

第22章 TCP的坚持定时器

22.1 引言

我们已经看到TCP通过让接收方指明希望从发送方接收的数据字节数（即窗口大小）来进行流量控制。如果窗口大小为0会发生什么情况呢？这将有效地阻止发送方传送数据，直到窗口变为非0为止。

可以在图20-3中看到这种情况。当发送方接收到报文段9时，它打开被报文段8关闭的窗口并立即开始发送数据。TCP必须能够处理打开此窗口的ACK（报文段9）丢失的情况。ACK的传输并不可靠，也就是说，TCP不对ACK报文段进行确认，TCP只确认那些包含有数据的ACK报文段。

如果一个确认丢失了，则双方就有可能因为等待对方而使连接终止：接收方等待接收数据（因为它已经向发送方通告了一个非0的窗口），而发送方在等待允许它继续发送数据的窗口更新。为防止这种死锁情况的发生，发送方使用一个坚持定时器（persist timer）来周期性地向接收方查询，以便发现窗口是否已增大。这些从发送方发出的报文段称为窗口探查（window probe）。在本章中，我们将讨论窗口探查和坚持定时器，还将讨论与坚持定时器有关的糊涂窗口综合症。

22.2 一个例子

为了观察到实际中的坚持定时器，我们启动一个接收进程。它监听来自客户的连接请求，接受该连接请求，然后在从网上读取数据前休眠很长一段时间。

sock程序可以通过指定一个暂停选项-P使服务器在接受连接和进行第一次读动作之间进入休眠。我们以这种方式调用服务器：

```
svr4 %sock -i -s -P100000 5555
```

该命令在从网络上读数据之前休眠100 000秒（27.8小时）。客户运行在主机bsdi上，并向服务器的5555端口执行1024字节的写操作。图22-1给出了tcpdump的输出结果（我们已经在结果中去掉了连接的建立过程）。

报文段1~13显示的是从客户到服务器的正常的数据传输过程，有9216个字节的数据填充了窗口。服务器通告窗口大小为4096字节，且默认的插口缓存大小为4096字节。但实际上它一共接收了9216字节的数据，这是在SVR4中TCP代码和流子系统(stream subsystem)之间某种形式交互的结果。

在报文段13中，服务器确认了前面4个数据报文段，然后通告窗口为0，从而使客户停止发送任何其他的数据。这就引起客户设置其坚持定时器。如果在该定时器时间到时客户还没有接收到一个窗口更新，它就探查这个空的窗口以决定窗口更新是否丢失。由于服务器进程处于休眠状态，所以TCP缓存9216字节的数据并等待应用进程读取。

请注意客户发出的窗口探查之间的时间间隔。在收到一个大小为0的窗口通告后的第1个

(报文段 14) 间隔为 4.949 秒, 下一个 (报文段 16) 间隔是 4.996 秒, 随后的间隔分别约为 6, 12, 24, 48 和 60 秒。

```

1    0.0          bsdi.1027 > svr4.5555: P 1:1025(1024) ack 1 win 4096
2    0.191961 ( 0.1920) svr4.5555 > bsdi.1027: . ack 1025 win 4096
3    0.196950 ( 0.0050) bsdi.1027 > svr4.5555: . 1025:2049(1024) ack 1 win 4096
4    0.200340 ( 0.0034) bsdi.1027 > svr4.5555: . 2049:3073(1024) ack 1 win 4096
5    0.207506 ( 0.0072) svr4.5555 > bsdi.1027: . ack 3073 win 4096
6    0.212676 ( 0.0052) bsdi.1027 > svr4.5555: . 3073:4097(1024) ack 1 win 4096
7    0.216113 ( 0.0034) bsdi.1027 > svr4.5555: P 4097:5121(1024) ack 1 win 4096
8    0.219997 ( 0.0039) bsdi.1027 > svr4.5555: P 5121:6145(1024) ack 1 win 4096
9    0.227882 ( 0.0079) svr4.5555 > bsdi.1027: . ack 5121 win 4096
10   0.233012 ( 0.0051) bsdi.1027 > svr4.5555: P 6145:7169(1024) ack 1 win 4096
11   0.237014 ( 0.0040) bsdi.1027 > svr4.5555: P 7169:8193(1024) ack 1 win 4096
12   0.240961 ( 0.0039) bsdi.1027 > svr4.5555: P 8193:9217(1024) ack 1 win 4096
13   0.402143 ( 0.1612) svr4.5555 > bsdi.1027: . ack 9217 win 0

14   5.351561 ( 4.9494) bsdi.1027 > svr4.5555: . 9217:9218(1) ack 1 win 4096
15   5.355571 ( 0.0040) svr4.5555 > bsdi.1027: . ack 9217 win 0

16   10.351714 ( 4.9961) bsdi.1027 > svr4.5555: . 9217:9218(1) ack 1 win 4096
17   10.355670 ( 0.0040) svr4.5555 > bsdi.1027: . ack 9217 win 0

18   16.351881 ( 5.9962) bsdi.1027 > svr4.5555: . 9217:9218(1) ack 1 win 4096
19   16.355849 ( 0.0040) svr4.5555 > bsdi.1027: . ack 9217 win 0

20   28.352213 (11.9964) bsdi.1027 > svr4.5555: . 9217:9218(1) ack 1 win 4096
21   28.356178 ( 0.0040) svr4.5555 > bsdi.1027: . ack 9217 win 0

22   52.352874 (23.9967) bsdi.1027 > svr4.5555: . 9217:9218(1) ack 1 win 4096
23   52.356839 ( 0.0040) svr4.5555 > bsdi.1027: . ack 9217 win 0

24   100.354224 (47.9974) bsdi.1027 > svr4.5555: . 9217:9218(1) ack 1 win 4096
25   100.358207 ( 0.0040) svr4.5555 > bsdi.1027: . ack 9217 win 0

26   160.355914 (59.9977) bsdi.1027 > svr4.5555: . 9217:9218(1) ack 1 win 4096
27   160.359835 ( 0.0039) svr4.5555 > bsdi.1027: . ack 9217 win 0

28   220.357575 (59.9977) bsdi.1027 > svr4.5555: . 9217:9218(1) ack 1 win 4096
29   220.361668 ( 0.0041) svr4.5555 > bsdi.1027: . ack 9217 win 0

30   280.359254 (59.9976) bsdi.1027 > svr4.5555: . 9217:9218(1) ack 1 win 4096
31   280.363315 ( 0.0041) svr4.5555 > bsdi.1027: . ack 9217 win 0

```

图22-1 坚持定时器探查一个0大小窗口的例子

为什么这些间隔总是比 5、6、12、24、48 和 60 小一个零点几秒呢? 因为这些探查被 TCP 的 500 ms 定时器超时例程所触发。当定时器时间到时, 就发送窗口探查, 并大约在 4 ms 之后收到一个应答。接收到应答使得定时器被重新启动, 但到下一个时钟滴答之间的时间则约为 500 减 4 ms。

计算坚持定时器时使用了普通的 TCP 指数退避。对一个典型的局域网连接, 首次超时时间算出来是 1.5 秒, 第 2 次的超时值增加一倍, 为 3 秒, 再下次乘以 4 为 6 秒, 之后再乘以 8 为 12 秒等。但是坚持定时器总是在 5~60 秒之间, 这与我们在图 22-1 中观察到的现象一致。

窗口探查包含一个字节的的数据 (序号为 9217)。TCP 总是允许在关闭连接前发送一个字节的的数据。请注意, 尽管如此, 所返回的窗口为 0 的 ACK 并不是确认该字节 (它们确认了包括 9216 在内的所有数据), 因此这个字节被持续重传。

坚持状态与第 21 章中介绍的重传超时之间一个不同的特点就是 TCP 从不放弃发送窗口探查。这些探查每隔 60 秒发送一次, 这个过程将持续到或者窗口被打开, 或者应用进程使用的连接被终止。

22.3 糊涂窗口综合症

基于窗口的流量控制方案, 如 TCP 所使用的, 会导致一种被称为 “糊涂窗口综合症 SWS

(Silly Window Syndrome) ”的状况。如果发生这种情况,则少量的数据将通过连接进行交换,而不是满长度的报文段[Clark 1982]。

该现象可发生在两端中的任何一端:接收方可以通告一个小的窗口(而不是一直等到有大的窗口时才通告),而发送方也可以发送少量的数据(而不是等待其他的数据以便发送一个大的报文段)。可以在任何一端采取措施避免出现糊涂窗口综合症的现象。

1) 接收方不通告小窗口。通常的算法是接收方不通告一个比当前窗口大的窗口(可以为0),除非窗口可以增加一个报文段大小(也就是将要接收的 MSS)或者可以增加接收方缓存空间的一半,不论实际有多少。

2) 发送方避免出现糊涂窗口综合症的措施是只有以下条件之一满足时才发送数据:(a)可以发送一个满长度的报文段;(b)可以发送至少是接收方通告窗口大小一半的报文段;(c)可以发送任何数据并且不希望接收 ACK(也就是说,我们没有还未被确认的数据)或者该连接上不能使用Nagle算法(见第19.4节)。

条件(b)主要对付那些总是通告小窗口(也许比1个报文段还小)的主机,条件(c)使我们在有尚未被确认的数据(正在等待被确认)以及在不能使用Nagle算法的情况下,避免发送小的报文段。如果应用进程在进行小数据的写操作(例如比该报文段还小),条件(c)可以避免出现糊涂窗口综合症。

这三个条件也可以让我们回答这样一个问题:在有尚未被确认数据的情况下,如果Nagle算法阻止我们发送小的报文段,那么多小才算是小呢?从条件(a)中可以看出所谓“小”就是指字节数小于报文段的大小。条件(b)仅用来对付较老的、原始的主机。

步骤2中的条件(b)要求发送方始终监视另一方通告的最大窗口大小,这是一种发送方猜测对方接收缓存大小的企图。虽然在连接建立时接收缓存的大小可能会减小,但在实际中这种情况很少见。

一个例子

现在我们通过仔细查看一个详细的例子来观察实际避免出现糊涂窗口综合症的情况,该例子也包括了坚持定时器。我们将在发送主机sun上运行sock程序,并向网络写6个1024字节的数据。

```
sun % sock -i -n6 bsdi 7777
```

但是在主机bsdi的接收过程中我们加入一些暂停。在第1次读数据前暂停4秒,之后每次读之前暂停2秒。而且,接收方进行的是256字节的读操作:

```
bsdi % sock -i -s -P4 -p2 -r256 7777
```

最初的暂停是为了让接收缓存被填满,迫使发送方停止发送。随后由于接收方从网络上进行了一些小数目的读取,我们预期能看到接收方采取的避免糊涂窗口综合症的措施。

图22-2是传输6144字节数据的时间系列(我们去掉了连接建立过程)。

我们还需要跟踪在每个时间点上读取数据时应用程序的运行情况、当前正在接收缓存中的数据的序号以及接收缓存中可用空间的大小。图22-3显示了所发生的每件事情。

图22-3中的第1列是每个行为的相对时间点。那些带有3位小数点的时间是从tcpdump的输出结果(图22-2)中得到的,而小数点部分为99的则是在接收服务器上产生行为的估计时间(使这些在接收方的估计时间包含一秒的99%仅与图22-2中的报文段20和22有关,它们是我们能

够从tcpdump的输出结果中看到的由接收主机超时引起的仅有的两个事件。而在主机 bsd1上观察到的其他分组, 则是由接收到来自发送方的一个报文段所引起的。这同样是有意义的, 因为这就使我们可以将最初的 4秒暂停刚好放置在发送方发送第 1个数据报文段的时间 0前面。这是接收方在连接建立过程中收到它的 SYN的 ACK之后将要获得控制权的大致时间)。

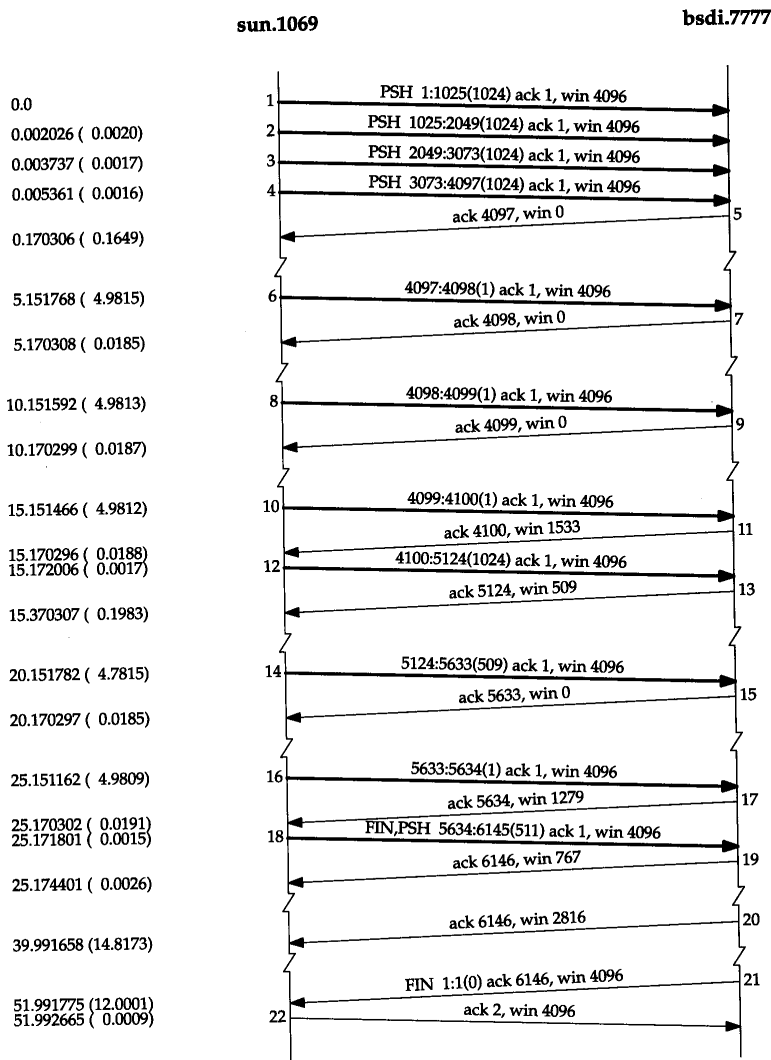


图22-2 显示接收方避免出现糊涂窗口综合症的时间系列

当接收到来自发送方的数据时, 接收方缓存中的数据增加, 而当应用进程从缓存中读取数据时, 数据就减少。接下来我们关注的是接收方发给发送方的窗口通告以及这些窗口通告是什么。这样就可以使我们看到接收方是如何避免糊涂窗口综合症的。

前4个数据报文段及其ACK (报文段1~5) 表示发送方正在填充接收方的缓存。在那个时刻发送方停止了发送, 但仍然有更多的数据需要发送。它将自己的坚持定时器置为最小值 5分钟。

当坚持定时器时间到时, 就发送出 1个字节的数据 (报文段 6)。接收的应用进程已经从接收缓存中读取了 256字节的数据 (在时刻 3.99), 因此这个字节被接受并被确认 (报文段 7段)。但是通告窗口仍为 0, 由于接收方仍然没有足够的空间来接收一个满长度的报文, 或者不能腾

出缓存空间的一半。这就是接收方的糊涂窗口避免措施。

时间	报文段号 (图22-2)	行 为			接收缓冲区	
		发送TCP	接收TCP	应用	数据	可用的
0.000	1	1:1025(1024)			1024	3072
0.002	2	1025:2049(1024)			2048	2048
0.003	3	2049:3073(1024)			3072	1024
0.005	4	3073:4097(1024)			4096	0
0.170	5		ACK 4097, win 0			
3.99				读取256	3840	256
5.151	6	4097:4098(1)			3841	255
5.17	7		ACK 4098, win 0			
5.99				读取256	3585	511
7.99				读取256	3329	767
9.99				读取256	3073	1023
10.151	8	4098:4099(1)			3074	1022
10.170	9		ACK 4099, win 0			
11.99				读取256	2818	1278
13.99				读取256	2562	1534
15.151	10	4099:4100(1)			2563	1533
15.170	11		ACK 4100, win 1533			
15.172	12	4100:5124(1024)			3587	509
15.370	13		ACK 5124, win 509			
15.99				读取256	3331	765
17.99				读取256	3075	1021
19.99				读取256	2819	1277
20.151	14	5124:5633(509)			3328	768
20.170	15		ACK 5633, win 0			
21.99				读取256	3072	1024
23.99				读取256	2816	1280
25.151	16	5633:5634(1)			2817	1279
25.170	17		ACK 5634, win 1279			
25.171	18	5634:6145(511)			3328	768
25.174	19		ACK 6146, win 767			
25.99				读取256	3072	1024
27.99				读取256	2816	1280
29.99				读取256	2560	1536
31.99				读取256	2304	1792
33.99				读取256	2048	2048
35.99				读取256	1792	2304
37.99				读取256	1536	2560
39.99				读取256	1280	2816
39.99	20		ACK 6146, win 2816			
41.99				读取256	1024	3072
43.99				读取256	768	3328
45.99				读取256	512	3584
47.99				读取256	256	3840
49.99				读取256	0	4096
51.99				读取256(EOF)	0	4096
51.991	21		ACK 6146, win 4096			
51.992	22	ACK 2				

图22-3 接收方避免出现糊涂窗口综合症的事件序列

发送方的坚持定时器被复位，并在5秒后再次到时（在时刻10.151）。然后又发送一个字节并被确认（报文段8和9），而接收方的缓存空间还不够用（1022字节），使得通告窗口为0。

发送方的坚持定时器在时刻15.151再次时间到，又发送了另一个字节并被确认（报文段10和11）。这一次由于接收方有1533字节的有效缓存空间，因此通告了一个非0窗口。发送方立即利用这个窗口发送了1024字节的数据（报文段12）。对这1024字节数据的确认（报文段13）通告其窗口为509字节。这看起来与我们在前面看到的小窗口通告相抵触。

在这里之所以发生这种情况，是因为报文段11通告了一个大小为1533字节的窗口，而发送方只使用了其中的1024字节。如果在报文段13中的ACK通告其窗口为0，就会违反窗口的右边沿不能向左边沿移动而导致窗口收缩的TCP原则（见第20.3节）。这就是为什么必须通告

一个509字节的窗口的原因。

接下来我们看到发送方没有立即向这个小窗口发送数据。这就是发送方采取的糊涂窗口避免策略。相反,它等待另一个坚持定时器在时刻 20.151到时间,并在该时刻发送 509字节的数据。尽管它最终还是发送了一个长度为 509字节的小数据段,但在发送前它等待了 5秒钟,看是否会有一个ACK到达,以便可以将窗口开得更大。这 509字节的数据使得接收缓存仅剩下 768字节的有效空间,因此接收方通告窗口为 0 (报文段15)。

坚持定时器在时刻 25.151再次到时间,发送方发送 1个字节,于是接收缓存中有 1279字节的可用空间,这就是在报文段 17所通告的窗口大小。

发送方只有另外的 511个字节的数据需要发送,因此在收到 1279的窗口通告后立刻发送了这些数据 (报文段 18)。这个报文段也带有 FIN标志。接收方确认数据和 FIN,并通告窗口大小为 767 (见习题 22.2)。

由于发送应用进程在执行完 6个1024字节的写操作后发出关闭命令,发送方的连接从 ESTABLISHED状态转变到 FIN_WAIT_1状态,再到 FIN_WAIT_2状态 (见图 18-12)。它一直处于这个状态,直到收到对方的 FIN。在这个状态上没有设置定时器 (回忆我们在 18.6节结束时的讨论),因为它在报文段 18中发送的 FIN被报文段 19确认。这就是为什么我们看到发送方直到接收到 FIN (报文段 21) 为止没有发送其他任何数据的原因。

接收应用进程继续每隔 2秒从接收缓存区中读取 256个字节的数据。为什么在时刻 39.99发送 ACK (报文段 20) 呢?这是因为应用进程在时刻 39.99读取数据时,接收缓存中的可用空间已经从原来通告的 767 (报文段 19) 变为 2816,这相当于接收缓存中增加了额外的 2049字节的可用空间。回忆本节开始讲的第 1个规则,因为现在接收缓存已经增加了其空间的一半,因此接收方现在发送窗口更新。这意味着每次当应用进程从 TCP的接收缓存中读取数据时,接收的 TCP将检查是否需要更新发送窗口。

应用进程在时间 51.99发出最后一个读操作,然后收到一个文件结束标志,因为缓存已经变空。这就导致了最后两个完成连接终止的报文段 (报文段 21和22) 的发送。

22.4 小结

在连接的一方需要发送数据但对方已通告窗口大小为0时,就需要设置TCP的坚持定时器。发送方使用与第21章类似的重传间隔时间,不断地探查已关闭的窗口。这个探查过程将一直持续下去。

当运行一个例子来观察坚持定时器时,我们还观察到了 TCP的避免出现糊涂窗口综合症的现象。这就是使 TCP避免通告小的窗口大小或发送小的报文段。在我们的例子中,可以观察到发送方和接收方为避免糊涂窗口综合症所使用的策略。

习题

- 22.1 在图22-3中注意到所有确认 (报文段5、7、9、11、13、15和17) 的发送时刻为 :0.170, 5.170、10.170、15.170、20.170和25.170,还注意到在接收数据和发送 ACK之间的时间差分别为 :164.5、18.5、18.7、18.8、198.3、18.5和19.1 ms。试解释可能会发生的情况。
- 22.2 在图22-3中的时刻 25.174,发送出一个 767字节的通告窗口,而在接收缓存中有 768字节的可用空间。为什么相差 1个字节?