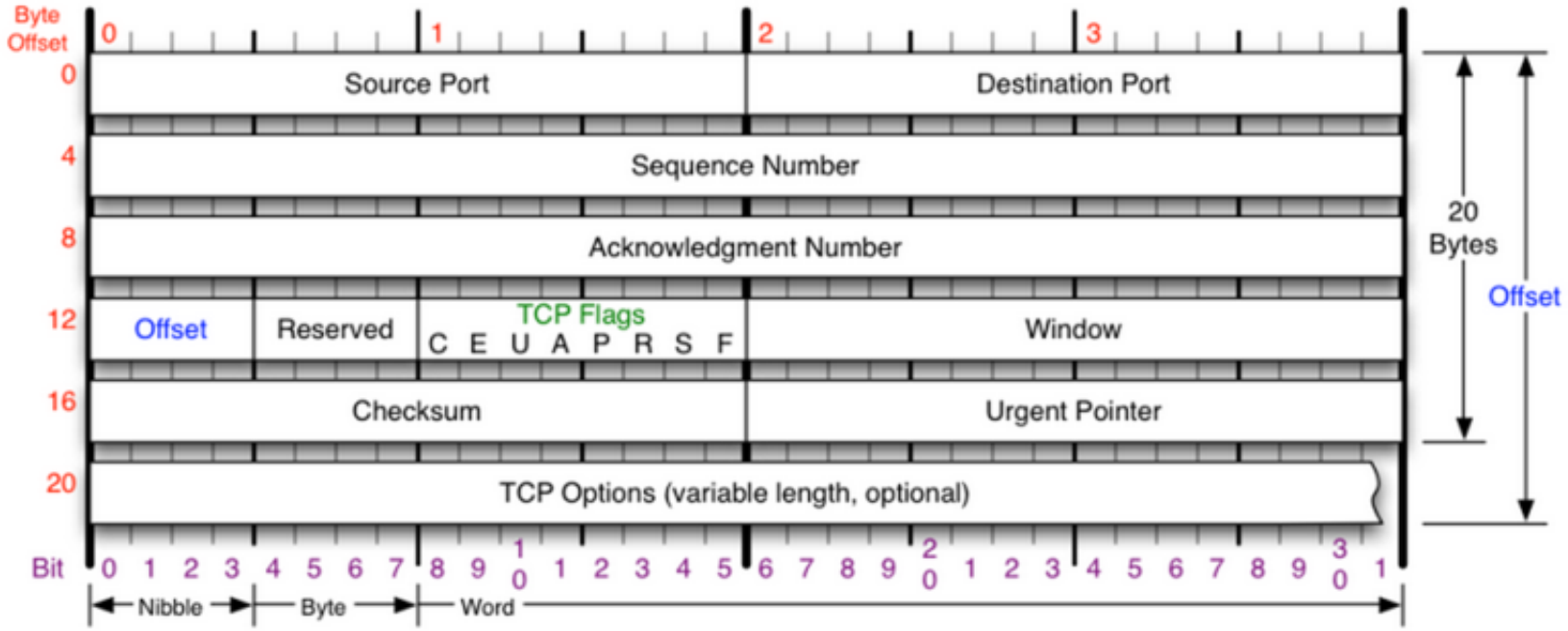
layout	title	description	category
post	浅谈 TCP		tcp

这篇文章介绍 TCP 理论，主要分为「TCP 连接，TCP 优化，TCP 重连」三个方面。看《TCP/IP 详解》 TCP 章节时候有些疑问，网上也搜到一些有价值的信息，整理后记录成文章，同时加深下自己的理解。 关于 TCP 实际应用 在这篇文章中介绍。

1. TCP 连接

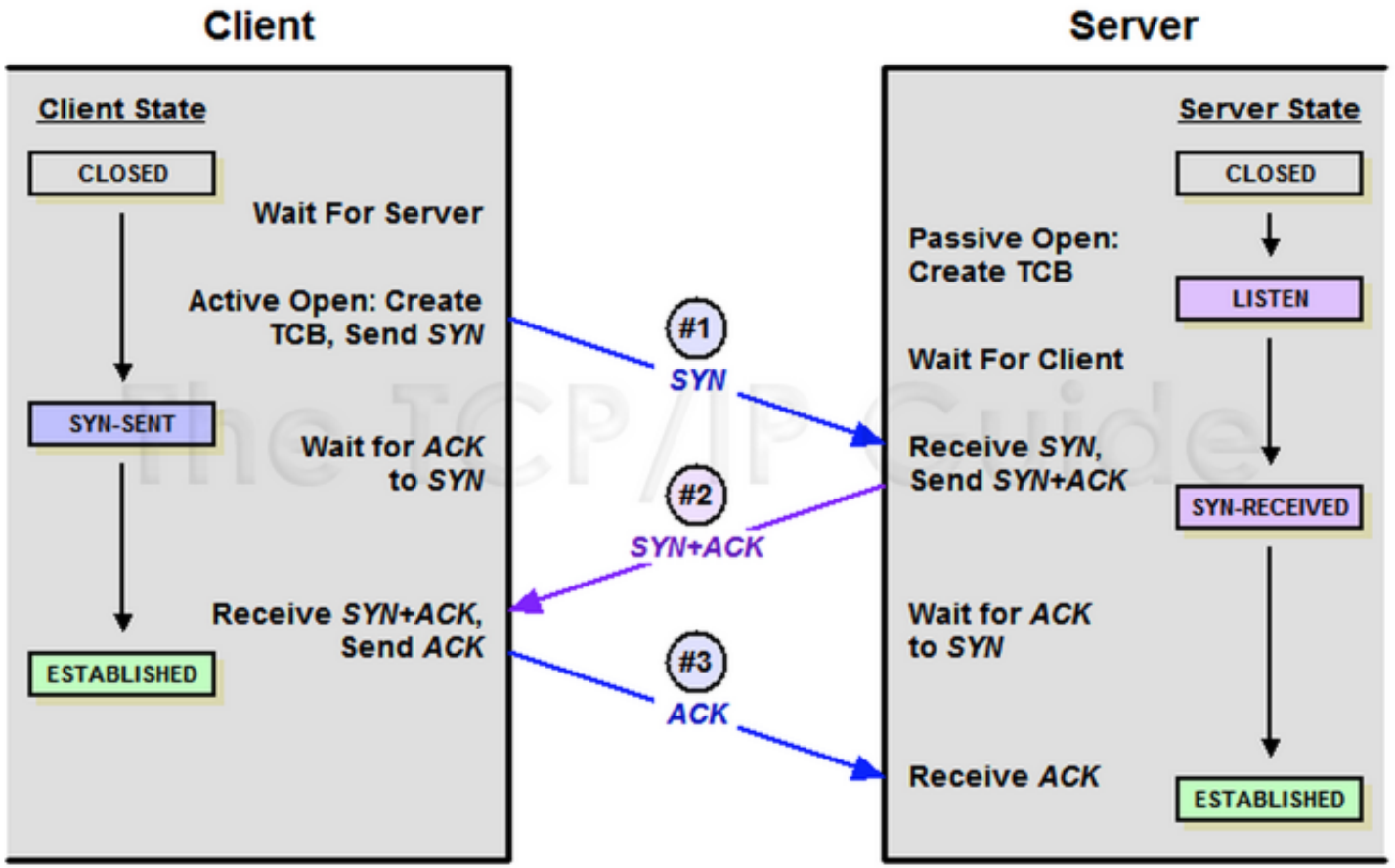
1.1 TCP 格式



这些字段的含义：

1. Source Port/Destination Port：表示本机/目的端口号。**通过四元组（源 IP，源端口，目的端 IP，目的端端口）确定唯一的 TCP 连接。**源 IP 和目的端 IP 保存在 IP 头部，TCP 数据包被包裹上 IP 数据的头形成 IP 数据包，再在网络上进行传输。
2. Sequence Number：标识序号。发送端会首先产生初始序号 ISN（Initial Sequence Number），以后发送端每次向接收端发送的数据包的 Sequence Number 都需要累加上次数据包字节长度，这样只需要根据 Sequence Number 就可以判断出数据包的传输顺序。（数据重传后可辨别原来的数据传输顺序）
3. Acknowledgment Number：TCP Flags 为 ACK 时有效，返回 SYN + 1 表示成功接受到数据。
4. Window：窗口大小（Sliding Window Protocol），用来通知对方自己接受数据的最大窗口。

1.2 三次握手（建立连接



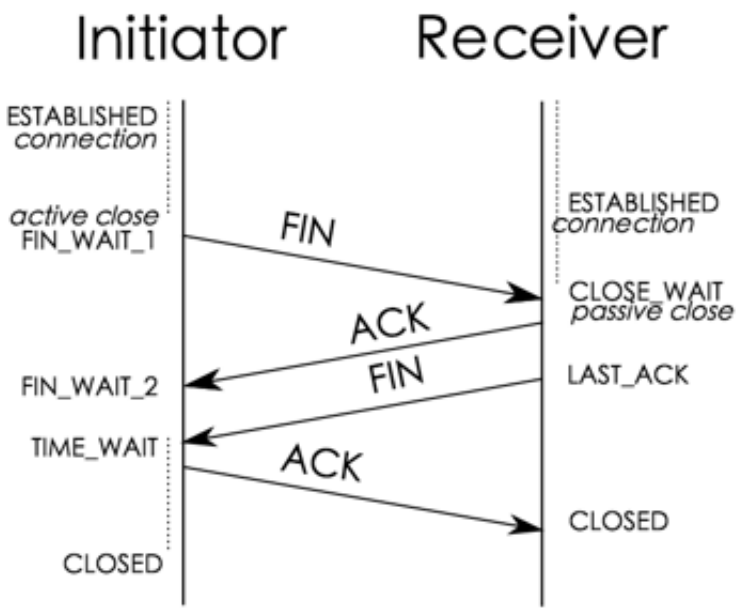
发送 SYN 的一端执行主动打开（active open），接受这个 SYN 并发回 SYN 的一端执行被动打开（active open）。TCP 不对 ACK 报文段进行确认，TCP 只确认那些包含数据的 ACK 报文段。

为什么要三次握手：

如果只有两次握手，client 发送数据后因为一些原因数据在网络中被延迟了，并在连接关闭后传递到 server，server 误以为 client 建立了一个新连接，所以返回一个 ACK，但 client 并没有创建新连接，所以不会理睬 server 的请求，也不向 server 发送数据，而 server 又不知道 client 的状态，导致 server 一直在等待 client 的请求造成资源浪费。所以三次握手的目的是：为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误。

TCP 作为安全协议，仅在建立连接时需要进行三次握手，数据传输不需要三次握手（ACK 叠加机制）。有次同事问，为什么建立连接需要三次握手，直接发数据会怎样？仔细想之后，建立连接的过程就是确立第一个 `Sequence Number` 的过程。这样双方都认可第一个 `Sequence Number` 之后，就可以保证数据的有序性。

1.3 四次挥手（断开连接



发送 FIN 信号后，表示 Initiator 没有数据要发送了。因为 TCP 是全双工模型，所以每个方向都需要单独关闭，收到一个 FIN 仅意味着在一个方向上没有数据流动。一个 TCP 连接在收到 FIN 后还可以继续发送数据。

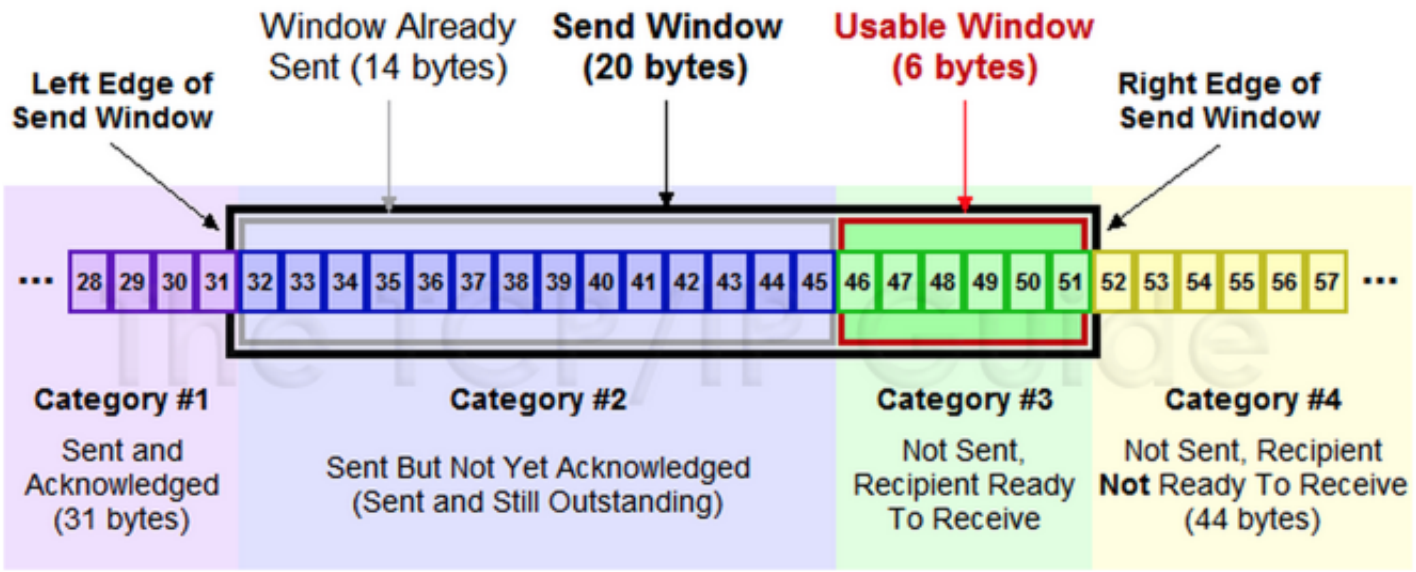
为什么要四次挥手？

FIN 仅表示 Initiator 数据发送完了，但 Receiver 还可以继续发出数据。所以在第二/三次挥手，即第二个 `ACK` 和 `FIN` 之间，Receiver 可以继续向 Initiator 发送数据。所以 ACK 和 FIN 不能放在一起发送。

另外一个重要的概念是：**TIME_WAIT** 状态，有兴趣可以自己查资料。

2. TCP 的优化

2.1 滑动窗口



上图分成了四部分，黑色框即窗口大小：

- Category #1：表示已经 ACK 确认的数据。
- Category #2：表示已经发送，但是还没收到 ACK 确认的数据。
- Category #3：还未发送，等待发送的数据。已发送数据还小于窗口大小。
- Category #4：还不能发送的数据，需要等待窗口移动。

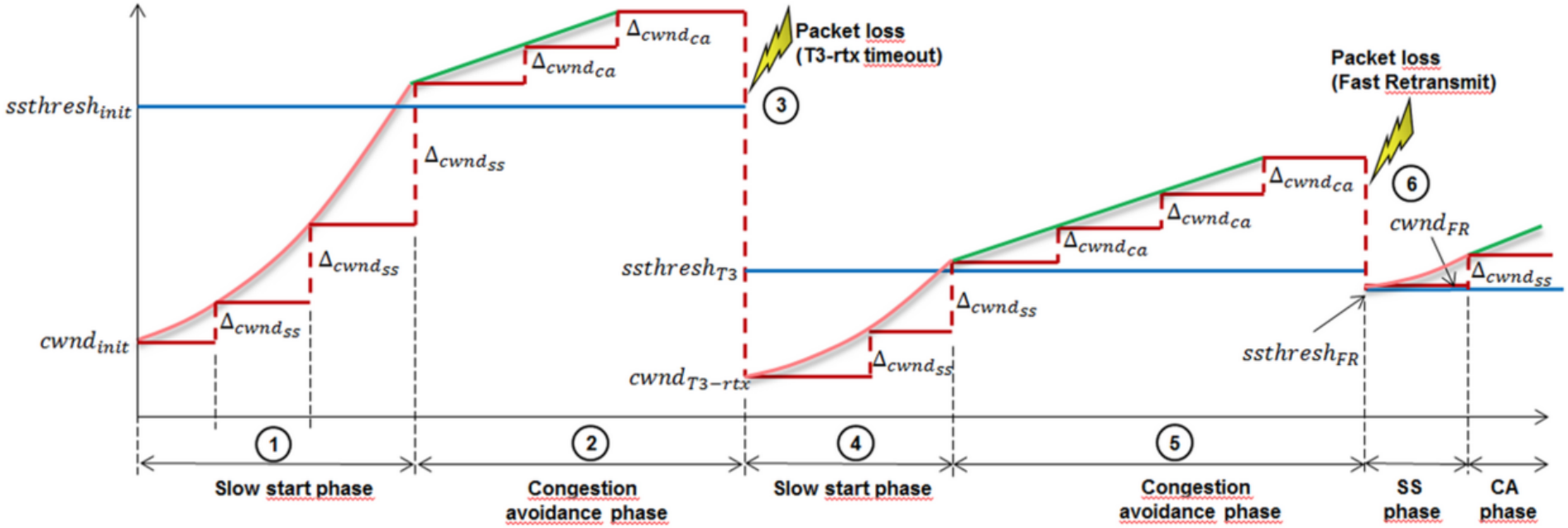
工作原理：

采用 TCP 协议，发送端发出数据后需要等待接收到 ACK 才能继续数据传输，当网络中存在较大延迟后会导致大部分时间都是用来等待 ACK 。所以诞生了著名的滑动窗口协议。 在每台机器上都存在两个滑动窗口，一个用于接收数据，一个用于传输数据。要发送和要确认的数据首先被暂存在滑动窗口中，发送端可以连续发出多组数据，而不需要每发出一组数据就等待 **ACK**。当收到 **ACK** 后再更新滑动窗口，避免 **ACK** 的等待。（发送多个 SYN 后可能会只收到一个 ACK，因为接收端可保证收到了 ACK 序号减 1 的序号之前的所有数据。） 另外一个用处是：通过 TCP Header 中的 Window 可获得连接对方的窗口大小（cwnd），避免发出的数据超出接收端的接受能力，造成数据丢包。 所以，滑动窗口主要用于解决 **TCP** 传输效率和流量控制的问题。 延迟确认/*Nagle* 算法：避免网络中传输大量小包导致网络拥塞。

2.2 慢启动/拥塞控制

滑动窗口可解决 TCP 连接过程中接收端数据超载的问题，却无法判断网络中数据是否超载。TCP 最大数据传输量为接受窗口（rwnd）和 拥塞窗口（cwnd）的较小值。 慢启动首先初始化 cwnd（拥塞窗口）和 ssthresh 的值，每次经历 RTT（Run Trip Time：也就是一个数据包来回的时间），cwnd 的值会做如下调整：

- 图中 ① 阶段：当 cwnd < ssthresh，cwnd = 2 * cwnd，这段时间 cwnd 的值呈指数形式增长。
- 图中 ② 阶段：当 cwnd >= ssthresh，cwnd = cwnd + 1，直至拥塞发生。
- 图中 ③ ⑥ 表示两种拥塞情况（超时和收到重复确认）。



拥塞窗口（**cwnd**）是发送方的流量控制，接受窗口（**rwnd**）是接收方的流量控制。

3. TCP 的重连

TCP 要保证所有数据到达，所以一定要有重传机制。 首先介绍下 ACK 的传输机制，当发送方发出 1, 2, 3, 4, 5 后，接收方只收到了 1, 2，那么就会返回 ACK = 3，接收端只需返回 SYN 最大 ACK，表示之前的数据都已经接受。而如果当 3 丢失，接收端收到了 4，那么也不能直接发送 ACK = 5 表示 4 成功接收。**SeqNum 和 ACK 都是以字节数为单位，ACK 不能跳着确认，否则发送端以为之前的数据都被成功接收到了。**于是接收到会受到重复的 ACK = 3。

3.1 超时重传：

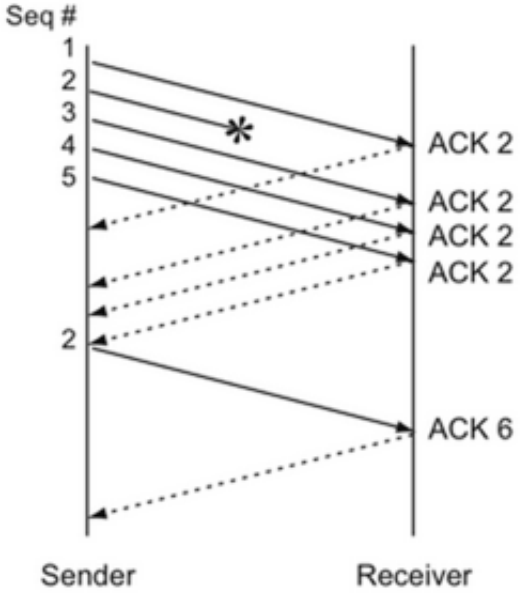
一种重连方式是：死等 3 的 ACK，因为也收不到 4, 5 的 ACK，所以发送端以为 4, 5 也丢失了。超出 RTO（Retransmission Time Out）就进行重连。确认重连后有两种策略：

- 只发送 3 的数据。发送成功后如果收到了 ACK=6，表示 3, 4, 5 都已被成功接受。如果只收到了 ACK = 4，随后继续超时，那么还需要对数据 4, 5 进行重复重连的操作。
- 3, 4, 5 的数据一起发出。 所以第一种方式省带宽，慢，而第二种方式浪费带宽，快。因为都在等 Timeout 所以等待时间可

能会很长。

3.2 快速重传：

TCP 引入一种 Fast Retransmit 的机制，不以事件驱动，而通过数据驱动。当接受者收到 3 个连续的 ACK 后，就重传，不再等



待 Timeout。 但 Sender 收到三个重复 ACK 后同样面临只重传 3 还是 3, 4, 5 一起重传的问题。Selective Acknowledgment (SACK) 解决了这个问题，有兴趣的可以自己了解下 RFC 2018。

参考连接

- 《TCP/IP 详解》
- TCP为什么需要 3 次握手与 4 次挥手
- TCP 的那些事儿（上）
- TCP 的那些事儿（下）
- 浅谈 TCP 优化