本文对应的运行时参数位于：/proc/sys/net/ipv4/\*。“掌控”这些参数，能使您在Linux网络服务问题解决、调优方面功力大增。此乃“系统高手”之必备知识。

临时改动某个系统参数的值，可用两种方法来实现，例如，想启用IP路由转发功能：

echo 1 > /proc/sys/net/ipv4/ip\_forward

或

sysctl -w net.ipv4.ip\_forward=1

以上两种方法都可以即时开启路由转发功能。但如果系统重启，或执行了“service network restart”命令，所设置的值即会丢失，如果想永久保留设置，可以修改：

/etc/sysctl.conf 将 net.ipv4.ip\_forward=0 改为 net.ipv4.ip\_forward=1 然后执行 sysctl –p。

系统命令 sysctl 常用参数的意义：

-w 临时改动某个指定参数的值。   
-a 显示所有的系统参数。   
-p 从指定的文件加载系统参数，默认的从 /etc/sysctl.conf 中加载。

**ICMP相关服务参数配置**

通常我们使用 icmp包 来探测目标主机上的其它协议（如TCP和UDP）是否可用。比如：包含“destination unreachable”信息的 icmp包 就是最常见的icmp应用。

icmp\_destunreach\_rate

INTEGER。设置内容为“Destination Unreachable”的icmp包响应速率。设置值应为整数，以jiffies计。

举例。假设有A、B两部主机，首先我们在主机A上执行以下 ipchains 语句：“ipchains －A input -p icmp -j REJECT”，这里的REJECT和DENY不同，DENY会丢掉符合条件的包如同没有接收到该包一样，而REJECT会在丢掉该包的同时给请求主机发回一个"Destination Unreachable"的icmp。然后在主机B上ping主机A，这时候我们会发现“Destination Unreachable”icmp包的响应速度是很及时的。接着我们在主机A上执行：“echo "1000" > /proc/sys/net/ipv4/icmp\_destunreach\_rate”，也即每10秒钟响应一个“Destination Unreachable”的icmp包。这时候再从主机B上ping主机A就会发现“Destination Unreachable”icmp包的响应速度已经明显变慢，刚好是每10秒响应一次。

**icmp\_echo\_ignore\_all**

**BOOLEAN** 。设置系统是否忽略所有的 ICMP Echo 请求，如果设置了一个非0值，系统将忽略所有的 icmp echo 请求。其实这是 icmp\_echoreply\_rate 的一种极端情况。1：表示忽略，0：表示响应。

Ping命令通过向计算机发送 ICMP Echo 请求报文并且监听回应报文的返回，以校验与远程计算机或本地计算机的连接。

设置为1可能防止你被利用成为DoS攻击的跳板。

**icmp\_echo\_ignore\_broadcasts**

**BOOLEAN** 。设置是否响应 icmp echo 请求广播，设置值应为布尔值，0表示响应 icmp echo 请求广播，1表示忽略。

这可以成为非常好用的防止拒绝服务攻击的工具。设置为1来忽略这些子网广播消息。

注意：windows系统是不响应 icmp echo 请求广播的。

举例：在 RedHat6.x 和 RedHat7 上该值缺省为0，这样当有个用户ping这2台服务器所在的网段（子网）的网络地址时，所有的Linux服务器就会响应，从而也能让让该用户得到服务器的IP地址。现在可以执行“echo "1" > /proc/sys/net/ipv4/icmp\_echo\_ignore\_broadcasts”来关闭该功能，从而防止icmp风暴，防止网络阻塞。

icmp\_echoreply\_rate

INTEGER。设置系统响应 icmp echo 请求的icmp包的响应速度。

举例：假设有A、B两部主机，首先我们在主机B上ping主机A，可以看到响应很正常，然后在主机A上执行“echo "1000" > /proc/sys/net/ipv4/icmp\_echoreply\_rate”也即每10秒钟响应一个 icmp echo 请求包。然后再ping主机A就可以看到响应速度已经变成10秒一次。

可以合理的调整该参数的值来防止icmp风暴。

icmp\_ignore\_bogus\_error\_responses

BOOLEAN。设置它之后，可以忽略由网络中的那些声称回应地址是广播地址的主机生成的ICMP错误。

icmp\_paramprob\_rate

INTEGER。当系统接收到数据报的损坏的IP或TCP头时，就会向源发出一个包含有该错误信息的 icmp包。这个参数就是用来设置向源发送这种 icmp包 的速度。当然，在通常情况下IP或TCP头出错是很少见的。

icmp\_timeexceed\_rate

INTEGER。数据报在网络上传输时，其生存时间（time to live）字段会不断减少，当生存时间为0时，正在处理该数据报的路由器就会丢弃该数据报，同时给源主机发送一个“time to live exceeded”的icmp包。该参数就是用来设置这种 icmp包 的发送速度。当然，这通常用于充当路由器的Linux主机。

在traceroute时导致著名的“Solaris middle star”。

**IP相关服务参数配置**

ip\_autoconfig

BOOLEAN。表示主机是（1）否（0）通过RARP、BOOTP、DHCP或者其它机制取得其IP配置。

ip\_default\_ttl

INTEGER。默认值为64。数据包的生存期。设置为64是安全的。如果你的网络规模巨大就提高这个值，不要因为好玩而这么做——那样会产生有害的路由环路。实际上，在很多情况下你要考虑能否减小这个值。

TTL是IP协议包中的一个值，它告诉网络，数据包在网络中的时间是否太长而应被丢弃。有很多原因使包在一定时间内不能被传递到目的地。解决方法就是在一段时间后丢弃这个包，然后给发送者一个报文，由发送者决定是否要重发。TTL的初值通常是系统缺省值，是包头中的8位的域。TTL的最初设想是确定一个时间范围，超过此时间就把包丢弃。由于每个路由器都至少要把TTL域减一，TTL通常表示包在被丢弃前最多能经过的路由器个数。当记数到0时，路由器决定丢弃该包，并发送一个ICMP报文给最初的发送者。

ip\_dynaddr

0 1 2 枚举。该参数通常用于使用拨号连接的情况，可以使系统能够立即改变IP包的源地址为该IP地址，同时中断原有的TCP对话而用新地址重新发出一个syn请求包，开始新的TCP对话。在使用IP欺骗时，该参数可以立即改变地址伪装为新的IP地址。该参数的参数值可以是： 1 - 启用该功能； 2 - 使用冗余模式启用该功能；0 - 禁止该功能。

应用实例：在使用ipchains配置IP欺骗带动局域网共享一个ppp连接上网时，有时会出现刚开时连接一个站点连不通，再次刷新又可以连接的情况，这时候就可以设置该参数的值为1，从而立即改变伪装地址为新的ip地址，就可以解决这类问题。命令为：echo "1" > /proc/sys/net/ipv4/ip\_dynaddr。

ip\_forward

BOOLEAN。默认值是0。内核是否转发数据包。缺省禁止（0）。启用（1）该参数可以使系统充当路由器。注意，我们可以在单网卡或双网卡的主机上实现IP转发。

举例：假设我们使用一部装有双网卡的Linux主机充当防火墙，这时候我们就必须执行以下命令来打开IP转发功能：“echo "1" > /proc/sys/net/ipv4/ip\_forward”。

**ip\_local\_port\_range**

**INTEGER** 。用于向外连接的端口范围。缺省情况下：1024到4999（其实很小）。

ip\_no\_pmtu\_disc

BOOLEAN。默认值是0。如果你想禁止“沿途MTU发现”就设置它。

“沿途MTU发现”是一种技术，可以在传输路径上检测出最大可能的MTU（最大传输单元，Maximum Transmission Unit）值。

**ipfrag\_high\_thresh**

**INTEGER** 。用于IP分片汇聚的最大内存用量。分配了这么多字节的内存后，一旦用尽，分片处理程序就会丢弃分片。

**ipfrag\_low\_thresh**

**INTEGER** 。用于IP分片汇聚的最小内存用量。

**ipfrag\_time**

**INTEGER** 。IP分片在内存中的保留时间（秒数）。

ip\_nonlocal\_bind

BOOLEAN。默认值是0。如果您希望应用程序能够绑定到不属于本地网卡的地址上时，设置这个选项。这在你的机器没有专线连接（甚至是动态连接）时非常有用，因为即使你的连接断开，你的服务也可以启动并绑定在一个指定的地址上。

**TCP相关服务参数配置**

在 TCP/IP 协议中，TCP协议提供可靠的连接服务，采用三次握手建立一个连接。   
第一次握手：建立连接时，客户端发送syn包（syn=j）到服务器，并进入 SYN\_SEND 状态，等待服务器确认；   
第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即 SYN+ACK 包，此时服务器进入 SYN\_RECV 状态；

第三次握手：客户端收到服务器的 SYN＋ACK 包，向服务器发送确认包 ACK（ack=k+1），此包发送完毕，客户端和服务器都进入 ESTABLISHED 状态，完成三次握手，客户端与服务器开始传送数据。

tcp\_abort\_on\_overflow

BOOLEAN。缺省值是0。当守护进程太忙而不能接受新的连接，就象对方发送reset消息（RST包）。这意味着当溢出的原因是因为一个偶然的猝发，那么连接将恢复状态。只有在你确信守护进程真的不能完成连接请求时才打开该选项，该选项会影响客户的使用。

对待已经满载的 sendmail，apache 这类服务，开启之可以很快让客户端终止连接，可以给予服务程序处理已有连接的缓冲机会，所以很多防火墙推荐打开它。

RST（Reset the connection）：用于复位因某种原因引起出现的TCP错误连接，也用来拒绝非法数据和请求。如果接收到RST位时候，通常发生了某些错误。

**tcp\_fin\_timeout**

**INTEGER** 。默认值是 60。对于本端断开的 socket 连接，这个参数决定了它保持在 FIN-WAIT-2 状态的时间，对端可以出错并永远不关闭连接，甚至意外宕机。缺省值是60秒，Linux 2.2 内核的通常值是180秒，你可以按这个设置，但要记住的是，即使你的机器是一个轻载的WEB服务器，也有因大量死套接字而内存溢出的风险。FIN- WAIT-2 的危险性比 FIN-WAIT-1 要小，因为它最多只能吃掉1.5K内存，但是它们的生存期长些。参见 tcp\_max\_orphans。

事实上做NAT的时候，降低该值也是好处显著的，可以考虑降低该值为30。

说明：TCP的终止是通过双方的四次握手实现。发起终止的一方执行主动关闭，响应的另一方执行被动关闭。   
发起方更改状态为 FIN\_WAIT\_1，关闭应用程序进程，发出一个TCP的FIN段；   
接收方收到FIN段，返回一个带确认序号的ACK，同时向自己对应的进程发送一个文件结束符EOF，同时更改状态为CLOSE\_WAIT，发起方接到ACK后状态更改为FIN\_WAIT\_2；   
接收方关闭应用程序进程，更改状态为LAST\_ACK，并向对方发出一个TCP的FIN段；   
发起方接到FIN后状态更改为TIME\_WAIT，并发出这个FIN的ACK确认。ACK发送成功后（2MSL内）双方TCP状态变为CLOSED。

**tcp\_keepalive\_time**

**INTEGER** 。默认值是7200秒（2小时）。 当keepalive起用时，TCP发送keepalive消息的频度。

由于目前网络攻击等因素，造成了利用这个进行的攻击很频繁。如果两边建立了连接，然后不发送任何数据或者 rst/fin 消息，那么持续的时间是不是就是2小时，那么就会加大空连接攻击的风险。tcp\_keepalive\_time 就是预防此情形的。一般在做nat服务的时候修改此值为1800秒。

**tcp\_keepalive\_intvl**

**INTEGER** 。默认值为75秒。当 keepalive 探测没有确认时，探测消息发送的频率。默认值75秒乘以 tcp\_keepalive\_probes 就得到对于从开始探测以来没有响应的连接杀除的时间。也就是没有活动的连接将在大约11分钟后被丢弃。

对于普通应用来说，这个值有一些偏大，可以根据需要改小。特别是web类服务器需要改小该值，15秒也许是个比较合适的值。

**tcp\_keepalive\_probes**

**INTEGER** 。默认值是9。在认定连接失效/断开之前，发送多少个TCP的 keepalive 探测包。默认值9乘以 tcp\_keepalive\_intvl 之后决定了：一个连接发送了 keepalive 之后可以有多少时间没有回应（同上）。

注意：保持连接仅在 SO\_KEEPALIVE 套接字选项被打开是才发送。参数默认不需要修改，当然根据情况也可以适当地缩短此值，设置为5也许比较合适。

**tcp\_max\_orphans**

**INTEGER** 。缺省值是8192。系统所能处理不属于任何进程的 TCP sockets 的最大数量。假如超过这个数量，那么不属于任何进程的连接将立即被复位并打印出警告信息。之所以要设定这个限制，纯粹为了抵御那些简单的 DoS 攻击，你绝对不能过分依靠这个参数或者人为地减小这个参数值，更应该增加这个值（如果增加了内存之后）。

注意：每个“孤儿套接字”最多能够吃掉你64K不可交换的内存。

这个参数值在 Redhat AS 版本中被设置为32768，但是很多防火墙修改的时候，建议该值为2000。

**tcp\_orphan\_retries**

**INTEGER** 。默认值是7。在本端丢弃TCP连接之前，要进行多少次重试。默认值是7次，相当于 50秒 - 16分钟，视 RTO 而定。如果您的系统是负载很大的Web服务器，那么也许需要降低该值，因为这类 sockets 可能会耗费大量的资源。另外请参考 tcp\_max\_orphans。

做NAT的时候，降低该值也是好处显著的，可以尝试降低该值为3或更小。

**tcp\_max\_syn\_backlog**

**INTEGER** 。对于那些尚未收到客户端确认的连接请求，需要保存在队列中的最大数目。对于超过 128MB 内存的系统，默认值是 1024；低于 128MB 内存的系统则为 128。如果服务器经常出现过载，可以尝试增加这个数字。

警告：假如您将此值设为大于 1024﹐最好修改 include/net/tcp.h 里面的 TCP\_SYNQ\_HSIZE，以保持 TCP\_SYNQ\_HSIZE\*16<=tcp\_max\_syn\_backlog，并且重新编译内核。

SYN Flood攻击利用TCP协议散布握手的缺陷，伪造虚假源IP地址发送大量 TCP-SYN 半打开连接到目标系统，最终导致目标系统 Sockets 队列资源耗尽而无法接受新的连接。为了应付这种攻击，现代Unix系统中普遍采用多连接队列处理的方式来缓冲（而不是解决）这种攻击：用一个基本队列处理正常的完全连接应用（Connect() 和 Accept()），而用另一个队列单独存放半打开连接。这种双队列处理方式和其它一些系统内核措施（如Syn-Cookies/Caches）联合应用时，能够比较有效的缓解小规模的 SYN Flood 攻击（事实证明<1000p/s）加大SYN队列长度可以容纳更多等待连接的网络连接数，所以对Server来说可以考虑增大该值。

**tcp\_max\_tw\_buckets**

**INTEGER** 。默认值是180000。系统同时能处理的最大 time-wait sockets 数目。如果超过此数的话，time-wait socket 会被立即清除并且显示警告信息。之所以要设定这个限制，纯粹为了抵御那些简单的 DoS 攻击，千万不要人为的降低这个限制。不过，如果网络条件需要比默认值更多，则可以在增加内存的情况下提高它。

做NAT的时候最好可以适当地增加该值。

**tcp\_retrans\_collapse**

**BOOLEAN** 。缺省值为1 。为兼容某些糟糕的打印机设置的“将错就错”选项。即再次发送时，把数据包增大一些，来避免某些TCP协议栈的BUG。

一般网络服务器不需要这个支持，可以关闭它。

tcp\_retries1

INTEGER。默认值是3。放弃回应一个TCP连接请求（即认定出错并向网络层提交错误报告）前，需要进行多少次重试。RFC 规定最低的数值是3，这也是默认值，根据RTO的值不同大约在 3秒～8分钟 之间。

注意：这个值同时还决定进入的syn连接。

**tcp\_retries2**

**INTEGER** 。在丢弃激活（已建立通讯状况）的TCP连接之前，需要进行多少次重试。默认值为15，根据RTO的值不同，相当于13-30分钟（RFC1122规定必须大于100秒）。

这个值根据目前的网络设置，可以适当地改小，例如Web服务网络内修改为5。

tcp\_rfc1337

BOOLEAN。缺省值为0。这个开关可以启动对于在 RFC1337 中描述的“tcp 的time-wait暗杀危机”问题的修复。启用后，内核将丢弃那些发往 time-wait 状态TCP套接字的RST 包。

tcp\_sack

BOOLEAN。默认值为1。是否启用有选择的应答（Selective Acknowledgment），它可以用来查找特定的遗失数据包——因此有助于快速恢复状态。该设置表示，这可以通过有选择的应答乱序接收到的报文来提高性能（这样可以让发送者只发送丢失的报文段）。

对于广域网通信来说这个选项应该启用，但是这会增加对 CPU 的占用。

tcp\_fack

BOOLEAN。默认值为1。开启FACK拥塞避免和快速重传功能。

注意：当 tcp\_sack 设置为0的时候，这个值即使设置为1也无效。

tcp\_dsack

BOOLEAN。默认值为1。允许TCP发送“两个完全相同”的SACK。

tcp\_stdurg

BOOLEAN。默认值为0。启用TCP紧急指针的主机需求解释。因为绝大多数主机采用BSD解释，所以如果你在Linux上打开它，可能会影响它与其它机器的正常通讯。

**tcp\_syn\_retries**

**INTEGER** 。对于一个新建连接，内核要发送多少个 SYN 连接请求才决定放弃。不应该大于255，默认值是5，对应于180秒左右时间。

对于大负载而物理通信良好的网络而言，这个值偏高，可修改为2。这个值仅仅是针对对外的连接，对进来的连接，则是由 tcp\_retries1 决定的。

**tcp\_synack\_retries**

**INTEGER** 。默认值是5。对于远端的连接请求SYN，内核会发送 SYN ＋ ACK 数据报，以确认收到上一个SYN连接请求包。这是所谓的三次握手（threeway handshake）机制的第二个步骤。这里决定内核在放弃连接之前所送出的 SYN+ACK 数目。不应该大于255，默认值是5，对应于180秒左右时间。

可以根据上面的 tcp\_syn\_retries 来决定这个值。

tcp\_timestamps

BOOLEAN。缺省值为1。时间戳可以避免序列号的卷绕。可以防范那些伪造的 sequence 号码。一条1G的宽带线路或许会重遇到带 out-of-line 数值的旧 sequence 号码（假如它是由于上次产生的）。时间戳会让它知道这是个“旧封包”。

该配置表示是否启用以一种比超时重发更精确的方法（RFC 1323）来启用对 RTT 的计算。为了实现更好的性能应该启用这个选项。

**tcp\_tw\_recycle**

**BOOLEAN** 。默认值是0。打开快速 TIME-WAIT sockets 回收。除非得到技术专家的建议或要求，请不要随意修改这个值。

做NAT的时候，建议打开它。

**tcp\_tw\_reuse**

**BOOLEAN** 。默认值是0。该配置表示是否允许重用处于 TIME-WAIT 状态的socket用于新的TCP连接。

这个对快速重启动某些服务，而启动后提示端口已经被使用的情形非常有帮助，例如Web应用服务。

tcp\_window\_scaling

BOOLEAN。缺省值为1。该配置表示设置 tcp/ip 会话的滑动窗口大小是否可变。参数值为布尔值，为1时表示可变，为0时表示不可变。tcp/ip 通常使用的窗口最大可达到 65535 字节，对于高速网络，该值可能太小，这时候如果启用了该功能，可以使 tcp/ip 滑动窗口大小增大几个数量级，从而提高数据传输的能力（RFC 1323）。

对普通的百兆网络而言，关闭会降低开销，所以如果不是高速网络，可以考虑设置为0。

一旦内核认为它无法发包，就会丢弃这个包，并向发包的主机发送ICMP通知。

**tcp\_syncookies**

**BOOLEAN** 。默认值是0。只有在内核编译时选择了 CONFIG\_SYNCOOKIES 时才会发生作用。当syn等候队列出现溢出时象对方发送syncookies。目的是为了防止 synflood 攻击。

注意：该选项千万不能用于那些没有受到攻击的高负载服务器，如果在日志中出现 synflood 消息，但是调查发现没有收到 synflood 攻击，而是合法用户的连接负载过高的原因，你应该调整其它参数来提高服务器性能。参考: tcp\_max\_syn\_backlog / tcp\_synack\_retries / tcp\_abort\_on\_overflow。

syncookie严重的违背TCP协议，不允许使用TCP扩展，可能对某些服务导致严重的性能影响（如SMTP转发）。该实现与BSD上面使用的 tcp proxy 一样,是违反了RFC中关于tcp连接的三次握手实现的，但是对于防御 synflood 攻击的确很有用。

tcp\_ecn

BOOLEAN。默认值为0。打开TCP的直接拥塞通告功能。

tcp\_reordering

INTEGER。默认值是3。TCP流中重排序的数据报最大数量 。

有推荐把这个数值略微调整大一些，比如5。

**tcp\_wmem**

**3个INTEGER列举：min default max** 。

min：为 TCP socket 预留用于发送缓冲的内存最小值。默认值为4K（4096 字节）。   
default：为 TCP socket 预留用于发送缓冲的默认内存数量，该值会影响其它协议使用的 net.core.wmem\_default 值，一般要低于 net.core.wmem\_default 的值。默认值为16K（16384 字节）。   
max：用于 TCP socket 发送缓冲的内存最大值。该值不会影响 net.core.wmem\_max 值，”静态“选择参数 SO\_SNDBUF 不受该值影响。默认值为128K（131072 字节）。对于服务器而言，增加这个参数的值对于发送数据很有帮助，例如在某Web服务网络环境（内存8G）中，修改为：“4096    16384    4194304”。

**tcp\_rmem**

**3个INTEGER列举：min default max** 。

min：为 TCP socket 预留用于接收缓冲的内存数量，即使在内存出现紧张情况下 tcp socket 都至少会有这么多数量的内存用于接收缓冲，默认值为8K。   
default：为 TCP socket 预留用于接收缓冲的默认内存数量，该值会影响其它协议使用的 net.core.wmem\_default 值，该值决定了在 tcp\_adv\_win\_scale 和 tcp\_app\_win 等于默认值0的情况下，TCP窗口大小为 65535 。默认值为 87380。   
max：用于 TCP socket 接收缓冲的内存最大值。该值不会影响 net.core.wmem\_max 值，”静态“选择参数 SO\_SNDBUF 不受该值影响。默认值为128K。max的设置一般是default的两倍。对于NAT应用来说应该增加它。例如在某Web服务网络环境（内存8G）中，修改为“4096    87380    4194304”。

**tcp\_mem**

**3个INTEGER列举：low pressure high** 。

low：当TCP使用了低于该值的“内存页面数”时，TCP不会考虑释放内存。理想情况下，这个值应与指定给 tcp\_wmem 的第3个值相匹配——根据这第3个值，最大页面大小乘以最大并发请求数除以页大小：131072 \* 300 / 4096。   
pressure：当TCP使用了超过该值的“内存页面数”时，TCP试图稳定其内存使用，进入 pressure 模式，当内存消耗低于low值时则退出 pressure 状态。理想情况下这个值应该是 TCP 可以使用的总缓冲区大小的最大值：204800 \* 300 / 4096。   
high：允许所有 tcp sockets 用于排队缓冲数据报的“内存页面数”。如果超过这个值，TCP 连接将被拒绝，这就是为什么不要令其过于保守的原因了。在这种情况下，增加这个值的价值很大，它能处理很多连接，是所预期的 2.5 倍，或者使现有连接能够传输 2.5 倍的数据。一般情况下这些值是在系统启动时根据系统内存数量计算得到的。例如在某Web服务网络环境（内存8G）中，修改为“196608    262144    393216”。

tcp\_app\_win

INTEGER。默认值为31。保留max（window/2^tcp\_app\_win, mss）数量的窗口为了应用缓冲。当为0时表示不需要缓冲。

tcp\_adv\_win\_scale

INTEGER。默认值为2。计算缓冲开销 bytes/2^tcp\_adv\_win\_scale（如果 tcp\_adv\_win\_scale > 0）或 bytes-bytes/2^(-tcp\_adv\_win\_scale)（如果 tcp\_adv\_win\_scale <= 0）。

tcp\_low\_latency

BOOLEAN。缺省值为0。允许 TCP/IP 栈适应在高吞吐量下低延时的情况；这个选项一般情况下是禁用的，但在构建 Beowulf 集群的时候，打开它很有帮助。

tcp\_westwood

BOOLEAN。缺省值为0。启用发送者端的拥塞控制算法，它可以维护对吞吐量的评估，并试图对带宽的整体利用情况进行优化；对于 WAN 通信来说应该启用这个选项。

tcp\_bic

BOOLEAN。缺省值为0。为快速长距离网络启用“Binary Increase Congestion”，这样可以更好地利用以 GB 速度进行操作的链接；对于 WAN 通信来说应该启用这个选项。

**IGMP相关服务参数配置**

即因特网组管理协议（Internet Group Management Protocol或简写IGMP）是用于管理因特网协议多播组成员的一种通信协议。IP主机和相邻的路由器利用IGMP来建立多播组的组成员。像ICMP用于单播连接一样，IGMP也是IP多播说明的一个完整部分。

igmp\_max\_memberships

INTEGER。主机上最多有多少个igmp（多播）套接字进行监听。

**INET相关服务参数配置**

inet\_peer\_gc\_maxtime

INTEGER。默认值是120。在较低的内存压力下碎片收集（Garbage Collector, GC）忽略存储内存池的频率。

**inet\_peer\_gc\_mintime**

**INTEGER** 。默认值是10，以jiffies（X86/Linux，约10ms）计。每一次碎片收集（Garbage Collector）之间的最小时间间隔。当服务器负载比较重（内存压力较大）的时候，调整这个间隔很有效。

inet\_peer\_maxttl

INTEGER。默认值是600，以jiffies计。Inet入口的最大生存期。在pool没有内存压力的情况下（比如，pool 中 Inet 入口数量很少的时候），未使用的Inet入口经过一段时间就会过期。

inet\_peer\_minttl

INTEGER。默认值是120，以jiffies计。Inet入口的最小生存期。这个时间必须比 net.ipv4.inet\_peer\_threshold 参数小。

inet\_peer\_threshold

INTEGER。默认值是65644。设定Inet的存储容量，当这个限制达到后，Inet入口将被丢弃，使用 net.ipv4.inet\_peer\_gc\_mintime 限制超时。

**实战**

以一台 4CPU、8GB 内存，CenterOS 5.X 的Web应用服务器为例，实战说明上述配置参数，以典型（片段）供参照：

net.ipv4.udp\_wmem\_min = 4096   
net.ipv4.udp\_rmem\_min = 4096   
net.ipv4.udp\_mem = 774240    1032320    1548480   
net.ipv4.cipso\_rbm\_strictvalid = 1   
net.ipv4.cipso\_rbm\_optfmt = 0   
net.ipv4.cipso\_cache\_bucket\_size = 10   
net.ipv4.cipso\_cache\_enable = 1   
net.ipv4.tcp\_slow\_start\_after\_idle = 1   
net.ipv4.tcp\_dma\_copybreak = 4096   
net.ipv4.tcp\_workaround\_signed\_windows = 0   
net.ipv4.tcp\_base\_mss = 512   
net.ipv4.tcp\_mtu\_probing = 0   
net.ipv4.tcp\_abc = 0   
net.ipv4.tcp\_congestion\_control = bic   
net.ipv4.tcp\_tso\_win\_divisor = 3   
net.ipv4.tcp\_moderate\_rcvbuf = 1   
net.ipv4.tcp\_no\_metrics\_save = 0   
net.ipv4.tcp\_low\_latency = 0   
net.ipv4.tcp\_frto = 0   
**net.ipv4.tcp\_tw\_reuse = 1**net.ipv4.tcp\_adv\_win\_scale = 2   
net.ipv4.tcp\_app\_win = 31   
**net.ipv4.tcp\_rmem = 4096    87380    4194304   
net.ipv4.tcp\_wmem = 4096    16384    4194304   
net.ipv4.tcp\_mem = 196608    262144    393216**net.ipv4.tcp\_dsack = 1   
net.ipv4.tcp\_ecn = 0   
net.ipv4.tcp\_reordering = 3   
net.ipv4.tcp\_fack = 1   
**net.ipv4.tcp\_orphan\_retries = 0**net.ipv4.inet\_peer\_gc\_maxtime = 120   
**net.ipv4.inet\_peer\_gc\_mintime = 10**net.ipv4.inet\_peer\_maxttl = 600   
net.ipv4.inet\_peer\_minttl = 120   
net.ipv4.inet\_peer\_threshold = 65664   
net.ipv4.igmp\_max\_msf = 10   
net.ipv4.igmp\_max\_memberships = 20   
net.ipv4.route.secret\_interval = 600   
net.ipv4.route.min\_adv\_mss = 256   
net.ipv4.route.min\_pmtu = 552   
net.ipv4.route.mtu\_expires = 600   
net.ipv4.route.gc\_elasticity = 8   
net.ipv4.route.error\_burst = 5000   
net.ipv4.route.error\_cost = 1000   
net.ipv4.route.redirect\_silence = 20480   
net.ipv4.route.redirect\_number = 9   
net.ipv4.route.redirect\_load = 20   
net.ipv4.route.gc\_interval = 60   
net.ipv4.route.gc\_timeout = 300   
net.ipv4.route.gc\_min\_interval\_ms = 500   
net.ipv4.route.gc\_min\_interval = 0   
net.ipv4.route.max\_size = 4194304   
net.ipv4.route.gc\_thresh = 262144   
net.ipv4.route.max\_delay = 10   
net.ipv4.route.min\_delay = 2   
net.ipv4.icmp\_ratemask = 6168   
net.ipv4.icmp\_ratelimit = 1000   
net.ipv4.icmp\_errors\_use\_inbound\_ifaddr = 0   
net.ipv4.icmp\_ignore\_bogus\_error\_responses = 1   
**net.ipv4.icmp\_echo\_ignore\_broadcasts = 1   
net.ipv4.icmp\_echo\_ignore\_all = 0   
net.ipv4.tcp\_max\_syn\_backlog = 8192**net.ipv4.tcp\_rfc1337 = 0   
net.ipv4.tcp\_stdurg = 0   
net.ipv4.tcp\_abort\_on\_overflow = 0   
**net.ipv4.tcp\_tw\_recycle = 0   
net.ipv4.tcp\_syncookies = 1   
net.ipv4.tcp\_fin\_timeout = 15   
net.ipv4.tcp\_retries2 = 15**net.ipv4.tcp\_retries1 = 3   
**net.ipv4.tcp\_keepalive\_intvl = 75   
net.ipv4.tcp\_keepalive\_probes = 9   
net.ipv4.tcp\_keepalive\_time = 1800   
net.ipv4.tcp\_max\_tw\_buckets = 180000   
net.ipv4.tcp\_max\_orphans = 65536   
net.ipv4.tcp\_synack\_retries = 5   
net.ipv4.tcp\_syn\_retries = 5**net.ipv4.ipfrag\_max\_dist = 64   
net.ipv4.ipfrag\_secret\_interval = 600   
**net.ipv4.ipfrag\_time = 30   
net.ipv4.ipfrag\_low\_thresh = 196608   
net.ipv4.ipfrag\_high\_thresh = 262144   
net.ipv4.ip\_local\_port\_range = 1024    65000**net.ipv4.ip\_dynaddr = 0   
net.ipv4.ip\_nonlocal\_bind = 0   
net.ipv4.ip\_no\_pmtu\_disc = 0   
net.ipv4.ip\_default\_ttl = 64   
net.ipv4.ip\_forward = 0   
**net.ipv4.tcp\_retrans\_collapse = 0**net.ipv4.tcp\_sack = 1   
net.ipv4.tcp\_window\_scaling = 1   
net.ipv4.tcp\_timestamps = 1