第4章 T/TCP协议(续)

4.1 概述

本章继续讨论T/TCP协议。我们首先讨论T/TCP客户程序如何根据连接持续时间是否会大于报文段最大生存时间MSL来分配端口号，以及这个分配结果对TCP的TIME_WAIT状态有什么影响。接下来我们研究TCP协议为什么要定义TIME_WAIT状态，因为人们对TCP协议的这一特点普遍缺乏理解。T/TCP协议的重要优点之一就是在连接持续时间小于报文段最大生存时间MSL时，使协议的TIME_WAIT状态由240秒缩短至大约12秒。我们将讨论T/TCP协议是如何实现这一点的，以及这样做的正确性。

本章最后我们将讨论T/TCP协议的TAO，即TCP加速打开。它使T/TCP的客户-服务器事务能够跳过三次握手过程，从而节省了一次往返时间，这也正是T/TCP协议给我们带来的最大好处。

4.2 客户的端口号和TIME_WAIT状态

我们编写TCP客户程序的时候通常不关心如何选择端口号。大部分TCP客户程序(如Telnet、FTP以及WWW等)都是使用临时端口，让主机的TCP模块选择一个当前未使用的端口。从伯克利演变来的系统往往选择1 024~5 000之间的临时端口(见图14-14)，而Solaris则在32 768~65 535之间选择。然而，T/TCP协议根据事务速率和持续时间，对端口号的选择有额外的要求。

常规的TCP主机和常规的TCP客户程序

图4-1描述的是一个TCP客户程序(例如图1-5所示的程序)与同一个服务器之间执行的三次事务，每次事务的持续时间为1秒，前后事务之间的间隔也为1秒。三次连接分别开始于第0秒、第2秒和第4秒，而分别终止于第1秒、第3秒和第5秒。*x*轴表示时间，单位为秒；三次连接分别用粗线段表示。

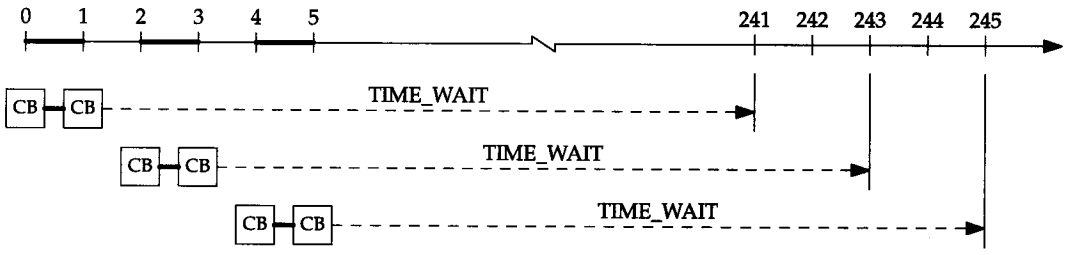

图4-1 TCP客户，不同的事务选用不同的本地端口

每次事务各建立一条TCP连接。我们假定客户程序在创建插口时并不显式地将其绑定到某个端口，而是让系统的TCP模块来选择临时端口。我们还假定客户端TCP模块的报文段最大

生存时间MSL为120秒。第1条连接要保持在TIME_WAIT状态，直至第241秒；第2条和第3条连接则分别从第3秒和第5秒开始保持TIME_WAIT状态，直至第243秒和第245秒。

在图中，CB表示“控制块”，实际上表示连接使用期间和处于TIME_WAIT状态期间，TCP协议维持的几个控制块的组合，包括：Internet进程控制块PCB、TCP控制块和首部模板。在第2章一开始时我们就说过，在Net/3实现中，这3个控制块的大小总和为264字节。除了内存要求以外，TCP协议还需要占用CPU时间来周期性地处理这些控制块(例如在卷2的25.4节和25.5节中，协议每200ms和500ms就要对所有TCP控制块处理一遍)。

> Net/3中为每个连接保存一份TCP和IP首部作为“首部模板”(卷2的第26.8节)。该模板中包含了给定连接中用到的所有字段，这些字段在该连接中不会有变化。这样就节省了每次发送报文段的处理时间，因为程序代码只要把首部模板中的内容复制到正在构造的输出分组中即可，而不需要分别填写每个字段。

常规的TCP是无法跳过三次握手过程的。客户程序不能在相继的3条连接中使用同一个本地端口，即使为插口设置了SO_REUSEADDR属性也是如此(卷2第592页给出了一个示例程序)。

T/TCP主机，每次事务用不同的客户端口

图4-2给出的是与图4-1一样的三次事务序列，但这里我们假定两端的主机都支持T/TCP协议。我们的客户程序与图4-1中的也是同一个。这有很重要的区别：客户和服务器应用程序不需要知道是TCP还是T/TCP，我们只要求两端的主机都支持T/TCP协议(即支持CC选项)。

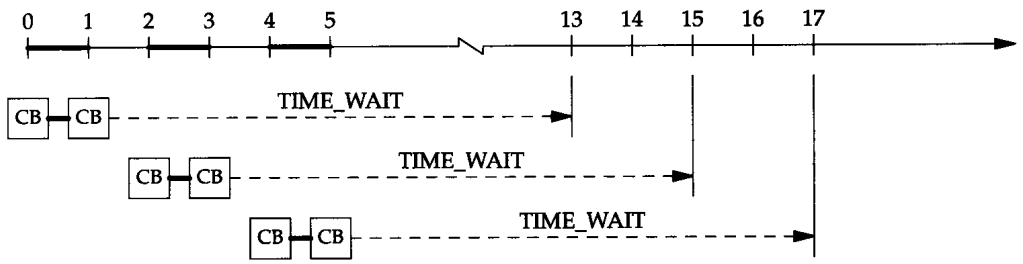

图4-2 当客户和服务器端都支持T/TCP协议时的TCP客户程序

图4-2与图4-1的不同之处在于，连接处于TIME_WAIT状态的时间被截断了，因为两端的主机都支持CC选项。我们这里假定重传超时是1.5秒(在局域网上运行的Net/3中，这是典型值，见[Brakmo and Peterson 1994])，T/TCP的TIME_WAIT是8倍，这样就将TIME_WAIT的保持时间从240秒缩短到了12秒。

当两端的主机都支持CC选项并且连接持续时间小于报文段最大生存时间MSL(120秒)时，T/TCP允许TIME_WAIT状态的保持被截断。这是因为CC选项提供了另外一种保护机制，可以防止过时的重复报文段被投递给另一个新的连接，这一点将在4.4节中讨论。

T/TCP主机，各次事务用同一个客户端口

图4-3给出了与图4-2相同的三次事务的序列，但不同的是，我们在这里假定每次事务中客户端都重复使用同一个端口。为了做到这一点，客户程序必须为插口设置SO_REUSEADDR选

项，并调用bind函数将该插口绑定到某一个特定的本地端口，然后再调用 connect函数(对常规的TCP客户程序)或sendto函数(对T/TCP客户程序)。与图4-2中一样，这里也假定两端的主机都支持T/TCP协议。

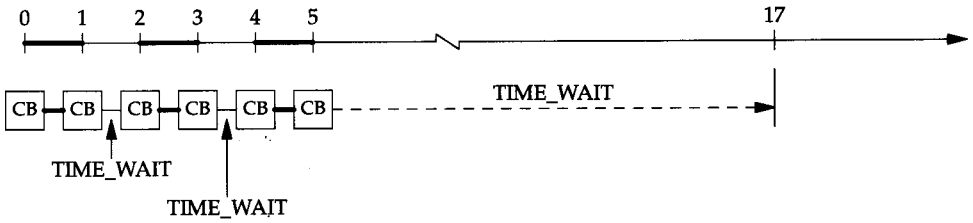

图4-3 TCP客户程序重用同一个端口；客户和服务器主机同时支持 T/TCP协议

在第2秒和第4秒创建连接时， TCP发现了具有相同插口对的控制块，并且正处于TIME_WAIT状态。但是由于前一条连接替身使用了 CC选项，尽管连接的持续时间小于报文段最大生存时间MSL，TIME_WAIT状态的持续时间还是被截断了，并且，当前的连接控制块将被删除，系统将为新的连接分配一个控制块 (新分配的连接控制块可能就是刚刚被删除的旧连接控制块，但那是实现的细节问题。重要的是当前连接控制块的总数没有增加)。当第3条连接在第5秒被关闭后，TIME_WAIT状态的持续时间也只有12秒，与图4-2所示的一样。

总之，本节说明了事务过程中的客户程序有两种可能的优化方式：

1) 不需要改动任何程序源代码，只要客户和服务器端都支持 T/TCP协议，就可将TIME_WAIT的持续时间缩短到连接中重传超时的8倍，而不是原来的240秒。

2) 只修改客户程序，使其重用同一个端口号，这时不但 TIME_WAIT状态的持续时间可以像前一种情况那样截断到连接中重传超时的 8倍，而且，如果同一连接的另一个替身被创建，TIME_WAIT状态就会更快地终止。

4.3 设置TIME_WAIT状态的目的

TIME_WAIT状态是TCP协议中最容易被误解的特性之一。这很可能是因为最初的规约RFC 793中只对该状态做了扼要的解释，尽管后来的 RFC，如RFC 1185，对TIME_WAIT状态做了详细说明。设置TIME_WAIT状态的原因主要有两个：

1) 它实现了全双工的连接关闭。

2) 它使过时的重复报文段作废。

下面我们对这两个原因做进一步的讨论。

TCP全双工关闭

图4-4给出了一般情况下连接关闭时的报文段交换过程。图中还给出了连接状态的变迁和在服务器端测得的RTT值。

图中左侧为客户端，右侧为服务器端。要注意，其中的任何一端都可以主动关闭连接，但一般都是客户端执行主动关闭。

下面我们来看看最后一个报文段 (最后一个ACK)丢失时会发生什么现象。这个现象就给在图4-5中。

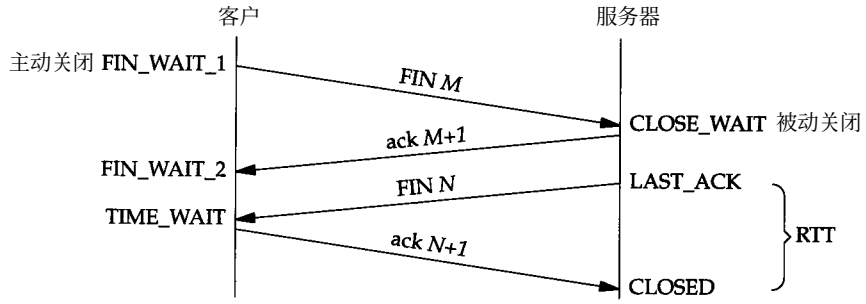

图4-4 通常情况下连接关闭时的报文段交换

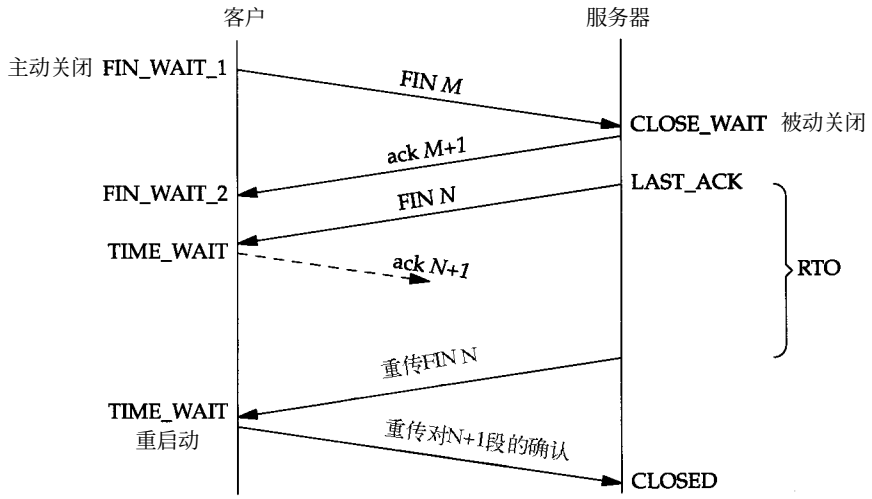

图4-5 最后一个报文段丢失时的TCP连接关闭

由于没有收到客户的最后一个确认，服务器会超时，并重传最后一个 FIN报文段。我们特意把服务器的重传超时(RTO)给得比图4-4中的RTT大，这是因为RTO的取值是估计的RTT值加上若干倍的RTT方差(卷2的第25章详细论述了如何测量 RTT值以及如何计算RTO)。处理最后一个FIN报文段丢失的方法也是一样：服务器在超时后继续重传 FIN。

这个例子说明了为什么 TIME_WAIT状态要出现在执行主动关闭的一端：该端发出最后一个ACK报文段，而如果这个 ACK丢失或是最后一个 FIN丢失了，那么另一端将超时并重传最后的FIN报文段。因此，在主动关闭的一端保留连接的状态信息，这样它才能在需要的时候重传最后的确认报文段；否则，它收到最后的 FIN报文段后就无法重传最后一个 ACK，而只能发出RST报文段，从而造成虚假的错误信息。

图4-5还说明了另一个问题，即如果重传的 FIN报文段在客户端主机仍处于 TIME_WAIT状态的时候到达，那么不仅仅最后一个 ACK会重传，而且TIME_WAIT状态也重新开始。这时，TIME_WAIT状态的持续时间定时器重置为2倍的报文段最大生存时间，即2MSL。

问题是，执行了主动关闭的一端，为了处理图 4-5所示的情况，需要在 TIME_WAIT状态保持多长的时间？这取决于对端的 RTO值；而RTO又取决于该连接的RTT值。RFC 1185中指出RTT的值超过1分钟不太可能。但实际上 RTO却很有可能长达1分钟：在广域网发生拥塞期间时就会有这种情形。这是因为拥塞会导致多次重传的报文段仍然丢失，从而使 TCP的指数退避算法生效，RTO的值越来越大。

过时的重复报文段失效

设置TIME_WAIT状态的第二个原因是为让过时的重复报文段失效。 TCP协议的运行基于一个基本的假设，即：互连网上的每一个 IP数据报都有一个有限的生存期限，这个期限值是由IP首部的TTL(生存时间)字段决定的。每一台路由器在转发 IP数据报时都要将其 TTL值减1；但如果该IP数据报在路由器中等待的时间超过 1秒，那就要把TTL的值减去等待的时间。实际上，很少有IP数据报在路由器中的等待时间超过 1秒的，因而每个路由器通常都是把 TTL的值减1(RFC 1812 [Baker 1995])的5.3.1节)。由于TTL字段的长度是8比特，因此每个IP数据报所能经历的转发次数至多为255。

RFC 793把该限制定义为报文段最大生存时间 MSL，并规定其值为2分钟。该RFC同时指出，将报文段最大生存时间MSL定义为2分钟是一个工程上的选择，其值可以根据经验进行修改。最后， RFC 793规定TIME_WAIT状态的持续时间为MSL的2倍。

图4-6给出的是一个连接关闭后在 TIME_WAIT状态保持了 2倍报文段最大生存时间(2MSL)，然后发起建立新的连接替身。

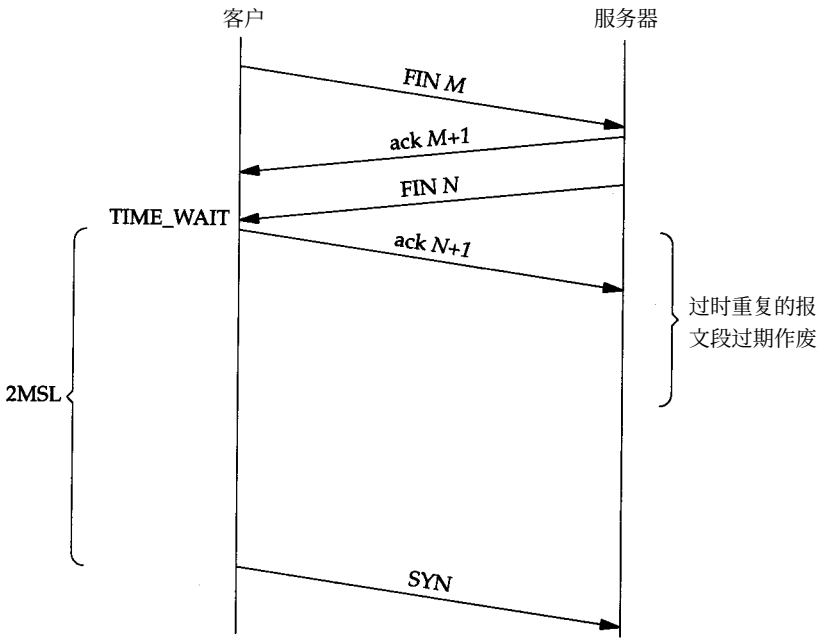

图4-6 前一个连接替身关闭2倍报文段最大生存时间后发起该连接的一个新的替身

由于该连接的新的替身必须在前一个连接替身关闭 2MSL之后才能再次发起，而且由于前一个连接替身的过时重复报文段在 TIME_WAIT状态的第1个报文段最大生存时间里就已经消失，因此我们可以保证前一次连接的过时重复报文段不会在新的连接中出现，也就不可能被误认为是第二次连接的报文段。

TIME_WAIT状态的自结束

RFC 793中规定，处于TIME_WAIT状态的连接在收到RST后变迁到CLOSED状态，这称为TIME_WAIT状态的自结束。 RFC 1337 [Braden 1992a]中则建议不要用 RST过早地结束

TIME_WAIT状态。

4.4 TIME_WAIT状态的截断

我们从图4-2和图4-3中已经看到，T/TCP协议可以截断TIME_WAIT状态的保持时间。采用T/TCP协议后，保持时间由报文段最大生存时间MSL的2倍缩短为RTO(重传超时)的8倍。在4.3节，我们也看到了设置TIME_WAIT状态的原因有两个。那么，截断了该状态的保持时间后，对应于每一个原因都分别产生了什么样的后果？

TCP全双工关闭

设置TIME_WAIT状态的第一个原因是最后一个FIN的重传保持所需的信息。如图4-5所示，花在TIME_WAIT状态的时间实际上应该根据RTO来定，而不是根据MSL。T/TCP中选用乘数8是为了保证对方有足够的时间超时，并重传最后一个报文段。这样就产生了如图 4-2所示的情形：双方都在等待截断的TIME_WAIT保持期(图中为12秒)过去。

但是让我们来看看图4-3中发生的情况。那里因新的客户程序又使用了同一个插口对，从而TIME_WAIT状态在8倍RTO的保持时间到期之前就被截断了。图4-7给出了一个例子。

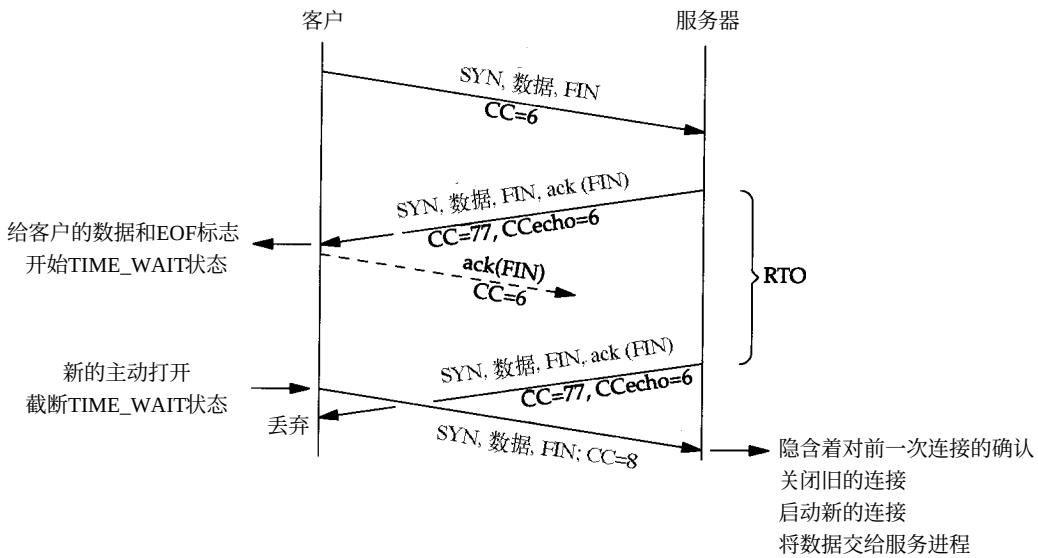

图4-7 最后一个ACK丢失时TIME_WAIT状态被截断的情形

图中，最后一个ACK丢失了，但是客户在收到服务器重传的最后一个报文段之前又再次发起了同一个连接的另一个替身。当服务器收到新的SYN报文段时，由于TAO测试成功(因为8大于6)，这就隐含着确认了服务器的待确认的报文段(图中第2个报文段)。老的连接于是关闭，新的连接开始建立。由于TAO测试成功，新SYN报文段中的数据被交付给服务器进程。

有意思的是，由于T/TCP协议把TIME_WAIT状态的保持时间定义为执行主动关闭一端所测得的RTO的函数，因而就有了一个隐含的假定：两端测得的RTO值相近，并且在一定的范围之内[Olah 1995]。如果执行主动关闭的一端在另一端重传最后的FIN之前就结束了TIME_WAIT状态，那么对重传FIN报文段的响应将是RST而不是重

传端所等待的ACK。

当RTT的值较小，最小的3个T/TCP交换报文段中的第3个报文段丢失，以及客户端和服务器端具有不同的软件时钟速率和不同的 RTO最小值时，就会发生上述情况(第14.7节给出了客户常用的一些 RTO值)。无论如何，当服务器无法测量 RTT的时候(由于第3个报文段丢失)，客户可以测出较小的 RTT值。例如，假设客户测得的 RTT值为10ms，RTO的最小值为100ms，这时客户在收到服务器响应 800ms以后就截断TIME_WAIT状态。但如果服务器是从伯克利演变而来的版本，那么其缺省的 RTO为6秒(如图14-13所示)。当服务器在大约6秒后重传其SYN/ACK/data/FIN时，客户端将发出一个RST作为响应，给服务器应用程序造成一个虚假的错误。

过时的重复报文段失效

TIME_WAIT状态的截断是可行的，因为CC选项能够防止过时重复报文段错误地传递给后续连接。但截断的前提是连接的持续时间小于报文段最大生存时间MSL。考虑图4-8所示的情况。我们让CC生成器(tcp_ccgen)以最大可能速率递增：每两个报文段最大生存时间 (2MSL)就增长$2^{32}-1$。这使得事务速率达到最大，为4 294 967 295除240，大约为每秒18 000 000次事务。

假设tcp_ccgen的值在时刻0时为1，并以上述最大速率递增，那么在 2MSL、4MSL等时刻，tcp_ccgen的值又重新回到1。而且由于tcp_ccgen的值永远不取0，在2MSL的时间里只有$2^{32}-1$个值而不是2^{32}个；因此我们在图中画出 MSL时的2 147 483 648这个值实际上是在MSL时刻之前很短的时间里出现。

我们假设连接始于时刻 0，CC值为1，连接持续时间为 100秒。TIME_WAIT状态从第100秒开始，一直保持到第 112秒，或者在主机发起下一次连接时提前结束 (这里假定RTO为1.5秒，因此TIME_WAIT状态的保持时间为 12秒)。由于连接的持续时间 (100秒)小于报文段最大生存时间MSL(120秒)，因而可以保证该次连接的所有过时重复报文段在第 220秒以后一定会消失。我们还假定tcp_ccgen计数器是以最大可能速率递增的。也就是说，主机在第 0~240秒的时间里建立超过40亿条TCP连接。

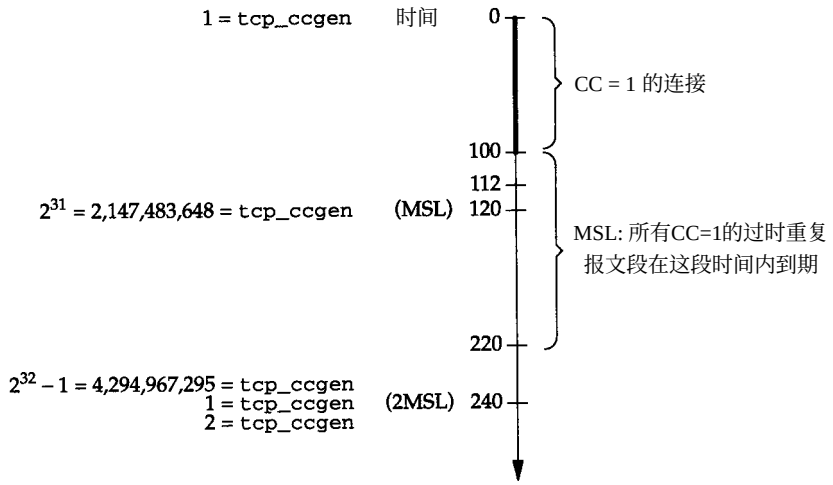

图4-8 持续时间小于MSL的连接：TIME_WAIT状态截断是可行的

因此，只要连接的持续时间小于报文段最大生存时间 MSL，将TIME_WAIT状态的保持时间截断就是安全的，因为CC选项的值直到所有的过时重复报文段都消失以后才会重复。

要明白为什么只有当连接的持续时间小于报文段最大生存时间 MSL时才能截断TIME_WAIT状态的保持时间，那得考察图4-9所示的情形。

我们仍然假设tcp_ccgen计数器以可能的最快速率递增。某个连接开始于时刻 0，CC值为2，持续时间为140秒。由于持续时间大于报文段最大生存时间 MSL，故TIME_WAIT状态不能被截断，从而该连接的插口对只有等到第 380秒以后才能被重用 (从技术上讲，由于我们给定tcp_ccgen在0时刻的值为1，故CC值为2的连接会在0时刻稍后建立，并在第140秒稍后终止。但这不影响我们的讨论)。

在第240~260秒之间，CC值2可以重用。如果 TIME_WAIT状态被截断(比如发生在第140秒~第152秒之间的某个时刻)，并且如果同一条连接的另一次实现在第 240~第260秒之间重新建立且CC值为2，那么由于所有过时的重复报文段只有在第 260秒以后才会全部消失，这样那些属于前一次连接的过时重复报文段就有可能被误认为是第 2次连接的新报文段。在第240~260秒这段时间，其他的连接 (即不同的插口对)将CC的值取为2并没有什么问题；但是同一个插口对就不能重复使用这个 CC值，因为此时网络中可能还有属于该连接的过时重复报文段。

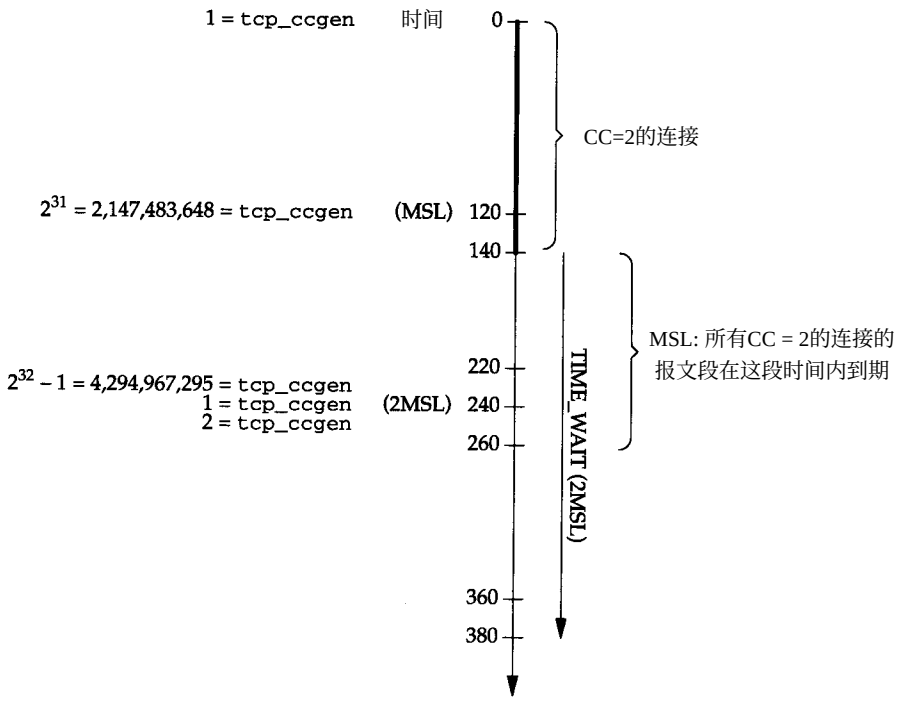

图4-9 持续时间大于MSL的连接：TIME_WAIT状态无法被截断

从应用程序的角度而言，所谓 TIME_WAIT状态截断就意味着客户程序在与同一个服务器进行一系列事务时，必须选择是让一系列连接使用同一个本地端口，还是让每次事务使用各自不同的本地端口。当连接的持续时间小于报文段最大生存时间 MSL时(在事务中这是典型的情况)，重用本地端口可以节约TCP资源(即减少了对控制块内存的需求，见图 4-2和图4-3)。但

是如果客户程序在前一次连接的持续时间大于报文段最大生存时间 MSL的情况下试图重用本地端口，建立连接时将返回EADDRINUSE错误(图12-2)。

如图4-2所示，无论应用程序采用哪种端口使用策略，如果两端的主机都支持 T/TCP协议，而且连接的持续时间小于报文段最大生存时间 MSL，那么TIME_WAIT状态的保持时间总是可以从2倍MSL截断到8倍RTO。这样就节约了资源(即内存和CPU时间)。这对支持T/TCP协议的2台主机之间的任何 TCP连接(如FTP、SMTP、HTTP以及其他等)，只要连接持续时间小于MSL就都适用。

4.5 利用TAO跳过三次握手

T/TCP协议的主要好处就是能够跳过三次握手。为了理解何以能跳过三次握手，我们需要先了解三次握手的目的。 RFC 793中对此只是做了一个简单的说明：“ 引入三次握手的主要原因是为了避免过时的重复连接在再次建连时造成的混乱。为此，专门定义了一个特殊的控制报文reset来解决这个问题 ”。

在三次握手中，每一端都是先发出 SYN报文段，其中含有各自的起始序列号；然后每一端都要确认对方的SYN报文段。这样就可以排除当重复的过时报文段到达某一端时可能带来的混淆。此外，常规的TCP不会在进入ESTABLISHED状态之前就把在SYN报文段中一起传送过来的数据交付给上层的用户进程。

T/TCP协议必须提供一种方法，使收到SYN报文段的一方能够不经过三次握手就保证这个SYN不是过时的重复报文段，从而使得随该 SYN报文段一起传送过来的数据能立刻交付给上层的用户进程。这里的保护手段是客户发出的 SYN报文段中附带的CC选项和服务器缓存的最近一次从该客户收到的合法CC值。

考虑图4-10时序图所示的情况。与图4-8中一样，我们假设tcp_ccgen计数器以最大可能速率递增：即每2MSL递增$2^{32}-1$。

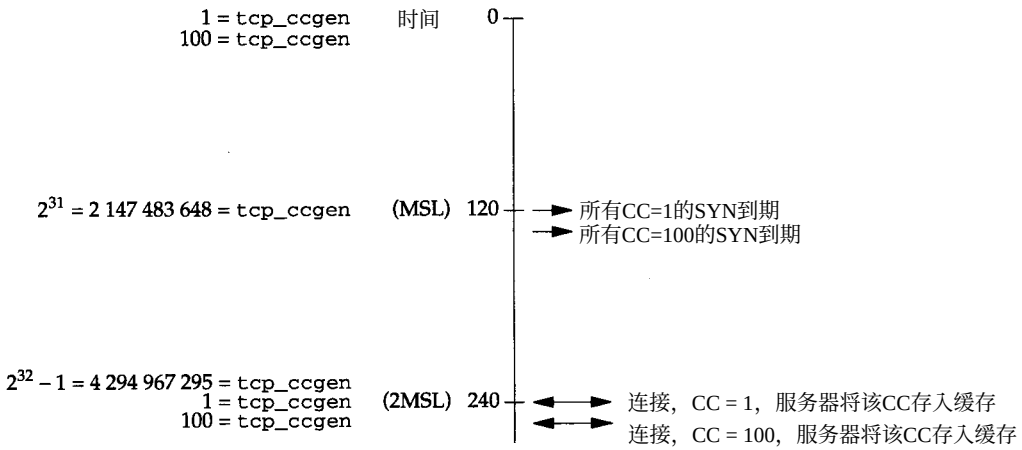

图4-10 SYN中较高的CC值保证该SYN不是过时复制品

在0时刻，tcp_ccgen的值为1；很小的一段时间后其值就变为 100。由于IP数据报的生存期有限，我们可以确信在第 120秒的时候(MSL秒以后)，CC值为1的所有SYN报文段都已经过期而在网络中消失；此后再过一小段时间，所有 CC值为100的SYN报文段也都消失了。

于是在第240秒的时候，又建立了一个CC值为1的新连接。假设服务器对该报文段的 TAO测试成功，那么它就把该客户的最新 CC值缓存下来。此后不久，该服务器主机又收到同一客户的另一个连接请求。由于这时 SYN报文段中的CC的值(为100)比服务器当前保存的该客户最近一个合法CC值(1)要大，而且以前CC值为100的连接中的所有 SYN都已经超过了MSL时间，因而服务器可以断定这是一个新的 SYN。

事实上，这就是RFC 1644中所述的TAO测试：“ 如果某个特定客户主机的第一个 SYN报文段(即只含SYN位而不含ACK位的报文段)中所携带的CC值大于缓存中的该客户 CC值， CC值的单调递增特性可以确保这是一个新的 SYN报文段，可以立即接收下来 ”。正是CC值的单调递增特性以及下面的两个假设确保了 SYN报文段是新的，使得T/TCP协议能够跳过三次握手：

1) 所有的报文段都只有有限的MSL秒的生存期；

2) `tcp_ccgen`计数器在2MSL的时间内的递增量不超过 $2^{32}-1$。

失序的SYN报文段

图4-11给出了两台T/TCP主机和一个失序到达的SYN报文段。这个SYN并不是一个过时重复报文段，只不过是没有按照正确的顺序到达服务器。

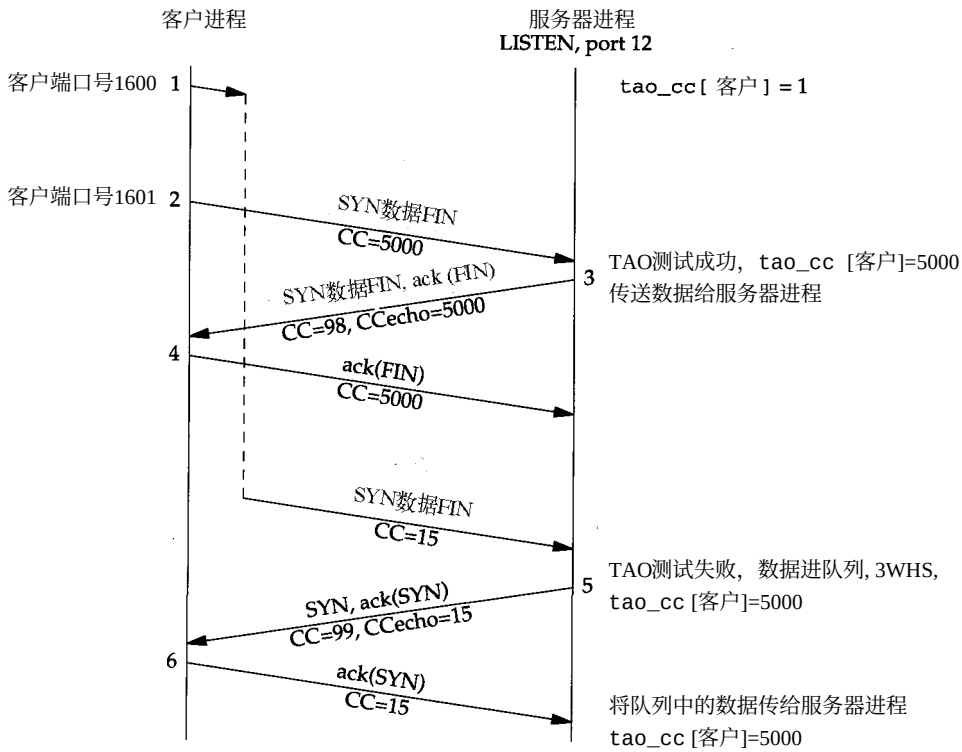

图4-11 两台T/TCP主机和失序到达的SYN报文段

服务器缓存的该客户的CC值为1。报文段1是从客户的端口1600发出的，携带的CC值为15，但它在网络上延迟了一段时间。报文段 2是从客户的端口 1601发出的，携带的 CC值为5000。当报文段2到达服务器时， TAO测试成功(5000大于1)，于是对该客户缓存的最新 CC值改为5000，并把数据交付给进程。报文段3和报文段4完成该次事务。

当报文段1终于到达服务器时，TAO测试失败(15小于5000)，于是服务器给出的响应也是一个SYN报文段，其中带有对所收到SYN的ACK，强迫执行三次握手过程。只有当三次握手完成后才会把数据交付给服务器进程。报文段6结束三次握手过程，队列中的数据于是被交付给服务器进程(图中没有给出该事务的后续过程)。但是，尽管三次握手成功结束，CC值为15的报文段也并不是一个过时的重复报文段(它只是到达的顺序不对)，服务器所记录的该客户的CC值并不更新。如果更新CC会使其值由5000回到15，可能会使服务器错误地收下来自该客户的、CC值介于15~5000之间的过时重复报文段。

翻转了符号位的CC值

在图4-11中我们看到，当TAO测试失败后，服务器强迫执行三次握手；即使握手过程成功结束，服务器所记录的该客户CC值也不更新。从协议的角度出发，这样做是正确的，但却降低了效率。

服务器端TAO测试失败是很可能发生的，因为CC值是客户端生成的。对客户端来说，这是所有连接的全局变量；对服务器来说，则是“翻转了它们的符号位(wrap their sign bit)”。(类似于TCP的序列号，CC值也是无符号的32位数。对CC值进行比较采用模运算，具体算法可见卷2第649~650页。当我们说CC值a的符号位相对于值b“翻转”了时，其具体含义是a的值增加后不再比b大了，反倒是比b小了)。看图4-12。

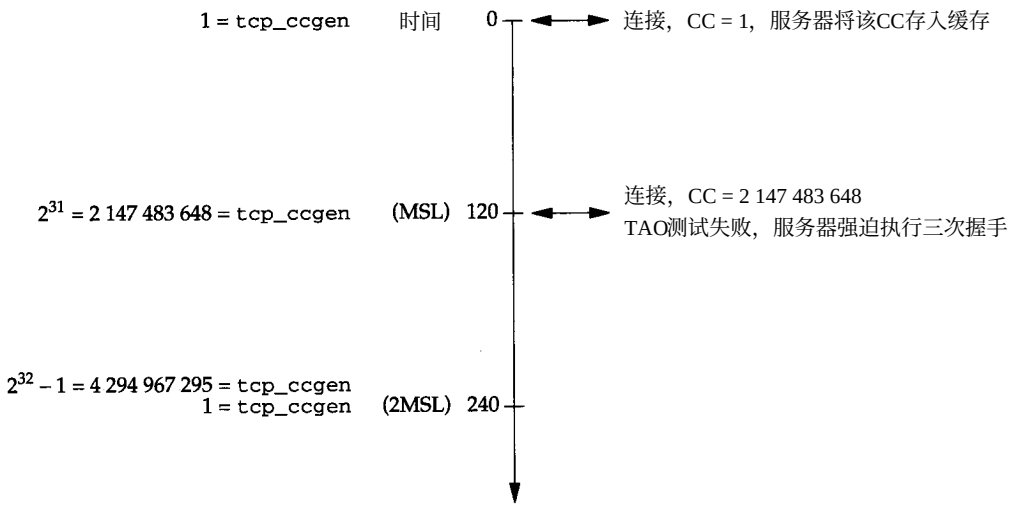

图4-12 CC的值翻转其符号位造成TAO测试失败

客户在0时刻与服务器建立连接，CC值为1。服务器TAO测试成功，并把该CC值记录为该客户当前的CC值。接着该客户与其他服务器建立了2 147 483 646个连接。在第120秒时，它与0时刻建立起连接的服务器又建立一个连接，但此时的CC值为2 147 483 648。服务器收到SYN后，TAO测试失败(按模运算，2 147 483 648比1小，如卷2的图24-26所示)，然后三次握手过程验证了该SYN报文段，但是服务器当前记录的该客户的CC值仍然是1。

这就意味着，从此开始到第240秒的这一段时间里，该客户向该服务器发送任何一个SYN都要经过三次握手过程。这里假设tcp_ccgen计数器持续以最快的速率递增。但更可能的情况是该计数器的递增速率会低得多，意味着计数器由2 147 483 648递增到4 294 967 295就不

只需要120秒了，而可能是几个小时甚至几天的时间。可是在该计数器的符号位再次翻转以前，该客户和该服务器之间的所有T/TCP连接都要执行三次握手。

这个问题的解决需要连接双方的共同努力。首先，不仅服务器要为每个客户记录其最新的CC值，而且客户也要记录发给每个服务器的最新CC值。这两个变量即为图2-5中的tao_cc和tao_ccsent。

其次，当客户发现它将要使用的tcp_ccgen值小于它最近发送给服务器的CC值时，客户就发出CCnew选项而不是CC选项。该选项强迫两台主机再次同步它们各自记录的CC值。

服务器收到一个带有CCnew选项的SYN报文段后，它把该客户的tao_cc标记为0(未定义)。三次握手成功结束后，如果TAO缓存中该客户的表项还是未定义，就将其更新为这次收到的CC值。

由此可见，当客户端没有记录该服务器的CC值(比如客户端重启，或是缓存中对应于该服务器的表项被冲掉)时，或者客户端检测到该服务器的CC值已翻转时，该客户将在其初始SYN报文段中发送CCnew选项而不是CC选项。

重复的SYN/ACK报文段

到目前为止，我们的注意力一直集中在服务器如何确定所收到的SYN报文段是新的还是过时和重复的。这使得服务器可以绕开三次握手的过程。但是客户端又如何确定所收到的服务器响应(SYN/ACK报文段)不是过时和重复的呢？

在常规的TCP协议中，客户端发送的SYN报文段中不带数据，因此服务器要确认的内容只有一个字节：SYN。此外，从伯克利演变来的实现又在建立新连接时将初始发送序号(ISS)递增64 000(TCP_ISSINCR除以2) (见卷2第808页)，这样客户端发送的后续SYN中的初始发送序号就总是比前一次连接的要大。因而过时的重复SYN/ACK报文段就不太可能含有客户端可接受的确认字段。

然而在T/TCP协议中，客户端的SYN报文段中通常都带有数据，这就扩大了客户端从服务器收到可接受确认的范围。RFC 1379 [Braden 1992b]的图7给出的一个例子中，客户端就错误地接受了一个过时的重复SYN/ACK报文段。然而，该例子出现这个问题的原因在于前后两个相继连接的初始发送序号只差100，小于客户端在SYN报文段中附带的数据字节数(300字节)。我们前面提到过，从伯克利演变来的实现中，每建立一个新的连接时总会把ISS增加至少64 000。而64 000已经大于T/TCP协议的默认发送窗口宽度(通常为4 096字节)，这就使客户端不可能接受一个过时的重复SYN/ACK报文段。

> 4.4BSD-Lite2系统中采用了我们在3.2节中讨论过的随机产生ISS值方法。ISS的实际增量平均分布于31 232和96 767之间，均值为63 999。31 232仍然比默认的发送窗口大，下面我们将要讨论的CCecho选项使得这个问题存有争议。

然而，T/TCP协议用CCecho选项对过时的重复SYN/ACK报文段问题提供完整的保护。客户端知道自己发出的SYN报文段中的CC值，而服务器必须在CCecho选项中把该值原样发回给客户端。如果服务器的响应中不含所期望的CCecho值，那么客户端就把该响应丢弃(图11-8)。CC的值具有单调递增特性，而且在至多2 MSL秒的时间内就循环一次，这就可以保证客户端不会接受过时的重复SYN/ACK报文段。

注意，客户不能对服务器传来的SYN/ACK报文段执行TAO测试：太晚了。服务器已经接

受了客户端的SYN，数据已经被交付给了服务器进程，而客户收到的 SYN/ACK报文段中也含有服务器给出的响应。此时客户端再要强迫执行三次握手就太迟了。

重传的SYN报文段

RFC 1644和我们在本节中的讨论都忽略了客户端的 SYN或服务器的SYN的重传可能性。例如，在图4-10中，我们假设tcp_ccgen在0时刻的值为1；接着假设所有CC值为1的SYN报文段在第120秒时都已经过时消失。而实际上也可能是这样的： CC值为1的SYN报文段可能在第0~75秒之间的某个时刻重传了，于是所有CC值为1的SYN报文段在第195秒时才会过时消失，而不是在第120秒时就过时了(如卷2第664页所述，从伯克利演变来的实现中，客户端 SYN报文段和服务器SYN报文段的重传超时上限为75秒)。

这并不影响TAO测试的正确性，但却降低了tcp_ccgen计数器的最大递增速率。前面我们讲过，tcp_ccgen计数器的最大递增速率是在2 MSL秒的时间里递增$2^{32}-1$，从而使最大事务速率达到大约每秒18 000 000次。当考虑到SYN报文段的重传之后，上述最大速率就变为在2 MSL+2 MRX秒的时间里递增$2^{32}-1$，其中MRX是系统规定的SYN报文段重传时限(在Net/3中是75秒)。最大事务速率也由此降低到大约每秒 11 000 000次。

4.6 小结

TCP协议的TIME_WAIT状态有两个功能：

1) 实现了连接的全双工关闭。

2) 使得过时的重复报文段超时作废。

如果T/TCP连接的持续时间小于120秒(1倍的报文段最大生存时间MSL)，那么TIME_WAIT状态的保持时间只要8倍的重传超时，而不是240秒。而且，在前一次连接还处于 TIME_WAIT状态时，客户端就可以建立一个连接的新的替身，从而进一步截短了等待时间。我们说明了为什么这么做是可行的，仅受限于T/TCP协议能支持的最大事务速率(大约每秒18 000 000次)。如果T/TCP客户知道要与同一台服务器执行大量事务，那么它每次都可以使用同一个本地端口号，从而可减少TIME_WAIT状态的控制块的数量。

TCP的TAO(TCP加速打开)测试使得T/TCP的客户 - 服务器程序可以跳过三次握手过程。这是在服务器收到客户端的 CC值大于服务器记录的该客户 CC值时实现的。正是CC值的单调递增性以及以下两点假定才确保了服务器能够断定客户端的 SYN是新的，使T/TCP能够跳过三次握手过程：

1) 所有报文段都有一个有限的生存期限： MSL秒；

2) tcp_ccgen计数器的最大递增速率为：在2MSL秒内增加$2^{32}-1$。