。主要是告诉对方，我窗口这这么大，别发太多或别发太少。

总结下，TCP的几个特点是：

* 顺序问题，依靠序号
* 丢包问题，依靠序号
* 流量控制，依靠滑动窗口
* 拥塞控制，依靠拥塞窗口+滑动窗口
* 连接维护，三次握手/四次挥手

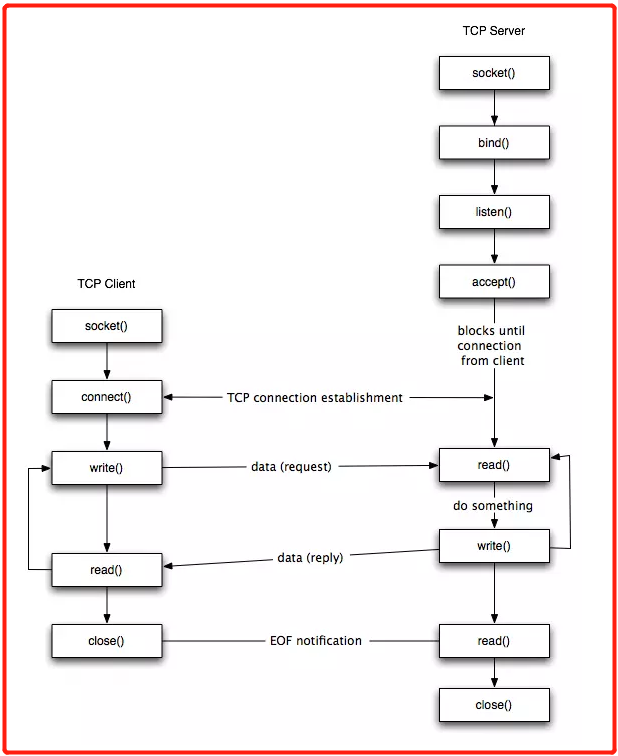
## 顺序与丢包问题

这个问题其实应该很好理解。由于数据在传输前我们已经有序号了，这里注意一下这个序号是随机的，重复的概率极低，避免了程序发生乱入的可能性。

由于我们每个数据包有序号，虽然发送与到达可能不是顺序的，但是TCP层收到数据后，可以根据序号从新排序；另外在这个排序的过程中，发现有了1,2,3,5,6这个几个包，一检查就知道4号要么延时未到达，要么丢包了，等待重传

这里需要重要说明的一点是。为了提升效率，TCP其实并不是收到一个包就发一个ack。是如何ACK的呢？还是以上面为例，TCP收到了1,2,3,5,6这几个包，它可能会发送一个 ack ，seq=3 的确认包，这样次一次确认了3个包。但是它不会发送 5,6 的ack。因为4没有收到啊！一旦4延时到达或者重发到达，就会发送一个 ack, seq=6，又一次确认了3个包。

## 是什么限制了你的连接



我们假设我的服务端就是 Nginx ，我来尝试解读一下。当客户端调用 connect() 时候就会发起三次握手，这次握手的时候有几个元素唯一确定了这次通信（或者说这个socket），**[源IP:源Port， 目的IP:目的Port]** ，当然这个socket还不是最终用来传输数据的socket，一旦握手完成后，服务端会在返回一个 **socket** 专门用来后续的数据传输。。这里暂且把第一个socket叫 **监听socket**，第二个叫 **传输socket** 方便后文叙述。

## 什么是半连接队列？

服务器第一次收到客户端的 SYN 之后，就会处于SYN\_RCVD状态，此时双方还没完全建立其连接，服务器会把此种状态下的请求放在一个队列里，我们把这种队列称之为半连接队列

当然还有一个全连接队列，就是已经完成了三次握手，建立起连接的就会放在全连接队列中。如果队列满了有可能会出现丢包现象。

在这里补充一点关于SYN\_ACK重传次数的问题：服务器发送完SYN\_ACK包后，如果未收到客户端确认包，服务器就会进行首次重传，等待一段时间后仍没收到，进行第二次重传。如果重传次数大于系统规定的最大重传次数，系统则会将改连接信息从半连接队列里面删除。注意，每次重传的时间间隔不一定相同，一般呈指数增加。

## 三次握手过程中可以携带数据吗？

其实第三次握手的时候，是可以携带数据的。但是，**第一次、第二次握手不可以携带数据**

为什么这样呢?大家可以想一个问题，假如第一次握手可以携带数据的话，如果有人要恶意攻击服务器，那他每次都在第一次握手中的SYN报文中放入大量数据。因为攻击者根本就不理服务器的接收、发送能力是否正常，然后疯狂的重复发送SYN报文的话，这会让服务器花费很多时间、内存来接收这些报文

也就是说，第一次握手不可以放数据，其中一个简单的原因就是**因为会让服务器更加容易受到攻击了**

## SYN攻击是什么？

**服务器端的资源分配是在二次握手时分配的，而客户端的资源是在完成三次握手时分配的，**以服务器容易受到SYN洪泛攻击。SYN攻击就是Client在短时间内伪造大量不存在的IP地址，并向Server不断地发送SYN包，并等待Client确认，由于源地址不存在，因此Server需要不断重发直至超时，这些伪造的SYN包将长时间占用未连接队列，导致正常的SYN请求因为队列满而被丢弃，从而引起网络拥塞甚至系统瘫痪。SYN 攻击是一种典型的 DoS/DDoS 攻击。

在 Linux/Unix 上可以使用系统自带的 netstats 命令来检测 SYN 攻击。

netstat -n -p TCP | grep SYN\_RECV

## 挥手为什么需要四次？

因为当服务端收到客户端的SYN连接请求报文后，可以直接发送SYN\_ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。但是关闭连接时，当服务器端受到FIN报文时，很可能并不会立即关闭SOCKET,所以只能先回复一个ACK报文，告诉客户端“你发的FIN报文我收到了”，只有等到我服务器端所有报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要4次挥手

## 四次挥手释放连接时，等待2MSL的意义?

为了保证客户端发送的最后一个ACK报文段能够到达服务器。因为这个ACK有可能丢失，从而导致处在LAST-ACK状态的服务器收不到对FIN-ACK的确认报文。服务器会超时重传这个FIN-ACK，接着客户端再重传一次确认，重新启动时间等待计时器。最后客户端和服务器都能正常的关闭。假设客户端不等待2MSL，而是在发送完ACK之后直接释放关闭，一但这个ACK丢失的话，服务器就无法正常的进入关闭连接状态。