
Subject: Re: [PATCH] [NETNS49] support for per/namespace routing cache cleanup

Posted by [Daniel Lezcano](#) on Wed, 17 Oct 2007 11:46:23 GMT

[View Forum Message](#) <> [Reply to Message](#)

Denis V. Lunev wrote:

- > /proc/sys/net/route/flush should be accessible inside the net namespace.
- > Though, the complete opening of this file will result in a DoS or
- > significant entire host slowdown if a namespace process will continually
- > flush routes.
- >
- > This patch introduces per/namespace route flush facility.
- >
- > Each namespace wanted to flush a cache copies global generation count to
- > itself and starts the timer. The cache is dropped for a specific namespace
- > iff the namespace counter is greater or equal global ones.
- >
- > So, in general, unwanted namespaces do nothing. They hold very old low
- > counter and they are unaffected by the requested cleanup.
- >
- > Signed-off-by: Denis V. Lunev <den@openvz.org>

That's right and that will happen when manipulating ip addresses of the network devices too. But I am not comfortable with your patchset. It touches the routing flush function too hardly and it uses current->nsproxy->net_ns.

IMHO we should have two flush functions. One taking a network namespace parameter and one without the network namespace parameter. The first one is called when a write to /proc/sys/net/ipv4/route/flush is done (we must use the network namespace of the writer) or when a interface address is changed or shutdown|up. The last one is called by the timer, so we have a global timer flushing the routing cache for all the namespaces.

```
> diff --git a/include/net/net_namespace.h b/include/net/net_namespace.h
> index 85abf14..b492ce8 100644
> --- a/include/net/net_namespace.h
> +++ b/include/net/net_namespace.h
> @@ -143,6 +143,8 @@ struct net {
>
>
> /* iptable_filter.c */
> struct xt_table *ip_packet_filter;
> +
> + unsigned long rt_flush_required;
> };
>
> extern struct net init_net;
```

```

> diff --git a/net/ipv4/route.c b/net/ipv4/route.c
> index f9a59ff..413587c 100644
> --- a/net/ipv4/route.c
> +++ b/net/ipv4/route.c
> @@ -91,6 +91,7 @@
> #include <linux/jhash.h>
> #include <linux/rcupdate.h>
> #include <linux/times.h>
> +#include <linux/nsproxy.h>
> #include <net/net_namespace.h>
> #include <net/protocol.h>
> #include <net/ip.h>
> @@ -642,6 +643,51 @@ static void rt_check_expire(unsigned long dummy)
>   mod_timer(&rt_periodic_timer, jiffies + ip_rt_gc_interval);
> }
>
> +
> +static DEFINE_SPINLOCK(rt_flush_lock);
> +
> +#ifdef CONFIG_NET_NS
> +static unsigned long rt_flush_gen;
> +
> +/* called under rt_flush_lock */
> +static void rt_flush_required_set(struct net *net)
> +{
> + /*
> + * If the global generation rt_flush_gen is equal to G, then
> + * the pass considering entries labelled by G is yet to come.
> + */
> + net->rt_flush_required = rt_flush_gen;
> +}
> +
> +static unsigned long rt_flush_required_reset(void)
> +{
> + unsigned long g;
> +
> + spin_lock_bh(&rt_flush_lock);
> + g = rt_flush_gen++;
> + spin_unlock_bh(&rt_flush_lock);
> + return g;
> +}
> +
> +static int rt_flush_required_check(struct net *net, unsigned long gen)
> +{
> + /* can be checked without the lock */
> + return net->rt_flush_required >= gen;
> +}
> +

```

```

> +#else
> +
> +static void rt_flush_required_reset(struct net *net)
> +{
> +}
> +
> +static unsigned long rt_flush_required_reset(void)
> +{
> + return 0;
> +}
> +#endif
> +
> +
> + /* This can run from both BH and non-BH contexts, the latter
> +  * in the case of a forced flush event.
> +  */
> @@ -649,16 +695,46 @@ static void rt_run_flush(unsigned long dummy)
> {
> int i;
> struct rtable *rth, *next;
> + unsigned long gen;
>
> rt_deadline = 0;
>
> get_random_bytes(&rt_hash_rnd, 4);
> + gen = rt_flush_required_reset();
>
> for (i = rt_hash_mask; i >= 0; i--) {
> +#ifdef CONFIG_NET_NS
> + struct rtable **prev, *p, *tail;
> +
> + spin_lock_bh(rt_hash_lock_addr(i));
> + rth = rt_hash_table[i].chain;
> +
> + /* defer releasing the head of the list after spin_unlock */
> + for (tail = rth; tail; tail = tail->u.dst.rt_next)
> + if (!rt_flush_required_check(tail->fl.fl_net, gen))
> + break;
> + if (rth != tail)
> + rt_hash_table[i].chain = tail;
> +
> + /* call rt_free on entries after the tail requiring flush */
> + prev = &rt_hash_table[i].chain;
> + for (p = *prev; p; p = next) {
> + next = p->u.dst.rt_next;
> + if (!rt_flush_required_check(p->fl.fl_net, gen)) {
> + prev = &p->u.dst.rt_next;
> + } else {

```

```

> + *prev = next;
> + rt_free(p);
> + }
> + }
> + #else
> spin_lock_bh(rt_hash_lock_addr(i));
> rth = rt_hash_table[i].chain;
> if (rth)
> rt_hash_table[i].chain = NULL;
> + tail = NULL;
> +
> + #endif
> spin_unlock_bh(rt_hash_lock_addr(i));
>
> for (; rth; rth = next) {
> @@ -668,8 +744,6 @@ static void rt_run_flush(unsigned long dummy)
> }
> }
>
> -static DEFINE_SPINLOCK(rt_flush_lock);
> -
> void rt_cache_flush(int delay)
> {
> unsigned long now = jiffies;
> @@ -697,6 +771,8 @@ void rt_cache_flush(int delay)
> delay = tmo;
> }
>
> + rt_flush_required_set(current->nsproxy->net_ns);
> +
> if (delay <= 0) {
> spin_unlock_bh(&rt_flush_lock);
> rt_run_flush(0);

```
