问一个问题，TCP中的超时重传，到底多久重传才合适?

超时重传时间的选择是TCP最复杂的问题之一，下面我们来看看具体复杂在哪里，以及推荐的计算方法是什么。

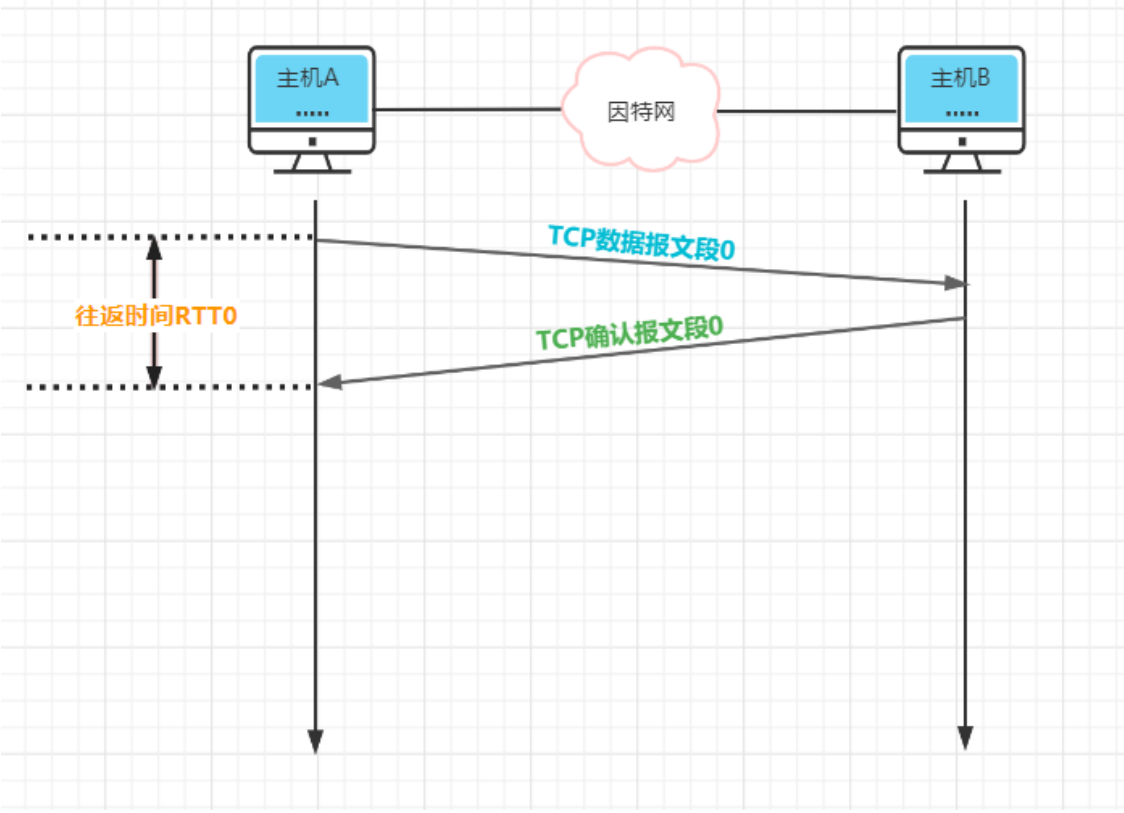
## 一、RTO选择的复杂性

**超时重传时间**英文名叫做**Retransmission Time-Out**，我们往往简称为**RTO**。

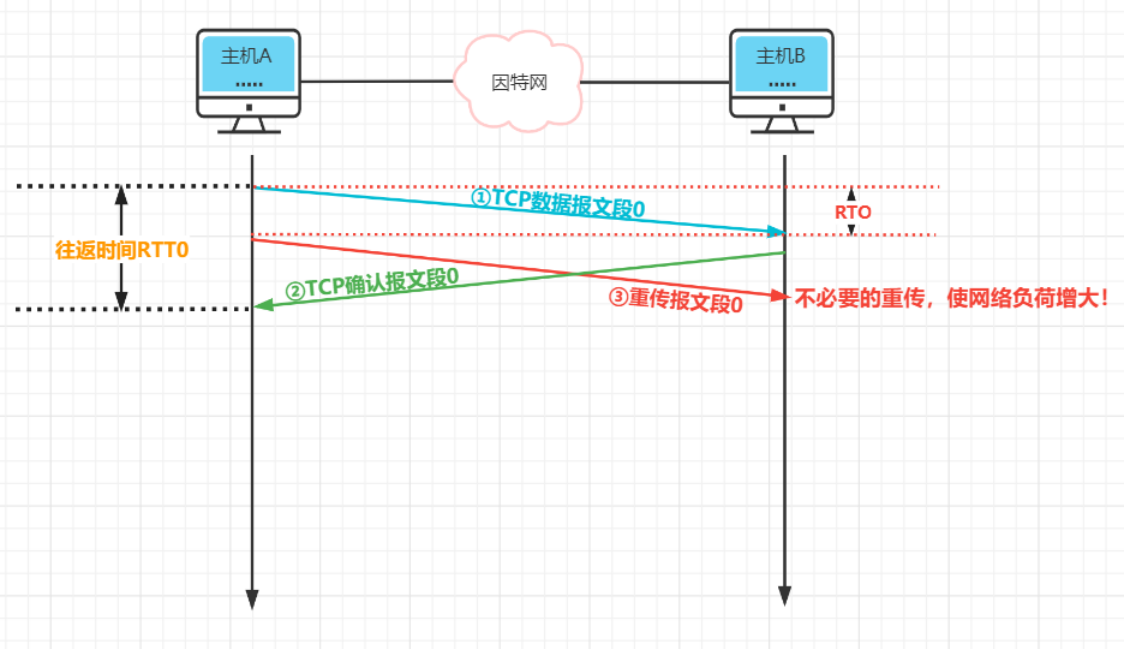
假设有主机A和主机B，通过互联网已经建立了TCP连接，现在主机A要给主机B发送TCP报文段0，并且记录发送时间t0；

主机B收到报文段0后，回复确认报文段0，当主机A收到此确认报文段0后，再次记录收到的时间t1。

t1-t0是报文段0的**往返时间RTT**，由于是0号报文段，因此我们将其称为RTT0。

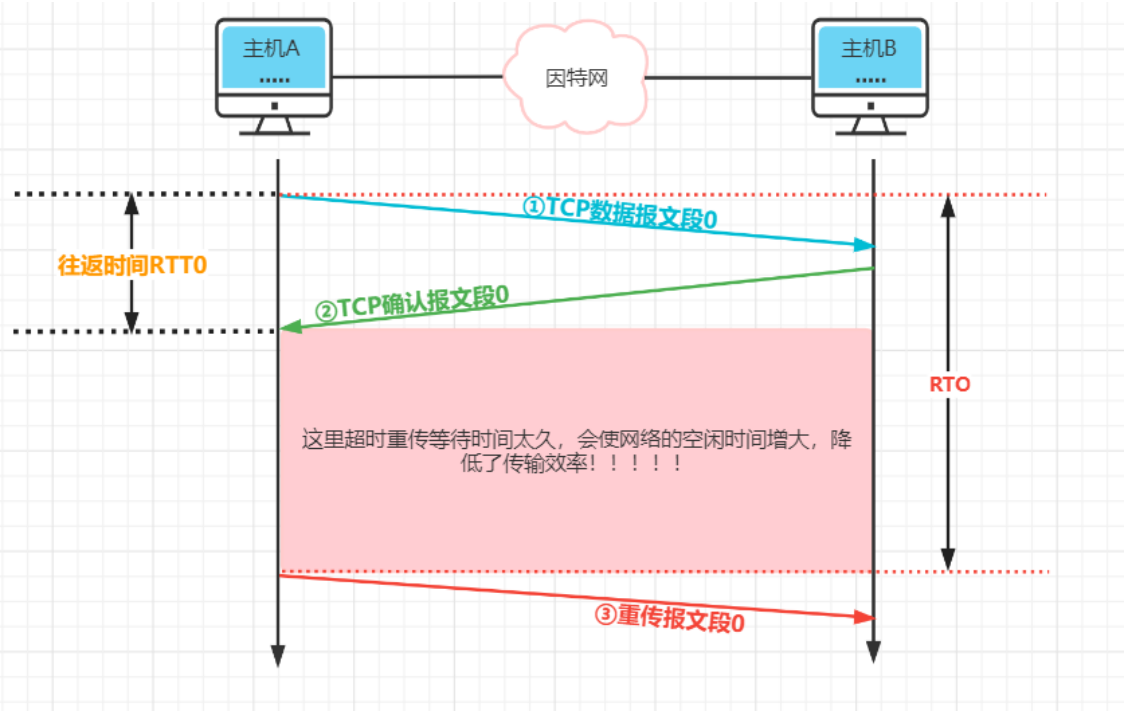



试想下，如果我们将RTO的值设置得比RTT0的值要小：



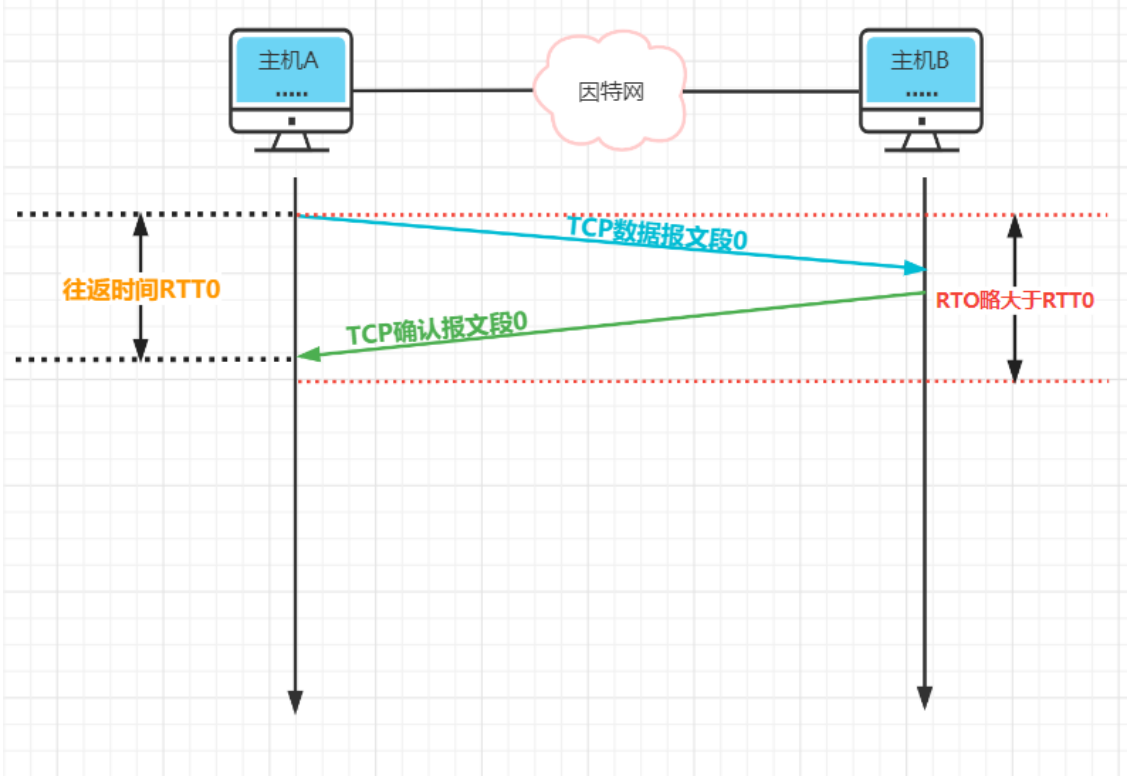

很显然，这会**引起报文段不必要的重传**，使网络负荷增大。

如果我们将RTO的值设置得远比RTT0的值要大呢?



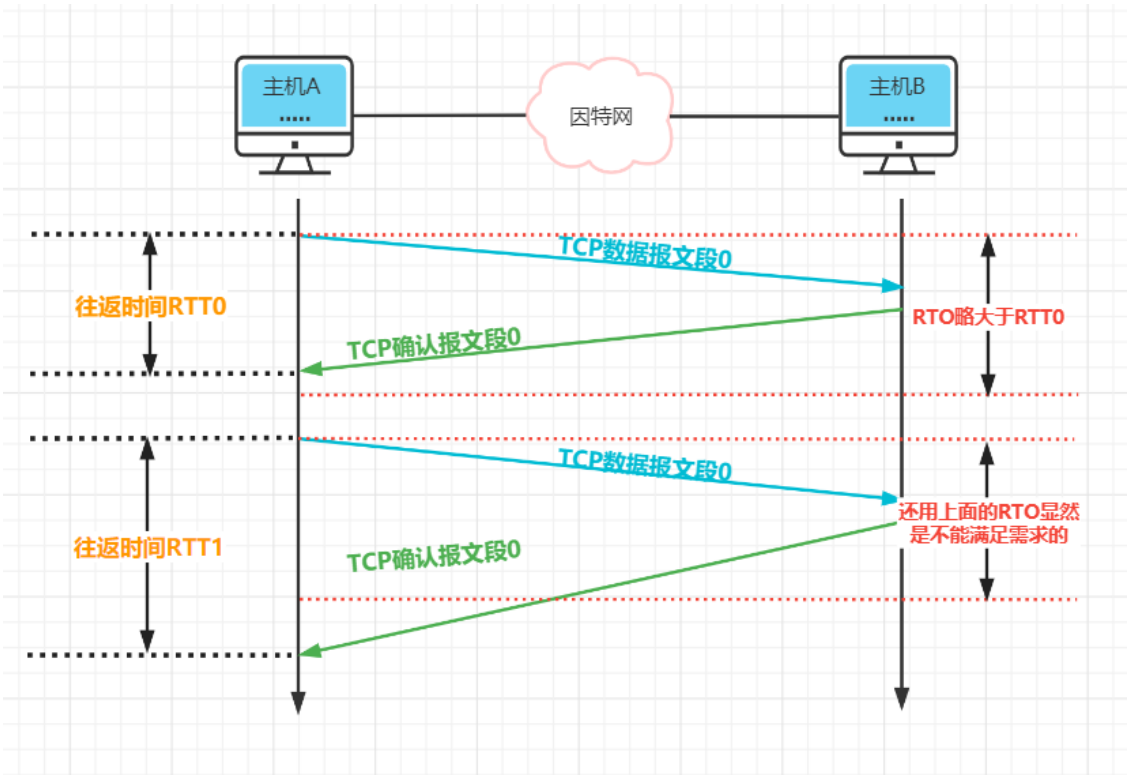

很显然，这会**使重传推迟的时间太长，使网络的空闲时间增大，降低了传输效率**。

综合上述两种情况，我们可以得出这样的结论：超时重传时间RTO的值应该设置为【略大于】报文段往返时间RTT的值。




**得出这个结论十分简单**，但是由于TCP下层是复杂的互联网环境，主机A发送的报文段可能只经过一个高速率局域网，也可能经过多个低速率网络，并且结合我们之前网络层所学，每个IP数据包的转发路由还有可能不同，因此报文段的往返时间RTT可能每次都不一样，那么这个略大于RTT的值就很难确定，比如下面这种情况。

主机A又发送了一个TCP数据报段1，主机B返回返回了确认报文段1，不过这次时间花的有点久，我们记为RTT1，显然RTT1大于RTT0，那么一开始设置的略大于RTT0的超时重传时间显然就不大合适了。



这么看来，超时重传的时间选择确实没有那么简单。

## 二、RTO计算方法

既然我们不能以某一个RTT样本来计算RTO，那么我们可以利用每次测量得到的RTT样本，**计算加权平均往返时间RTTs，又称之为平滑的往返时间**。

当测量到第一个RTT样本时，RTTs的值直接取为第一个样本值，即RTTs1=RTT1，当继续收到RTT样本后的计算方式是：

```
新的RTTs = (1-α) * 旧的RTTs + α * 新的RTT样本
```

当α接近0时，则新的RTT样本值对RTTs的影响不大；当α接近1时，则新的RTT样本值对RTTs的影响较大。

**已成为建议标准的RFC6298推荐的α值是1/8，即0.125。**

用这种方法得出的加权平均往返时间RTTs就比测量出来的RTT值更加**平滑**。

显然，超时重传时间RTO的值略大于加权平均往返时间RTTs的值即可，那么具体公式是什么呢?

RFC6298建议的超时重传时间RTO的值计算如下：

```
RTO = RTTs + 4 * RTTd
```

其中RTTs的计算方式如上所述，而RTTd是指RTT偏差的加权平均值，计算方法如下：

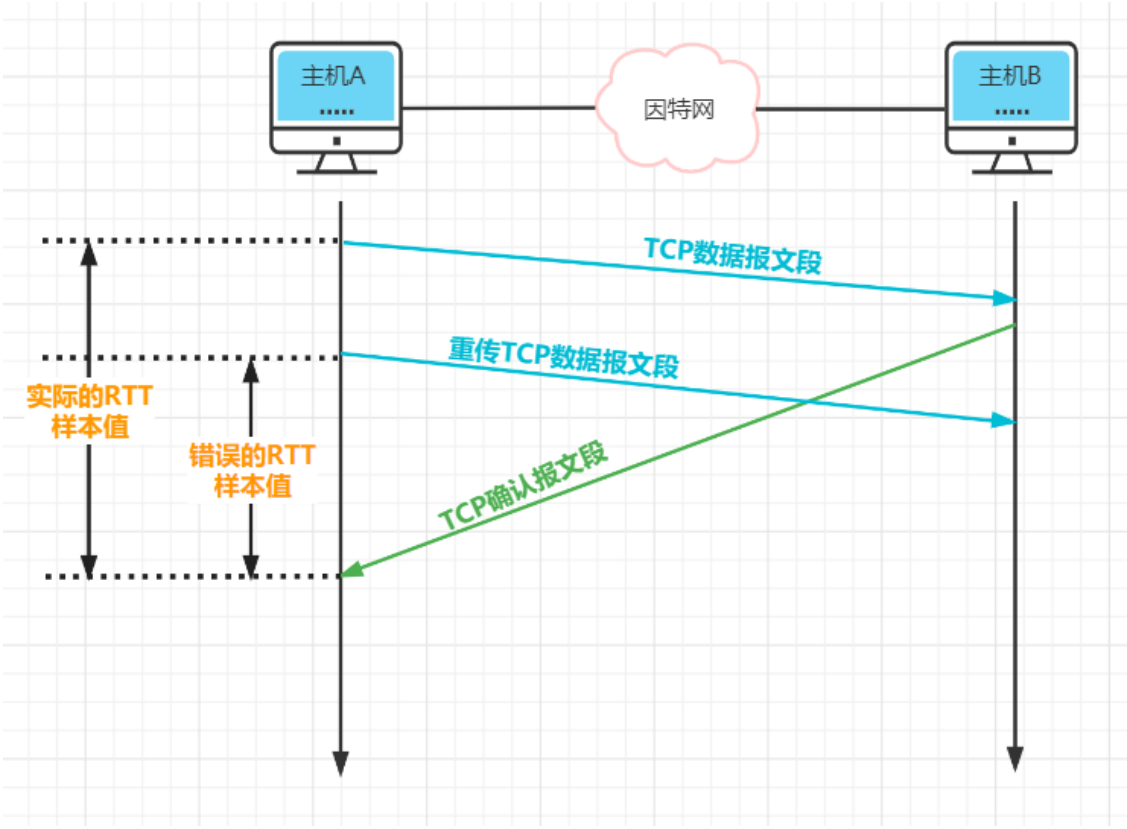
当测量到第一个RTT样本时，RTTd的值取为第一个样本值的一半，即RTTd1=RTT1/2，当继续收到RTT样本后的计算方式是：。

```
新的RTTd = (1-β) * 旧的RTTd + β * |RTTs-新的RTT样本|
```

**已成为建议标准的RFC6298推荐的β值是1/4，即0.25。**

可以看到，**不管是计算RTTs还是RTTd，都是基于所测量到的RTT样本进行测量的**。

也可以看出来，**RTT时间测量值的准确度就显得很重要了**，一旦RTT测量时间不准，将引起最终的RTO计算不准，**实际上RTT的测量确实是比较复杂的**，为什么这么说呢，下面举例说明下其困难性。




主机A给主机B发送一个报文段，主机A迟迟未收到主机B的确认报文段，引发主机A的超时重传，此时主机B的确认报文段姗姗到达主机A，**实际上这个确认报文对应第一次发送的报文段，而不是对应重发的报文段**。

**主机A误以为是针对重发的数据报文段的确认，计算出来的RTT样本值偏小**，最终导致RTO计算出来的也偏小，导致报文出现不必要的重传，增大网络负荷。

可以看出来，一旦出现超时重传，主机就很可能出现误判，导致RTO的值估算偏差变大。

针对出现超时重传时无法测准往返时间RTT的问题，Karn提出了一个算法：在计算加权平均往返时间RTTs时，只要报文段重传了，就不采用其往返时间RTT样本。

也就是说出现报文段重传时，不重新计算RTTs，进而超时重传时间RTO也不会重新计算。

不过这样也引起了新的问题，设想下：网络中报文段的时延突然增大了很多，并且之后很长的一段时间内都会保持这种较大时延，因此在原来得出的重传时间内，不会按时收到确认报文段，于是就重传报文段。但根据Karn算法，不考虑重传报文段的往返时间样本，这样超时重传时间无法得到更新，这会导致报文段反复被重传。

**因此要对Karn算法进行修正**，方法是：报文段每重传一次，就把超时重传时间RTO增大一些，典型做法是将新RTO的值取为旧RTO值的2倍。

这种修正方式比较粗暴，但是非常简单。

一个小小的时间计算，可以看得出来也算是历经多次磨难，不得不说，设计TCP的大师们真的是大师！

提到大师两个字，不禁让我想起一句经典台词：

> 真正的大师永远都怀着一颗学徒的心

大师们都如此谦虚低调、持续思考，作为普通人的我们，有什么理由停止思考呢？