链滴

TCP 协议基本特性

作者：zouchanglin

原文链接：https://ld246.com/article/1574346739605

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20190303.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h2 id="TCP协议特点">TCP 协议特点</h2>
<h2 id="1-面向连接">1、面向连接</h2>
传输控制协议 Transmission Control Protocol 要对数据的传输进行一个详细的控制，TCP 是面
连接(虚连接)的传输层协议，但注意此处的连接是虚连接：
<img src="https://ld246.com/images/img-loading.svg" alt="net01.png" data-src="https:
/b3logfile.com/file/2019/11/net01-6827890e.png?imageView2/2/interlace/1/format/jpg"></
>
<h2 id="2-每个TCP连接只能是点对点">2、每个 TCP 连接只能是点对点</h2>
TCP 连接是点对点的连接，而不是向广播那样的形式
<h2 id="3-可靠有序-不丢不重">3、可靠有序，不丢不重</h2>
TCP 提供可靠交付的服务，无差错、不丢失、不重复、按序到达。可靠有序，不丢不重，后面会
到这些功能是如何实现的
<h2 id="4-TCP提供全双工通信">4、TCP 提供全双工通信</h2>
发送缓存：准备发送的数据 + 已发送但尚未收到确认的数据
接收缓存：按序到达但尚未被接受应用程序读取的数据 + 不按序到达的数据
<h2 id="5-面向字节流">5、面向字节流</h2>
TCP 面向字节流，TCP 把应用程序交下来的数据看成仅仅是一连串的无结构的字节流。
<h2 id="TCP报文格式">TCP 报文格式</h2>
<img src="https://ld246.com/images/img-loading.svg" alt="net02.png" data-src="https:
/b3logfile.com/file/2019/11/net02-8d345924.png?imageView2/2/interlace/1/format/jpg"></
>
<h2 id="1-填充字段">1、填充字段</h2>
首先看看填充字段，这个字段主要是为了保证添加选项字段后 TCP 首部是还是 4 字节的整数倍
通常填充的是全零字段）。
固定首部就是 20 字节（图中一行就是 32 位，8 位是一个字节，所以图中的一行就是 4 字节，
行就是 20 字节）
<h2 id="2-源端口和目的端口">2、源端口和目的端口</h2>
源端口和目的端口分别占用 4 字节，总占用 8 字节
<h2 id="3-序号">3、序号</h2>
序号：在一个 TCP 连接中传送的字节流中的每一个字节都按顺序编号，本字段表示本报文段所
送数据的第一个字节的序号。如下图，如果发的是 123 号字节的数据，那么 TCP 报文中的头部序号
是 1，如果发送的是 456 号字节的数据，那么 TCP 报文中的头部序号就是 4：
<img src="https://ld246.com/images/img-loading.svg" alt="net03.png" data-src="https:
/b3logfile.com/file/2019/11/net03-f94f8dbc.png?imageView2/2/interlace/1/format/jpg"></p

<h2 id="4-确认号">4、确认号</h2>
确认号：期望收到对方下一一个报文段的第一个数据字节的序号。若确认号为 N，则证明到序号
-1 为止的所有数据都已正确收到。如下图：接收端收到数据后要反馈给发送端，还是向发送端发一个
认报文，填写自己的确认号，发送端就知道了接收端收到了那些数据，没收到的超时要重发：
<img src="https://ld246.com/images/img-loading.svg" alt="net04.png" data-src="https:
/b3logfile.com/file/2019/11/net04-7d08cffb.png?imageView2/2/interlace/1/format/jpg"></p

<h2 id="5-数据偏移">5、数据偏移</h2>
数据偏移(首部长度)： TCP 报文段的数据起始处距离 TCP 报文段的起始处有多远，以 4 字节为
位，即 1 个数值是 4 字节，首部长度包含固定的 20 字节，还有选项字段和填充字段，所以需要知道
部的长度，就需要这个字段来说明首部长度。 由于数据偏移字段只有 4 位的存储空间，最大为 1111
，就是最大值为 15，15 * 4 = 60 字节，所以 TCP 协议首部最长为 60 字节（20 字节固定大小 + 40
字节的选项和填充字段）。
<h2 id="6-6个控制位">6、6 个控制位</h2>
<h3 id="URG---紧急位">URG - 紧急位</h3>
紧急位 URG，URG=1 时， 标明此报文段中有紧急数据，是高优先级的数据，应尽快传送，不
在缓存里排队，配合紧急指针字段使用

<h3 id="ACK---确认位">ACK - 确认位</h3>
确认位 ACK，ACK=1 时确认号有效，在连接建立后所有传送的报文段都必须把 ACK 置为 1</p

<h3 id="PSH---推送位">PSH - 推送位</h3>
推送位 PSH，PSH=1 时， 接收方尽快交付接收应用进程，不再等到缓存填满再向上交付
<img src="https://ld246.com/images/img-loading.svg" alt="net05.png" data-src="https:/b3logfile.com/file/2019/11/net05-75e4a09c.png?imageView2/2/interlace/1/format/jpg"></>
<h3 id="RST---复位">RST - 复位</h3>
复位 RST，RST=1 时， 表明 TCP 连接中出现严重差错，必须释放连接，然后再重新建立传输接。
<h3 id="SYN---同部位">SYN - 同部位</h3>
同步位 SYN，SYN=1 时，表明是一个连接请求/连接接受报文，建立连接的时候表示此报文段请求/连接接受报文
<h3 id="FIN---终止位">FIN - 终止位</h3>
终止位 FIN，FIN=1 时， 表明此报文段发送方数据已发完，要求释放放连接
不过在书中也有这样描述的：
<img src="https://ld246.com/images/img-loading.svg" alt="net09.png" data-src="https:/b3logfile.com/file/2019/11/net09-ff1c395b.png?imageView2/2/interlace/1/format/jpg"></p

这个会在后面的拥塞控制中讲到！
<h2 id="7-窗口">7、窗口</h2>
窗口指的是发送本报文段的一方的接收窗口，即现在允许对方发送的数据量，反映了自己可以缓的字节流，那么对方就知道了你的接收能力怎么样，以此决定向你发送报文的数据量应该控制在多大范围。举个简单的例子，如果接收方发来的报文确认号是 701， 窗口是 1000，那么发送方接下来要的报文就是 701~1700 的数据。
<h2 id="8-校验和">8、校验和</h2>
检验首部 + 数据，检验时要加上 12 字节的伪首部（IP 首部）第四个字段为 6（UDP 中是 17）
<h2 id="9-紧急指针">9、紧急指针</h2>
当 URG=1 时，这个字段才有用，指出本报文段中紧急数据的字节数，假设紧急指针为 50，那数据部分的 1-50 的数据才是紧急数据，剩下的是普通数据
<h2 id="10-选项">10、选项</h2>
选项，例如最大报文段长度 MSS、窗口扩大、时间戳、选择确认...
<h2 id="TCP的连接管理">TCP 的连接管理</h2>
建立连接------》数据传送 --------》 释放连接
TCP 连接的建立采用客户服务器方式，主动发起连接建立的应用进程叫做客户，而被动等待连接立的应用进程叫服务器。
<h2 id="1-建立连接过程">1、建立连接过程</h2>
step1、客户端发送连接请求报文段，无应用层数据， SYN = 1，seq = x(随机值，由客户端主产生)，SYN 为 1 很好理解，前面说到了 SYN=1 表明是一个连接请求/连接接受报文，建立连接的时表示此报文段为请求/连接接受报文
step2、服务器端为该 TCP 连接分配缓存和变量，并向客户端返回确认报文段，允许连接，无应层数据，SYN = 1，ACK = 1，seq=y(随机值)，ack=x+1。怎么理解呢？客户端首次发报文，这个文是建立连接用的，所以客户端随机产生了一个序号 seq=y，但是这个序号对于服务器是有用的，因服务器通过客户端发来的 seq 就知道需要接收的下一个报文段的序号了，那就是 seq 的值再加一；</>
<img src="https://ld246.com/images/img-loading.svg" alt="net06.png" data-src="https:/b3logfile.com/file/2019/11/net06-cbb94454.png?imageView2/2/interlace/1/format/jpg"></>
step3、客户端为该 TCP 连接分配缓存和变量，并向服务器端返回确认的确认，并且可以携带数
<img src="https://ld246.com/images/img-loading.svg" alt="net07.png" data-src="https:/b3logfile.com/file/2019/11/net07-8db3b1ac.png?imageView2/2/interlace/1/format/jpg"></

>
总结一下就是：
第一次发数据包 (客户端-> 服务器)：客户端知道自己的发送能力正常；服务器知自己的接收能力正常，也知道客户端的发送能力正常
第二次发数据包 (服务器-> 客户端)：客户端知道服务器的接收、发送能力正常，户端知道自己的接收能力正常；服务器知道自己的发送能力正常
那么现在的问题就是：
客户端知道服务器的发送、接收能力都正常，同时也知道自己的发送、接收都正常，么是不是可以通信了呢？？？？ 不可以，因为服务器目前只知道：自己的发送、接收能力正常，客户的发送能力正常
唯一不能确定的是客户端的接收能力是否正常，所以通过第三次握手，确定了客户端接收能力也是正常的！
补上一张加上状态的图
<img src="https://ld246.com/images/img-loading.svg" alt="net11.png" data-src="https:/b3logfile.com/file/2019/11/net11-ffa39477.png?imageView2/2/interlace/1/format/jpg"></p

<h2 id="2-连接的释放过程">2、连接的释放过程</h2>
<img src="https://ld246.com/images/img-loading.svg" alt="net08.png" data-src="https:/b3logfile.com/file/2019/11/net08-bd2711cf.png?imageView2/2/interlace/1/format/jpg"></
>
可以看出这是一个四次握手的过程
参与一条 TCP 连接的两个进程中的任何一个都能终止该连接，连接结束后，主机中的 资源 (缓和变量) 将被释放。
step1、客户端发送连接释放报文段，停止发送数据，主动关闭 TCP 连接。
FIN=1，seq=u （u 就等于前面已经传送过来的数据的最后一个字节的序号加 1）
step2、服务器端回送一个确认报文段，客户到服务器这个方向的连接就释放了，处于半关闭状（close_wait），这个半关闭状态就是客户端停止发送数据，服务器端还可以发送数据。 TCP 服务器知高层的应用进程，客户端向服务器的方向就释放了，这时候处于半关闭状态，即客户端已经没有数要发送了，但是服务器若发送数据，客户端依然要接受。这个状态还要持续一段时间，也就是整个 CLSE-WAIT 状态持续的时间。
ACK=1，seq=v，ack=u+1
step3、服务器端发完数据，就发出来连接释放报文段，主动关闭 TCP 连接
FIN=1，ACK=1，seq=w，ack=u+1
step4、客户端回送确认报文段，再等待时间计时器设置的 2MSL（最长报文段寿命）后，连接底关闭
ACK=1，seq=u+1，ack=w+1
<img src="https://ld246.com/images/img-loading.svg" alt="net10.png" data-src="https:/b3logfile.com/file/2019/11/net10-a6b30c96.png?imageView2/2/interlace/1/format/jpg"><b
>
<img src="https://ld246.com/images/img-loading.svg" alt="net12.png" data-src="https://b3ogfile.com/file/2019/11/net12-2de5aad1.png?imageView2/2/interlace/1/format/jpg">
<h2 id="TCP的其他问题">TCP 的其他问题</h2>
<h2 id="1-TCP泛洪攻击">1、TCP 泛洪攻击</h2>
SYN 洪泛攻击，这种方式利用 TCP 协议的特性，就是三次握手。攻击者发送 TCP SYN, SYN 是CP 三次握手中的第-一个数据包，而当服务器返回 ACK 后，该攻击者就不对其进行再确认，那这个 TP 连接就处于挂起状态，也就是所谓的半连接状态，服务器收不到再确认的话，还会重复发送 ACK攻击者。这样更加会浪费服务器的资源。攻击者就对服务器发送非常大量的这种 TCP 连接，由于每个 都没法完成三次握手，所以在服务器上，这些 TCP 连接会因为挂起状态而消耗 CPU 和内存，最后务器可能死机，就无法为正常用户提供服务了。
如何解决 TCP 泛洪攻击？
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
② SYN Cookie
它使用一种特殊的算法生成 seq，这种算法考虑到了对方的 IP、端口、己方 IP、端口的固定信

，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，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 报后，重新计算一遍，看其是否与对方回应报文中的（seq-1）相同，从而决定是否分配 TCB 资源
③ SYN Cache
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 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。
④ SYN Proxy 防火墙
一种方式是防止墙 dqywb 连接的有效性后，防火墙才会向内部服务器发起 SYN 请求。防火墙服务器发出的 SYN ACK 包使用的序列号为 c, 而真正的服务器回应的序列号为 c', 这样，在每个数据文经过防火墙的时候进行序列号的修改。另一种方式是防火墙确定了连接的安全后，会发出一个 safe eset 命令，client 会进行重新连接，这时出现的 syn 报文会直接放行。这样不需要修改序列号了。但，client 需要发起两次握手过程，因此建立连接的时间将会延长。