第9章　T/TCP的实现：TCP输出

9.1　概述

本章介绍为了支持T/TCP而对tcp_output函数所做的修改。在TCP中有许多程序段都要调用该函数来决定是否应该发出一个报文段，并且如果必要就发出一个。在　T/TCP中作了以下修改：

- 两个隐藏的状态标志可以打开TH_SYN和TH_FIN标志。
- T/TCP可以在SYN_SENT状态下发出多个报文段，但其前提是确知对等端也支持 T/TCP。
- 发送程序糊涂窗口避免机制必须考虑到新的 TF_NOPUSH标志，这个标志我们在3.6节中讨论过。
- 可以发出新的T/TCP选项(CC、CCnew和CCecho)。

9.2　tcp_output函数

9.2.1　新的自动变量

在tcp_output中说明了两个新的自动变量：

```
struct rmxp_tao *taop;
struct rmxp_tao  tao_noncached;
```

其中第一个变量是一个指针，指向相应对等端的 TAO缓存记录项。如果TAO缓存记录项不存在(这种情况不应该发生)，则taop指向tao_noncached，并且将这个结构初始化为0(这样它的tao_cc值就是未定义的)。

9.2.2　增加隐藏的状态标志

在tcp_output的开头，要从tcp_outflags向量中读取说明当前连接状态的TCP标志。图2-7给出了每个状态的相关标志。图 9-1中的代码用于在相应的隐藏状态标志处于开状态时，对TH_FIN标志和TH_SYN标志执行逻辑或。

tcp_output.c

```
71   again:
72     sendalot = 0;
73     off = tp->snd_nxt - tp->snd_una;
74     win = min(tp->snd_wnd, tp->snd_cwnd);

75     flags = tcp_outflags[tp->t_state];
76     /*
77      * Modify standard flags, adding SYN or FIN if requested by the
78      * hidden state flags.
79      */
```

图9-1　tcp_output　：增加隐藏状态标志

```
80      if (tp->t_flags & TF_SENDFIN)
81          flags |= TH_FIN;
82      if (tp->t_flags & TF_SENDSYN)
83          flags |= TH_SYN;
```

tcp_output.c

图9-1 (续)

这些代码位于卷2第681~682页。

9.2.3 在SYN_SENT状态不要重传SYN

图9-2中的程序读取对等端的TAO缓存内容，并且查看是否已经发出了SYN。这段代码位于卷2中图26-3的开头。

1. 读取TAO缓存记录项

117-119 读取对等端的TAO缓存内容，如果不存在，则改用自动变量tao_noncached，其初始值置为0。

> 如果使用了全0的记录项，它的值永远不变。这样，tao_noncached结构就可以静态分配并初始化为0，而不必用bzero将其设置为0。

2. 检查客户请求是否超过MSS

121-133 如果状态标志表明需要发送SYN，并且如果已经发出SYN，那么TH_SYN标志就要关闭。当一个应用程序用T/TCP向对等端发送多倍MSS数量的数据时可能发生这种情况 (见3.6节)。如果对等端支持T/TCP协议，这时可以分多个报文段发送，但只有第一个报文段应该设置SYN标志。如果我们不能确定对等端是否支持T/TCP(tao_ccsent值为0)，这时我们必须在三次握手过程完成以后才可以发送多个报文段。

tcp_output.c

```
116     len = min(so->so_snd.sb_cc, win) - off;

117     if ((taop = tcp_gettaocache(tp->t_inpcb)) == NULL) {
118         taop = &tao_noncached;
119         bzero(taop, sizeof(*taop));
120     }
121     /*
122      * Turn off SYN bit if it has already been sent.
123      * Also, if the segment contains data, and if in the SYN-SENT state,
124      * and if we don't know that foreign host supports TAO, suppress
125      * sending segment.
126      */
127     if ((flags & TH_SYN) && SEQ_GT(tp->snd_nxt, tp->snd_una)) {
128         flags &= ~TH_SYN;
129         off--, len++;
130         if (len > 0 && tp->t_state == TCPS_SYN_SENT &&
131             taop->tao_ccsent == 0)
132             return (0);
133     }
134     if (len < 0) {
```

tcp_output.c

图9-2 tcp_output ：在SYN_SENT状态不重传SYN

9.2.4 发送器的糊涂窗口避免机制

发送器的糊涂窗口避免机制有两处作了修改 (卷2第715页)，如图9-3所示

tcp_output.c

```
168     if (len) {
169         if (len == tp->t_maxseg)
170             goto send;
171         if ((idle || tp->t_flags & TF_NODELAY) &&
172             (tp->t_flags & TF_NOPUSH) == 0 &&
173             len + off >= so->so_snd.sb_cc)
174             goto send;
175         if (tp->t_force)
176             goto send;
177         if (len >= tp->max_sndwnd / 2 && tp->max_sndwnd > 0)
178             goto send;
179         if (SEQ_LT(tp->snd_nxt, tp->snd_max))
180             goto send;
181     }
```

tcp_output.c

图9-3 tcp_output ：糊涂窗口避免机制中，确定是否发送报文段

1. 发送最大报文段

169-170 如果允许，就发出最大报文段。

2. 允许应用程序关闭隐式推送

171-174 BSD实现中是这样处理的：如果不是正在等待对等端的ACK(idle值为真)，或者如果Nagle算法禁用(TF_NODELAY值为真)，并且TCP正在清空发送缓存，那么它总是发出一个报文段。有时称这种方式为隐式推送，因为除非受Nagle算法所限，否则应用程序每写一次都会导致一个报文段发送出去。T/TCP提供了一个新的插口选项，可以使BSD的隐式推送失效，这个选项就是TCP_NOPUSH，最后变成了TF_NOPUSH标志。我们在3.6节研究过有关这个标志的一个例子。在这段代码中，我们看到了只有以下三个条件同时为真，报文段才能发出：

1) 并不在等待ACK(idle值为真)，或者Nagle算法已经禁用(TF_NODELAY值为真)；

2) TCP_NOPUSH插口选项没有使用(默认值)；

3) TCP正在清空发送缓存(即所有未发的数据可以在一个报文段中发出)。

3. 检查接收窗口是否打开了至少一半

177-178 在常规的TCP中，整个这部分代码段不会因为收到第一个SYN而执行，因为这时len应该是0。但是在T/TCP中，很有可能在接收到另一端发来的SYN之前就发送数据。这就意味着需要根据max_sndwnd是否大于0来检测接收窗口是否已经打开了一半。这个变量是对等端通告的最大窗口，但是在从对等端收到通告前，它一直是0(即一直到收到对等端的SYN)。

4. 重传定时器到时发送

179-180 重传定时器到时后，snd_nxt小于snd_max。

9.2.5 有RST或SYN标志时强制发送报文段

卷2第688页的179~180行代码在SYN标志或RST标志打开时总是要发送一个报文段。这两行要用图9-4中的代码替代。

tcp_output.c

```
207     if ((flags & TH_RST) ||
208         ((flags & TH_SYN) && (tp->t_flags & TF_SENDSYN) == 0))
209         goto send;
```

tcp_output.c

图9-4 tcp_output ：检查RST和SYN标志，确定是否发送报文段

207-209 如果RST标志打开了，就总要发出一个报文段。如果 SYN标志打开了，则只有在相应的隐藏状态标志关闭时才会发出报文段。加上这项限制的理由可以看图 2-7。在最后5个服务器加星状态(半同步状态)下，TF_SENDSYN标志是打开的，这就会使图 9-1中的SYN标志被打开。在tcp_output中执行这项测试的目的是只在 SYN_SENT、SYN_RCVD、SYN_SENT*和SYN_RCVD*状态下才发出报文段。

9.2.6 发送MSS选项

这一小段代码(卷2第697页)有一个小小的变化。Net/3中的函数tcp_mss(有两个参量)改为tcp_msssend(仅仅以tp为参量)。这是因为我们需要把计算 MSS并发送与处理收到的MSS选项区分开来。Net/3中的tcp_mss函数同时完成这两项处理；在 T/TCP中，我们则用两个不同的函数来完成，它们是tcp_msssend和tcp_mssrcvd，我们将在下一章讨论这两个函数。

9.2.7 是否发送时间戳选项

卷2第698页，如果以下三个条件都成立，就发出时间戳选项： (1)TCP配置中要求使用时间戳选项；(2)正在构造的报文段不包括 RST标志；以及(3)要么这是一次主动打开或者 TCP已经从另一端接收到了一个时间戳(TF_RCVD_TSTMP)。对主动打开的测试只要查看 SYN标志是否打开以及ACK标志是否关闭即可。完成这三项测试的 T/TCP代码如图9-5所示。

tcp_output.c

```
283     /*
284      * Send a timestamp and echo-reply if this is a SYN and our side
285      * wants to use timestamps (TF_REQ_TSTMP is set) or both our side
286      * and our peer have sent timestamps in our SYN's.
287      */
288     if ((tp->t_flags & (TF_REQ_TSTMP | TF_NOOPT)) == TF_REQ_TSTMP &&
289         (flags & TH_RST) == 0 &&
290         ((flags & TH_ACK) == 0 ||
291          (tp->t_flags & TF_RCVD_TSTMP))) {
```

tcp_output.c

图9-5 tcp_output ：是否发送时间戳选项？

283-291 因为我们希望从客户端到服务器方向上发送的所有第一个报文段都携带时间戳选项(在多报文段请求的情况下，如图 3-9所示)，而不仅仅只是含有 SYN的第一个报文段，所以在T/TCP中第三项测试的前一半有所改变。对所有初始报文段的新测试项都是在没有 ACK标志的情况下进行的。

9.2.8 发送T/TCP的CC选项

是否发送三个新 CC选项之一的测试是看 TF_REQ_CC标志是否打开 (如果全局变量tcp_do_rfc1644非零，该标志由tcp_newtcpcb激活)、TF_NOOPT标志是否关闭以及RST标志是否关闭。发送哪个 CC选项则取决于输出报文段中 SYN标志和ACK标志的状态。这样就有四种可能的组合，前两种如图 9-6所示(这段代码在卷2第698页的第268~269行)。

TF_NOOPT标志是由新增的 TCP_NOOPT插口选项控制的。该插口选项出现在 Thomas Skibo写的 RFC 1323 的代码中(见12.7节)。在卷2中曾指出，这个标志(不是指插口选项)自从4.2BSD以后就已经在伯克利源代码中存在了，但通常无法将其打开。

如果设置了这个选项，TCP就不用随SYN发送任何选项。新增这个选项用来处理TCP实现中的不一致性，因为这些实现不能忽略未知的TCP选项(自从RFC 1323修改以后，增加了两个新的TCP选项)。

T/TCP所作的修改并没有改变确定是否发送MSS选项的那段代码(卷2第697页)。如果TF_NOOPT标志没有设置，这段代码就不发送MSS选项。但是Bob Braden在他的RFC 1323代码中指出，没有真正的理由需要阻止发送MSS选项。MSS选项是RFC 793规范中的一部分内容。

tcp_output.c

```
299     /*
300      * Send CC-family options if our side wants to use them (TF_REQ_CC),
301      * options are allowed (!TF_NOOPT) and it's not a RST.
302      */
303     if ((tp->t_flags & (TF_REQ_CC | TF_NOOPT)) == TF_REQ_CC &&
304         (flags & TH_RST) == 0) {
305         switch (flags & (TH_SYN | TH_ACK)) {
306             /*
307              * This is a normal ACK (no SYN);
308              * send CC if we received CC from our peer.
309              */
310         case TH_ACK:
311             if (!(tp->t_flags & TF_RCVD_CC))
312                 break;
313             /* FALLTHROUGH */
314             /*
315              * We can only get here in T/TCP's SYN_SENT* state, when
316              * we're sending a non-SYN segment without waiting for
317              * the ACK of our SYN.  A check earlier in this function
318              * assures that we only do this if our peer understands T/TCP.
319              */
320         case 0:
321             opt[optlen++] = TCPOPT_NOP;
322             opt[optlen++] = TCPOPT_NOP;
323             opt[optlen++] = TCPOPT_CC;
324             opt[optlen++] = TCPOLEN_CC;
325             *(u_int32_t *) & opt[optlen] = htonl(tp->cc_send);
326             optlen += 4;
327             break;
```

tcp_output.c

图9-6 tcp_output：发送一个CC选项，第一部分

1. SYN关闭，ACK打开

310-313 如果SYN标志关闭，但ACK标志打开，这就是常规的ACK(即连接已经建立)。只有从对等端收到一个CC选项以后，才会发送CC选项。

2. SYN关闭，ACK关闭

314-320 只有在SYN_SENT*状态下，即在连接建立以前就发送了一个非SYN报文段时，这两个标志才会同时关闭。也就是说，在客户一次发送了多倍MSS数量的数据时才会这样。图9-2中的代码能够确保只有在对等端也支持T/TCP时才会进入这种状态。这种情况下就要发送CC选项。

3. 构造CC选项

321-327 在构造CC选项时，要先加上两个空字符。该连接cc_send的值就作为CC选项的

内容发送出去。

SYN标志和ACK标志状态组合的剩余两种情况如图9-7所示。

tcp_output.c

```
328             /*
329              * This is our initial SYN (i.e., client active open).
330              * Check whether to send CC or CCnew.
331              */
332         case TH_SYN:
333             opt[optlen++] = TCPOPT_NOP;
334             opt[optlen++] = TCPOPT_NOP;
335             opt[optlen++] =
336                 (tp->t_flags & TF_SENDCCNEW) ? TCPOPT_CCNEW : TCPOPT_CC;
337             opt[optlen++] = TCPOLEN_CC;
338             *(u_int32_t *) & opt[optlen] = htonl(tp->cc_send);
339             optlen += 4;
340             break;
341             /*
342              * This is a SYN, ACK (server response to client active open).
343              * Send CC and CCecho if we received CC or CCnew from peer.
344              */
345         case (TH_SYN | TH_ACK):
346             if (tp->t_flags & TF_RCVD_CC) {
347                 opt[optlen++] = TCPOPT_NOP;
348                 opt[optlen++] = TCPOPT_NOP;
349                 opt[optlen++] = TCPOPT_CC;
350                 opt[optlen++] = TCPOLEN_CC;
351                 *(u_int32_t *) & opt[optlen] = htonl(tp->cc_send);
352                 optlen += 4;
353                 opt[optlen++] = TCPOPT_NOP;
354                 opt[optlen++] = TCPOPT_NOP;
355                 opt[optlen++] = TCPOPT_CCECHO;
356                 opt[optlen++] = TCPOLEN_CC;
357                 *(u_int32_t *) & opt[optlen] = htonl(tp->cc_recv);
358                 optlen += 4;
359             }
360             break;
361         }
362     }
363     hdrlen += optlen;
```

tcp_output.c

图9-7 tcp_output ：发送CC选项之一，第二部分

4. SYN打开，ACK关闭(客户主动打开)

328-340 当客户执行主动打开时，SYN打开且ACK关闭。如果应该发送CCnew选项而不是CC选项，图12-3中的代码完成TF_SENDCCNEW标志的设置，同时也设置cc_send值。

5. SYN打开，ACK打开(服务器响应客户端的SYN)

341-360 当SYN标志和ACK标志同时处于打开状态时，这就是服务器对对等端的主动打开作出了响应。如果对等端发送了CC或CCnew选项之一(设置了TF_RCVD_CC)，这时我们要向对等端发送CC选项(cc_send)和对对等端CC值(cc_recv)的CCecho。

6. 根据TCP选项调整TCP首部长度

363 所有的TCP选项都加长了TCP首部的长度。

9.2.9 根据TCP选项调整数据长度

t_maxopd是tcpcb结构的新字段且是最大数据长度，并且也是常规 TCP报文段的选项。因为窗口宽度选项和CCecho选项只在SYN报文段中出现，因此在SYN报文段中的选项(见图2-2和图2-3)很有可能会比非SYN报文段的选项(见图2-4)需要更多的字节空间。图9-8中的代码根据TCP选项的大小调整发送报文段中的数据量。这段代码用于替代卷 2第698页的第270~277

tcp_output.c

```
364     /*
365      * Adjust data length if insertion of options will
366      * bump the packet length beyond the t_maxopd length.
367      * Clear the FIN bit because we cut off the tail of
368      * the segment.
369      */
370     if (len + optlen > tp->t_maxopd) {
371         /*
372          * If there is still more to send, don't close the connection.
373          */
374         flags &= ~TH_FIN;
375         len = tp->t_maxopd - optlen;
376         sendalot = 1;
377     }
```

tcp_output.c

行。

图9-8 tcp_output ：根据TCP选项的大小调整发送数据量

364-377 如果数据长度(len)加上选项长度超过了t_maxopd，发送的数据量就要缩减，FIN标志关闭(如果它原来是开状态)，且sendalot打开(在当前报文段发出后强迫再次执行tcp_output循环)。

这些代码并不是T/TCP专有的。它应该对任何一个既携带数据又有 TCP选项(例如：RFC 1323时间戳选项)的报文段执行。

9.3 小结

T/TCP在原本500行的tcp_output函数上增加了大约100行代码。增加的大部分代码都与发送新增的T/TCP选项CC、CCnew和CCecho等有关。

另外，如果对等端支持T/TCP，T/TCP的tcp_output函数可以在SYN_SENT状态下发送多个报文段。