# tcp短连接TIME_WAIT问题解决方法大全（3）——tcp_tw_recycle

2012年11月04日 21:52:09 阅读数：13712

【tcp_tw_recycle和tcp_timestamps】
参考官方文档（http://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt），tcp_tw_recycle解释如下：
tcp_tw_recycle选项作用为：Enable fast recycling TIME-WAIT sockets. Default value is 0.
tcp_timestamps选项作用为：Enable timestamps as defined in RFC1323. Default value is 1.

这两个选项是linux内核提供的控制选项，和具体的应用程序没有关系，而且网上也能够查询到大量的相关资料，但信息都不够完整，最主要的几个问题如下；
1）快速回收到底有多快？
2）有的资料说只要打开tcp_tw_recycle即可，有的又说要tcp_timestamps同时打开，具体是哪个正确？
3）为什么从虚拟机NAT出去发起客户端连接时选项无效，非虚拟机连接就有效？

为了回答上面的疑问，只能看代码，看出一些相关的代码供大家参考：

```
=====linux-2.6.37 net/ipv4/tcp_minisocks.c 269======
void tcp_time_wait(struct sock *sk, int state, int timeo)
{
struct inet_timewait_sock *tw = NULL;
const struct inet_connection_sock *icsk = inet_csk(sk);
const struct tcp_sock *tp = tcp_sk(sk);
int recycle_ok = 0;


    //判断是否快速回收，这里可以看出tcp_tw_recycle和tcp_timestamps两个选项都打开的时候才进行快速回收，
    //且还有进一步的判断条件，后面会分析，这个进一步的判断条件和第三个问题有关
if (tcp_death_row.sysctl_tw_recycle && tp->rx_opt.ts_recent_stamp)
recycle_ok = icsk->icsk_af_ops->remember_stamp(sk);


if (tcp_death_row.tw_count < tcp_death_row.sysctl_max_tw_buckets)
tw = inet_twsk_alloc(sk, state);


if (tw != NULL) {
struct tcp_timewait_sock *tcptw = tcp_twsk((struct sock *)tw);
```

```
        //计算快速回收的时间，等于 RTO * 3.5，回答第一个问题的关键是RTO（Retrans
mission Timeout）大概是多少
const int rto = (icsk->icsk_rto << 2) - (icsk->icsk_rto >> 1);

        //。。。。。。此处省略很多代码。。。。。。

    if (recycle_ok) {
            //设置快速回收的时间
tw->tw_timeout = rto;
} else {
tw->tw_timeout = TCP_TIMEWAIT_LEN;
if (state == TCP_TIME_WAIT)
timeo = TCP_TIMEWAIT_LEN;
}

        //。。。。。。此处省略很多代码。。。。。。
}
```

RFC中有关于RTO计算的详细规定，一共有三个：RFC-793、RFC-2988、RFC-6298，Linux的实现是参考RFC-2988。
对于这些算法的规定和Linuxde 实现，有兴趣的同学可以自己深入研究，实际应用中我们只要记住Linux如下两个边界值：

```
=====linux-2.6.37 net/ipv4/tcp.c 126=================
#define TCP_RTO_MAX ((unsigned)(120*HZ))
#define TCP_RTO_MIN ((unsigned)(HZ/5))
=======================================================
```

这里的HZ是1s，因此可以得出RTO最大是120s，最小是200ms，对于局域网的机器来说，正常情况下RTO基本上就是200ms，因此3.5 RTO就是700ms
也就是说，快速回收是TIME_WAIT的状态持续700ms，而不是正常的2MSL（Linux是1分钟，请参考：include/net/tcp.h 109行TCP_TIMEWAIT_LEN定义）。
实测结果也验证了这个推论，不停的查看TIME_WAIT状态的连接，偶尔能看到1个。

最后一个问题是为什么从虚拟机发起的连接即使设置了tcp_tw_recycle和tcp_timestamps，也不会快速回收，继续看代码：
tcp_time_wait函数中的代码行：recycle_ok = icsk->icsk_af_ops->remember_stamp(sk);对应的实现如下：

```
=====linux-2.6.37 net/ipv4/tcp_ipv4.c 1772=====
int tcp_v4_remember_stamp(struct sock *sk)
{
    //。。。。。。此处省略很多代码。。。。。。
```

```
    //当获取对端信息时，进行快速回收，否则不进行快速回收
if (peer) {
if ((s32)(peer->tcp_ts - tp->rx_opt.ts_recent) <= 0 ||
   ((u32)get_seconds() - peer->tcp_ts_stamp > TCP_PAWS_MSL &&
    peer->tcp_ts_stamp <= (u32)tp->rx_opt.ts_recent_stamp)) {
peer->tcp_ts_stamp = (u32)tp->rx_opt.ts_recent_stamp;
peer->tcp_ts = tp->rx_opt.ts_recent;
}
if (release_it)
inet_putpeer(peer);
return 1;
}


return 0;
}
```

上面这段代码应该就是测试的时候虚拟机环境不会释放的原因，当使用虚拟机NAT出去的时候，服务器无法获取隐藏在NAT后的机器信息。
生产环境也出现了设置了选项，但TIME_WAIT连接数达到4W多的现象，可能和虚拟机有关，也可能和组网有关。

总结一下：
1）快速回收到底有多快？
局域网环境下，700ms就回收；
2）有的资料说只要打开tcp_tw_recycle即可，有的又说要tcp_timestamps同时打开，具体是哪个正确？
需要同时打开，但默认情况下tcp_timestamps就是打开的，所以会有人说只要打开tcp_tw_recycle即可；
3）为什么从虚拟机发起客户端连接时选项无效，非虚拟机连接就有效？
和网络组网有关系，无法获取对端信息时就不进行快速回收；

综合上面的分析和总结，可以看出这种方法不是很保险，在实际应用中可能受到虚拟机、网络组网、防火墙之类的影响从而导致不能进行快速回收。

附：
1）tcp_timestamps的说明详见RF1323，和TCP的拥塞控制（Congestion control）有关。

2）打开此选项，可能导致无法连接，请参考：http://www.pagefault.info/?p=416