Subject: [PATCH net-next 0/9] selective (per/namespace) flush of rt_cache
Posted by den on Fri, 04 Jul 2008 13:16:00 GMT
View Forum Message <> Reply to Message

Hello, Dave!

This series of patches implements selective rt cache flushing to make
sure that in one namespace we'll not been able to affect the performance
of other from the user space.

Regards,
Den

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH net-next 1/9] netns: add namespace parameter to rt_cache_flush
Posted by den on Fri, 04 Jul 2008 13:17:04 GMT
View Forum Message <> Reply to Message

Signed-off-by: Denis V. Lunev <den@openvz.org>
---
include/net/route.h | 2 +-
net/ipv4/arp.c | 2 +-
net/ipv4/devinet.c | 8 +++++---
net/ipv4/fib_frontend.c | 17 +++++++++--------
net/ipv4/fib_hash.c | 6 +++---
net/ipv4/fib_rules.c | 2 +-
net/ipv4/fib_trie.c | 6 +++---
net/ipv4/route.c | 8 ++++----
8 files changed, 27 insertions(+), 24 deletions(-)

diff --git a/include/net/route.h b/include/net/route.h
index fc836ff..3140cc5 100644
--- a/include/net/route.h
+++ b/include/net/route.h
@@ -111,7 +111,7 @@ struct in_device;
extern int ip_rt_init(void);
extern void ip_rt_redirect(__be32 old_gw, __be32 dst, __be32 new_gw,
__be32 src, struct net_device *dev);
-extern void rt_cache_flush(int how);
+extern void rt_cache_flush(struct net *net, int how);
extern int __ip_route_output_key(struct net *, struct rtable **, const struct flowi *flp);
extern int ip_route_output_key(struct net *, struct rtable **, struct flowi *flp);
extern int ip_route_output_flow(struct net *, struct rtable **rp, struct flowi *flp, struct sock *sk, int

```
flags);
diff --git a/net/ipv4/arp.c b/net/ipv4/arp.c
index 20c515a..29df75a 100644
--- a/net/ipv4/arp.c
+++ b/net/ipv4/arp.c
@@ -1197,7 +1197,7 @@ static int arp_netdev_event(struct notifier_block *this, unsigned long
event, vo
  switch (event) {
  case NETDEV_CHANGEADDR:
   neigh_changeaddr(&arp_tbl, dev);
-  rt_cache_flush(0);
+  rt_cache_flush(dev_net(dev), 0);
   break;
  default:
   break;
diff --git a/net/ipv4/devinet.c b/net/ipv4/devinet.c
index 9de2514..2e667e2 100644
--- a/net/ipv4/devinet.c
+++ b/net/ipv4/devinet.c
@@ -1348,7 +1348,7 @@ static int devinet_sysctl_forward(ctl_table *ctl, int write,
     dev_disable_lro(idev->dev);
    }
    rtnl_unlock();
-   rt_cache_flush(0);
+   rt_cache_flush(net, 0);
   }
  }

@@ -1362,9 +1362,10 @@ int ipv4_doint_and_flush(ctl_table *ctl, int write,
  int *valp = ctl->data;
  int val = *valp;
  int ret = proc_dointvec(ctl, write, filp, buffer, lenp, ppos);
+ struct net *net = ctl->extra2;

  if (write && *valp != val)
-  rt_cache_flush(0);
+  rt_cache_flush(net, 0);

  return ret;
 }
@@ -1375,9 +1376,10 @@ int ipv4_doint_and_flush_strategy(ctl_table *table, int __user *name,
int nlen,
 {
  int ret = devinet_conf_sysctl(table, name, nlen, oldval, oldlenp,
          newval, newlen);
+ struct net *net = table->extra2;

  if (ret == 1)
```

```
-  rt_cache_flush(0);
+  rt_cache_flush(net, 0);

  return ret;
 }
diff --git a/net/ipv4/fib_frontend.c b/net/ipv4/fib_frontend.c
index 5ad01d6..65c1503 100644
--- a/net/ipv4/fib_frontend.c
+++ b/net/ipv4/fib_frontend.c
@@ -144,7 +144,7 @@ static void fib_flush(struct net *net)
  }

  if (flushed)
-  rt_cache_flush(-1);
+  rt_cache_flush(net, -1);
 }

 /*
@@ -897,21 +897,22 @@ static void fib_disable_ip(struct net_device *dev, int force)
 {
  if (fib_sync_down_dev(dev, force))
   fib_flush(dev_net(dev));
- rt_cache_flush(0);
+ rt_cache_flush(dev_net(dev), 0);
  arp_ifdown(dev);
 }

 static int fib_inetaddr_event(struct notifier_block *this, unsigned long event, void *ptr)
 {
  struct in_ifaddr *ifa = (struct in_ifaddr*)ptr;
+ struct net_device *dev = ifa->ifa_dev->dev;

  switch (event) {
  case NETDEV_UP:
   fib_add_ifaddr(ifa);
 #ifdef CONFIG_IP_ROUTE_MULTIPATH
-  fib_sync_up(ifa->ifa_dev->dev);
+  fib_sync_up(dev);
 #endif
-  rt_cache_flush(-1);
+  rt_cache_flush(dev_net(dev), -1);
   break;
  case NETDEV_DOWN:
   fib_del_ifaddr(ifa);
@@ -919,9 +920,9 @@ static int fib_inetaddr_event(struct notifier_block *this, unsigned long
event,
    /* Last address was deleted from this interface.
       Disable IP.
```

```
     */
-   fib_disable_ip(ifa->ifa_dev->dev, 1);
+   fib_disable_ip(dev, 1);
   } else {
-   rt_cache_flush(-1);
+   rt_cache_flush(dev_net(dev), -1);
   }
   break;
  }
@@ -949,14 +950,14 @@ static int fib_netdev_event(struct notifier_block *this, unsigned long
event, vo
 #ifdef CONFIG_IP_ROUTE_MULTIPATH
   fib_sync_up(dev);
 #endif
-  rt_cache_flush(-1);
+  rt_cache_flush(dev_net(dev), -1);
   break;
  case NETDEV_DOWN:
   fib_disable_ip(dev, 0);
   break;
  case NETDEV_CHANGEMTU:
  case NETDEV_CHANGE:
-  rt_cache_flush(0);
+  rt_cache_flush(dev_net(dev), 0);
   break;
  }
  return NOTIFY_DONE;
diff --git a/net/ipv4/fib_hash.c b/net/ipv4/fib_hash.c
index eeec4bf..c8cac6c 100644
--- a/net/ipv4/fib_hash.c
+++ b/net/ipv4/fib_hash.c
@@ -472,7 +472,7 @@ static int fn_hash_insert(struct fib_table *tb, struct fib_config *cfg)

    fib_release_info(fi_drop);
    if (state & FA_S_ACCESSED)
-    rt_cache_flush(-1);
+    rt_cache_flush(cfg->fc_nlinfo.nl_net, -1);
    rtmsg_fib(RTM_NEWROUTE, key, fa, cfg->fc_dst_len, tb->tb_id,
      &cfg->fc_nlinfo, NLM_F_REPLACE);
    return 0;
@@ -532,7 +532,7 @@ static int fn_hash_insert(struct fib_table *tb, struct fib_config *cfg)

  if (new_f)
   fz->fz_nent++;
- rt_cache_flush(-1);
+ rt_cache_flush(cfg->fc_nlinfo.nl_net, -1);

  rtmsg_fib(RTM_NEWROUTE, key, new_fa, cfg->fc_dst_len, tb->tb_id,
```

```
     &cfg->fc_nlinfo, 0);
@@ -614,7 +614,7 @@ static int fn_hash_delete(struct fib_table *tb, struct fib_config *cfg)
   write_unlock_bh(&fib_hash_lock);

   if (fa->fa_state & FA_S_ACCESSED)
-   rt_cache_flush(-1);
+   rt_cache_flush(cfg->fc_nlinfo.nl_net, -1);
   fn_free_alias(fa, f);
   if (kill_fn) {
    fn_free_node(f);
diff --git a/net/ipv4/fib_rules.c b/net/ipv4/fib_rules.c
index 1fb5687..bc05de4 100644
--- a/net/ipv4/fib_rules.c
+++ b/net/ipv4/fib_rules.c
@@ -260,7 +260,7 @@ static size_t fib4_rule_nlmsg_payload(struct fib_rule *rule)

 static void fib4_rule_flush_cache(void)
 {
- rt_cache_flush(-1);
+ rt_cache_flush(&init_net, -1);
 }

 static struct fib_rules_ops fib4_rules_ops_template = {
diff --git a/net/ipv4/fib_trie.c b/net/ipv4/fib_trie.c
index 394db9c..d16ae46 100644
--- a/net/ipv4/fib_trie.c
+++ b/net/ipv4/fib_trie.c
@@ -1271,7 +1271,7 @@ static int fn_trie_insert(struct fib_table *tb, struct fib_config *cfg)

    fib_release_info(fi_drop);
    if (state & FA_S_ACCESSED)
-    rt_cache_flush(-1);
+    rt_cache_flush(cfg->fc_nlinfo.nl_net, -1);
    rtmsg_fib(RTM_NEWROUTE, htonl(key), new_fa, plen,
     tb->tb_id, &cfg->fc_nlinfo, NLM_F_REPLACE);

@@ -1316,7 +1316,7 @@ static int fn_trie_insert(struct fib_table *tb, struct fib_config *cfg)
  list_add_tail_rcu(&new_fa->fa_list,
     (fa ? &fa->fa_list : fa_head));

- rt_cache_flush(-1);
+ rt_cache_flush(cfg->fc_nlinfo.nl_net, -1);
  rtmsg_fib(RTM_NEWROUTE, htonl(key), new_fa, plen, tb->tb_id,
     &cfg->fc_nlinfo, 0);
 succeeded:
@@ -1664,7 +1664,7 @@ static int fn_trie_delete(struct fib_table *tb, struct fib_config *cfg)
   trie_leaf_remove(t, l);
```

```
  if (fa->fa_state & FA_S_ACCESSED)
-  rt_cache_flush(-1);
+  rt_cache_flush(cfg->fc_nlinfo.nl_net, -1);

  fib_release_info(fa->fa_info);
  alias_free_mem_rcu(fa);
diff --git a/net/ipv4/route.c b/net/ipv4/route.c
index fe3a022..cedc366 100644
--- a/net/ipv4/route.c
+++ b/net/ipv4/route.c
@@ -791,7 +791,7 @@ static void rt_cache_invalidate(void)
  * delay < 0  : invalidate cache (fast : entries will be deleted later)
  * delay >= 0 : invalidate & flush cache (can be long)
  */
-void rt_cache_flush(int delay)
+void rt_cache_flush(struct net *net, int delay)
 {
  rt_cache_invalidate();
  if (delay >= 0)
@@ -2825,7 +2825,7 @@ done:

 void ip_rt_multicast_event(struct in_device *in_dev)
 {
- rt_cache_flush(0);
+ rt_cache_flush(dev_net(in_dev->dev), 0);
 }

 #ifdef CONFIG_SYSCTL
@@ -2837,7 +2837,7 @@ static int ipv4_sysctl_rtcache_flush(ctl_table *ctl, int write,
  if (write) {
   proc_dointvec(ctl, write, filp, buffer, lenp, ppos);
-  rt_cache_flush(flush_delay);
+  rt_cache_flush(&init_net, flush_delay);
   return 0;
  }

@@ -2857,7 +2857,7 @@ static int ipv4_sysctl_rtcache_flush_strategy(ctl_table *table,
   return -EINVAL;
  if (get_user(delay, (int __user *)newval))
   return -EFAULT;
- rt_cache_flush(delay);
+ rt_cache_flush(&init_net, delay);
  return 0;
 }

--
1.5.3.rc5
```

Subject: [PATCH net-next 2/9] net: add fib_rules_ops to flush_cache method
Posted by den on Fri, 04 Jul 2008 13:17:05 GMT
View Forum Message <> Reply to Message

This is required to pass namespace context into rt_cache_flush called from
->flush_cache.

Signed-off-by: Denis V. Lunev <den@openvz.org>
---
 include/net/fib_rules.h | 2 +-
 net/core/fib_rules.c | 2 +-
 net/decnet/dn_rules.c | 2 +-
 net/ipv4/fib_rules.c | 4 ++--
 4 files changed, 5 insertions(+), 5 deletions(-)

```
diff --git a/include/net/fib_rules.h b/include/net/fib_rules.h
index a5c6ccc..c2bb5ca 100644
--- a/include/net/fib_rules.h
+++ b/include/net/fib_rules.h
@@ -62,7 +62,7 @@ struct fib_rules_ops

  /* Called after modifications to the rules set, must flush
   * the route cache if one exists. */
- void   (*flush_cache)(void);
+ void   (*flush_cache)(struct fib_rules_ops *ops);

  int   nlgroup;
  const struct nla_policy *policy;
diff --git a/net/core/fib_rules.c b/net/core/fib_rules.c
index e3e9ab0..1c2943a 100644
--- a/net/core/fib_rules.c
+++ b/net/core/fib_rules.c
@@ -69,7 +69,7 @@ static void rules_ops_put(struct fib_rules_ops *ops)
 static void flush_route_cache(struct fib_rules_ops *ops)
 {
  if (ops->flush_cache)
-  ops->flush_cache();
+  ops->flush_cache(ops);
 }

 int fib_rules_register(struct fib_rules_ops *ops)
```

```
diff --git a/net/decnet/dn_rules.c b/net/decnet/dn_rules.c
index 5b7539b..14fbca5 100644
--- a/net/decnet/dn_rules.c
+++ b/net/decnet/dn_rules.c
@@ -229,7 +229,7 @@ static u32 dn_fib_rule_default_pref(struct fib_rules_ops *ops)
  return 0;
 }

-static void dn_fib_rule_flush_cache(void)
+static void dn_fib_rule_flush_cache(struct fib_rules_ops *ops)
 {
  dn_rt_cache_flush(-1);
 }
diff --git a/net/ipv4/fib_rules.c b/net/ipv4/fib_rules.c
index bc05de4..6080d71 100644
--- a/net/ipv4/fib_rules.c
+++ b/net/ipv4/fib_rules.c
@@ -258,9 +258,9 @@ static size_t fib4_rule_nlmsg_payload(struct fib_rule *rule)
         + nla_total_size(4); /* flow */
 }

-static void fib4_rule_flush_cache(void)
+static void fib4_rule_flush_cache(struct fib_rules_ops *ops)
 {
- rt_cache_flush(&init_net, -1);
+ rt_cache_flush(ops->fro_net, -1);
 }

 static struct fib_rules_ops fib4_rules_ops_template = {
--
1.5.3.rc5
```

Subject: [PATCH net-next 3/9] ipv4: remove static flush_delay variable
Posted by den on Fri, 04 Jul 2008 13:17:06 GMT
View Forum Message <> Reply to Message

flush delay is used as an external storage for net.ipv4.route.flush sysctl
entry. It is write-only.

The ctl_table->data for this entry is used once. Fix this case to point
to the stack to remove global variable. Do this to avoid additional
variable on struct net in the next patch.

Possible race (as it was before) accessing this local variable is removed
using flush_mutex.

Signed-off-by: Denis V. Lunev <den@openvz.org>
---
 net/ipv4/route.c | 11 ++++++++---
 1 files changed, 8 insertions(+), 3 deletions(-)

diff --git a/net/ipv4/route.c b/net/ipv4/route.c
index cedc366..790de32 100644
--- a/net/ipv4/route.c
+++ b/net/ipv4/route.c
@@ -2829,14 +2829,20 @@ void ip_rt_multicast_event(struct in_device *in_dev)
 }

 #ifdef CONFIG_SYSCTL
-static int flush_delay;
-
 static int ipv4_sysctl_rtcache_flush(ctl_table *ctl, int write,
      struct file *filp, void __user *buffer,
      size_t *lenp, loff_t *ppos)
 {
  if (write) {
+ int flush_delay;
+ static DEFINE_MUTEX(flush_mutex);
+
+ mutex_lock(&flush_mutex);
+ ctl->data = &flush_delay;
   proc_dointvec(ctl, write, filp, buffer, lenp, ppos);
+ ctl->data = NULL;
+ mutex_unlock(&flush_mutex);
+
   rt_cache_flush(&init_net, flush_delay);
   return 0;
  }
@@ -2865,7 +2871,6 @@ ctl_table ipv4_route_table[] = {
  {
   .ctl_name = NET_IPV4_ROUTE_FLUSH,
   .procname = "flush",
- .data = &flush_delay,
   .maxlen = sizeof(int),
   .mode = 0200,
   .proc_handler = &ipv4_sysctl_rtcache_flush,
--
1.5.3.rc5

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH net-next 4/9] netns: register net.ipv4.route.flush in each namespace
Posted by den on Fri, 04 Jul 2008 13:17:07 GMT
View Forum Message <> Reply to Message

Signed-off-by: Denis V. Lunev <den@openvz.org>

```
---
 include/net/netns/ipv4.h |    1 +
 net/ipv4/route.c         |   79 +++++++++++++++++++++++++++++++++++++++++++------
 2 files changed, 70 insertions(+), 10 deletions(-)

diff --git a/include/net/netns/ipv4.h b/include/net/netns/ipv4.h
index 6ef90b5..a29adf1 100644
--- a/include/net/netns/ipv4.h
+++ b/include/net/netns/ipv4.h
@@ -18,6 +18,7 @@ struct netns_ipv4 {
  struct ctl_table_header *forw_hdr;
  struct ctl_table_header *frags_hdr;
  struct ctl_table_header *ipv4_hdr;
+ struct ctl_table_header *route_hdr;
 #endif
  struct ipv4_devconf *devconf_all;
  struct ipv4_devconf *devconf_dflt;
diff --git a/net/ipv4/route.c b/net/ipv4/route.c
index 790de32..6fe799d 100644
--- a/net/ipv4/route.c
+++ b/net/ipv4/route.c
@@ -2835,6 +2835,7 @@ static int ipv4_sysctl_rtcache_flush(ctl_table *ctl, int write,
 {
  if (write) {
   int flush_delay;
+  struct net *net;
   static DEFINE_MUTEX(flush_mutex);

   mutex_lock(&flush_mutex);
@@ -2843,7 +2844,8 @@ static int ipv4_sysctl_rtcache_flush(ctl_table *ctl, int write,
   ctl->data = NULL;
   mutex_unlock(&flush_mutex);

-  rt_cache_flush(&init_net, flush_delay);
+  net = (struct net *)ctl->extra1;
+  rt_cache_flush(net, flush_delay);
   return 0;
```

```
 }

@@ -2859,24 +2861,18 @@ static int ipv4_sysctl_rtcache_flush_strategy(ctl_table *table,
       size_t newlen)
 {
  int delay;
+ struct net *net;
  if (newlen != sizeof(int))
   return -EINVAL;
  if (get_user(delay, (int __user *)newval))
   return -EFAULT;
- rt_cache_flush(&init_net, delay);
+ net = (struct net *)table->extra1;
+ rt_cache_flush(net, delay);
  return 0;
 }

 ctl_table ipv4_route_table[] = {
  {
-  .ctl_name  = NET_IPV4_ROUTE_FLUSH,
-  .procname = "flush",
-  .maxlen  = sizeof(int),
-  .mode  = 0200,
-  .proc_handler = &ipv4_sysctl_rtcache_flush,
-  .strategy = &ipv4_sysctl_rtcache_flush_strategy,
- },
- {
   .ctl_name = NET_IPV4_ROUTE_GC_THRESH,
   .procname = "gc_thresh",
   .data  = &ipv4_dst_ops.gc_thresh,
@@ -3014,6 +3010,66 @@ ctl_table ipv4_route_table[] = {
  },
  { .ctl_name = 0 }
 };
+
+static __net_initdata struct ctl_path ipv4_route_path[] = {
+ { .procname = "net", .ctl_name = CTL_NET, },
+ { .procname = "ipv4", .ctl_name = NET_IPV4, },
+ { .procname = "route", .ctl_name = NET_IPV4_ROUTE, },
+ { },
+};
+
+
+static struct ctl_table ipv4_route_flush_table[] = {
+ {
+  .ctl_name  = NET_IPV4_ROUTE_FLUSH,
+  .procname = "flush",
+  .maxlen  = sizeof(int),
```

```
+  .mode  = 0200,
+  .proc_handler = &ipv4_sysctl_rtcache_flush,
+  .strategy = &ipv4_sysctl_rtcache_flush_strategy,
+ },
+ { .ctl_name = 0 },
+};
+
+static __net_init int sysctl_route_net_init(struct net *net)
+{
+ struct ctl_table *tbl;
+
+ tbl = ipv4_route_flush_table;
+ if (net != &init_net) {
+  tbl = kmemdup(tbl, sizeof(ipv4_route_flush_table), GFP_KERNEL);
+  if (tbl == NULL)
+   goto err_dup;
+ }
+ tbl[0].extra1 = net;
+
+ net->ipv4.route_hdr =
+  register_net_sysctl_table(net, ipv4_route_path, tbl);
+ if (net->ipv4.route_hdr == NULL)
+  goto err_reg;
+ return 0;
+
+err_reg:
+ if (tbl != ipv4_route_flush_table)
+  kfree(tbl);
+err_dup:
+ return -ENOMEM;
+}
+
+static __net_exit void sysctl_route_net_exit(struct net *net)
+{
+ struct ctl_table *tbl;
+
+ tbl = net->ipv4.route_hdr->ctl_table_arg;
+ unregister_net_sysctl_table(net->ipv4.route_hdr);
+ BUG_ON(tbl == ipv4_route_flush_table);
+ kfree(tbl);
+}
+
+static __net_initdata struct pernet_operations sysctl_route_ops = {
+ .init = sysctl_route_net_init,
+ .exit = sysctl_route_net_exit,
+};
 #endif
```

```
 #ifdef CONFIG_NET_CLS_ROUTE
@@ -3090,6 +3146,9 @@ int __init ip_rt_init(void)
 #endif
  rtnl_register(PF_INET, RTM_GETROUTE, inet_rtm_getroute, NULL);

+#ifdef CONFIG_SYSCTL
+ register_pernet_subsys(&sysctl_route_ops);
+#endif
  return rc;
 }

--
1.5.3.rc5
```

Subject: [PATCH net-next 5/9] netns: make rt_secret_rebuild timer per namespace
Posted by den on Fri, 04 Jul 2008 13:17:08 GMT
View Forum Message <> Reply to Message

```
Signed-off-by: Denis V. Lunev <den@openvz.org>
---
 include/net/netns/ipv4.h |    2 ++
 net/ipv4/route.c         |   40 ++++++++++++++++++++++++++++++++----------
 2 files changed, 32 insertions(+), 10 deletions(-)

diff --git a/include/net/netns/ipv4.h b/include/net/netns/ipv4.h
index a29adf1..356617f 100644
--- a/include/net/netns/ipv4.h
+++ b/include/net/netns/ipv4.h
@@ -46,5 +46,7 @@ struct netns_ipv4 {
  int sysctl_icmp_ratelimit;
  int sysctl_icmp_ratemask;
  int sysctl_icmp_errors_use_inbound_ifaddr;
+
+ struct timer_list rt_secret_timer;
 };
 #endif
diff --git a/net/ipv4/route.c b/net/ipv4/route.c
index 6fe799d..f99d9db 100644
--- a/net/ipv4/route.c
+++ b/net/ipv4/route.c
@@ -132,7 +132,6 @@ static int ip_rt_secret_interval __read_mostly = 10 * 60 * HZ;
```

```
 static void rt_worker_func(struct work_struct *work);
 static DECLARE_DELAYED_WORK(expires_work, rt_worker_func);
-static struct timer_list rt_secret_timer;

 /*
  * Interface to generic destination cache.
@@ -801,10 +800,11 @@ void rt_cache_flush(struct net *net, int delay)
 /*
  * We change rt_genid and let gc do the cleanup
  */
-static void rt_secret_rebuild(unsigned long dummy)
+static void rt_secret_rebuild(unsigned long __net)
 {
+ struct net *net = (struct net *)__net;
  rt_cache_invalidate();
- mod_timer(&rt_secret_timer, jiffies + ip_rt_secret_interval);
+ mod_timer(&net->ipv4.rt_secret_timer, jiffies + ip_rt_secret_interval);
 }

 /*
@@ -3072,6 +3072,31 @@ static __net_initdata struct pernet_operations sysctl_route_ops = {
 };
 #endif

+
+static __net_init int rt_secret_timer_init(struct net *net)
+{
+ net->ipv4.rt_secret_timer.function = rt_secret_rebuild;
+ net->ipv4.rt_secret_timer.data = (unsigned long)net;
+ init_timer_deferrable(&net->ipv4.rt_secret_timer);
+
+ net->ipv4.rt_secret_timer.expires =
+  jiffies + net_random() % ip_rt_secret_interval +
+  ip_rt_secret_interval;
+ add_timer(&net->ipv4.rt_secret_timer);
+ return 0;
+}
+
+static __net_exit void rt_secret_timer_exit(struct net *net)
+{
+ del_timer_sync(&net->ipv4.rt_secret_timer);
+}
+
+static __net_initdata struct pernet_operations rt_secret_timer_ops = {
+ .init = rt_secret_timer_init,
+ .exit = rt_secret_timer_exit,
+};
+
```

```
+
 #ifdef CONFIG_NET_CLS_ROUTE
 struct ip_rt_acct *ip_rt_acct __read_mostly;
 #endif /* CONFIG_NET_CLS_ROUTE */
@@ -3124,19 +3149,14 @@ int __init ip_rt_init(void)
  devinet_init();
  ip_fib_init();

- rt_secret_timer.function = rt_secret_rebuild;
- rt_secret_timer.data = 0;
- init_timer_deferrable(&rt_secret_timer);
-
  /* All the timers, started at system startup tend
     to synchronize. Perturb it a bit.
   */
  schedule_delayed_work(&expires_work,
   net_random() % ip_rt_gc_interval + ip_rt_gc_interval);

- rt_secret_timer.expires = jiffies + net_random() % ip_rt_secret_interval +
-  ip_rt_secret_interval;
- add_timer(&rt_secret_timer);
+ if (register_pernet_subsys(&rt_secret_timer_ops))
+  printk(KERN_ERR "Unable to setup rt_secret_timer\n");

  if (ip_rt_proc_init())
   printk(KERN_ERR "Unable to create route proc files\n");
--
1.5.3.rc5
```

Subject: [PATCH net-next 6/9] netns: add struct net parameter to rt_cache_invalidate

Posted by den on Fri, 04 Jul 2008 13:17:09 GMT

View Forum Message <> Reply to Message

Signed-off-by: Denis V. Lunev <den@openvz.org>

```
---
 net/ipv4/route.c |    6 +++---
 1 files changed, 3 insertions(+), 3 deletions(-)

diff --git a/net/ipv4/route.c b/net/ipv4/route.c
index f99d9db..9725223 100644
--- a/net/ipv4/route.c
```

```
+++ b/net/ipv4/route.c
@@ -778,7 +778,7 @@ static void rt_worker_func(struct work_struct *work)
  * many times (2^24) without giving recent rt_genid.
  * Jenkins hash is strong enough that litle changes of rt_genid are OK.
  */
-static void rt_cache_invalidate(void)
+static void rt_cache_invalidate(struct net *net)
 {
  unsigned char shuffle;

@@ -792,7 +792,7 @@ static void rt_cache_invalidate(void)
  */
 void rt_cache_flush(struct net *net, int delay)
 {
- rt_cache_invalidate();
+ rt_cache_invalidate(net);
  if (delay >= 0)
   rt_do_flush(!in_softirq());
 }
@@ -803,7 +803,7 @@ void rt_cache_flush(struct net *net, int delay)
 static void rt_secret_rebuild(unsigned long __net)
 {
  struct net *net = (struct net *)__net;
- rt_cache_invalidate();
+ rt_cache_invalidate(net);
  mod_timer(&net->ipv4.rt_secret_timer, jiffies + ip_rt_secret_interval);
 }

--
1.5.3.rc5


_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers
```

Subject: [PATCH net-next 7/9] ipv4: pass current value of rt_genid into rt_hash
Posted by den on Fri, 04 Jul 2008 13:17:10 GMT

View Forum Message <> Reply to Message

Basically, there is no difference to atomic_read internally or pass it as
a parameter as rt_hash is inline.

Signed-off-by: Denis V. Lunev <den@openvz.org>

```
---
 net/ipv4/route.c |   28 ++++++++++++++++++----------
 1 files changed, 17 insertions(+), 11 deletions(-)
```

```
diff --git a/net/ipv4/route.c b/net/ipv4/route.c
index 9725223..e4e37ed 100644
--- a/net/ipv4/route.c
+++ b/net/ipv4/route.c
@@ -256,11 +256,12 @@ static DEFINE_PER_CPU(struct rt_cache_stat, rt_cache_stat);
 #define RT_CACHE_STAT_INC(field) \
  (__raw_get_cpu_var(rt_cache_stat).field++)

-static inline unsigned int rt_hash(__be32 daddr, __be32 saddr, int idx)
+static inline unsigned int rt_hash(__be32 daddr, __be32 saddr, int idx,
+  int genid)
 {
  return jhash_3words((__force u32)(__be32)(daddr),
        (__force u32)(__be32)(saddr),
-       idx, atomic_read(&rt_genid))
+       idx, genid)
   & rt_hash_mask;
 }

@@ -1180,7 +1181,8 @@ void ip_rt_redirect(__be32 old_gw, __be32 daddr, __be32 new_gw,

  for (i = 0; i < 2; i++) {
   for (k = 0; k < 2; k++) {
-   unsigned hash = rt_hash(daddr, skeys[i], ikeys[k]);
+   unsigned hash = rt_hash(daddr, skeys[i], ikeys[k],
+       atomic_read(&rt_genid));

    rthp=&rt_hash_table[hash].chain;

@@ -1295,7 +1297,8 @@ static struct dst_entry *ipv4_negative_advice(struct dst_entry *dst)
   } else if ((rt->rt_flags & RTCF_REDIRECTED) ||
       rt->u.dst.expires) {
    unsigned hash = rt_hash(rt->fl.fl4_dst, rt->fl.fl4_src,
-       rt->fl.oif);
+       rt->fl.oif,
+       atomic_read(&rt_genid));
 #if RT_CACHE_DEBUG >= 1
    printk(KERN_DEBUG "ipv4_negative_advice: redirect to "
        NIPQUAD_FMT "/%02x dropped\n",
@@ -1444,7 +1447,8 @@ unsigned short ip_rt_frag_needed(struct net *net, struct iphdr *iph,

  for (k = 0; k < 2; k++) {
   for (i = 0; i < 2; i++) {
-   unsigned hash = rt_hash(daddr, skeys[i], ikeys[k]);
+   unsigned hash = rt_hash(daddr, skeys[i], ikeys[k],
+       atomic_read(&rt_genid));
```

```
    rcu_read_lock();
    for (rth = rcu_dereference(rt_hash_table[hash].chain); rth;
@@ -1709,7 +1713,7 @@ static int ip_route_input_mc(struct sk_buff *skb, __be32 daddr,
__be32 saddr,
  RT_CACHE_STAT_INC(in_slow_mc);

  in_dev_put(in_dev);
- hash = rt_hash(daddr, saddr, dev->ifindex);
+ hash = rt_hash(daddr, saddr, dev->ifindex, atomic_read(&rt_genid));
  return rt_intern_hash(hash, rth, &skb->rtable);

 e_nobufs:
@@ -1870,7 +1874,7 @@ static int ip_mkroute_input(struct sk_buff *skb,
   return err;

  /* put it into the cache */
- hash = rt_hash(daddr, saddr, fl->iif);
+ hash = rt_hash(daddr, saddr, fl->iif, atomic_read(&rt_genid));
  return rt_intern_hash(hash, rth, &skb->rtable);
 }

@@ -2026,7 +2030,7 @@ local_input:
   rth->rt_flags  &= ~RTCF_LOCAL;
  }
  rth->rt_type = res.type;
- hash = rt_hash(daddr, saddr, fl.iif);
+ hash = rt_hash(daddr, saddr, fl.iif, atomic_read(&rt_genid));
  err = rt_intern_hash(hash, rth, &skb->rtable);
  goto done;

@@ -2077,7 +2081,7 @@ int ip_route_input(struct sk_buff *skb, __be32 daddr, __be32 saddr,

  net = dev_net(dev);
  tos &= IPTOS_RT_MASK;
- hash = rt_hash(daddr, saddr, iif);
+ hash = rt_hash(daddr, saddr, iif, atomic_read(&rt_genid));

  rcu_read_lock();
  for (rth = rcu_dereference(rt_hash_table[hash].chain); rth;
@@ -2266,7 +2270,8 @@ static int ip_mkroute_output(struct rtable **rp,
  int err = __mkroute_output(&rth, res, fl, oldflp, dev_out, flags);
  unsigned hash;
  if (err == 0) {
-  hash = rt_hash(oldflp->fl4_dst, oldflp->fl4_src, oldflp->oif);
+  hash = rt_hash(oldflp->fl4_dst, oldflp->fl4_src, oldflp->oif,
+         atomic_read(&rt_genid));
   err = rt_intern_hash(hash, rth, rp);
  }
```

```
@@ -2478,7 +2483,8 @@ int __ip_route_output_key(struct net *net, struct rtable **rp,
  unsigned hash;
  struct rtable *rth;

- hash = rt_hash(flp->fl4_dst, flp->fl4_src, flp->oif);
+ hash = rt_hash(flp->fl4_dst, flp->fl4_src, flp->oif,
+         atomic_read(&rt_genid));

  rcu_read_lock_bh();
  for (rth = rcu_dereference(rt_hash_table[hash].chain); rth;
--
1.5.3.rc5
```

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH net-next 8/9] netns: place rt_genid into struct net
Posted by den on Fri, 04 Jul 2008 13:17:11 GMT
View Forum Message <> Reply to Message

Signed-off-by: Denis V. Lunev <den@openvz.org>

```
---
 include/net/netns/ipv4.h |    1 +
 net/ipv4/route.c         |   76 ++++++++++++++++++++++++++++++++-------------
 2 files changed, 44 insertions(+), 33 deletions(-)

diff --git a/include/net/netns/ipv4.h b/include/net/netns/ipv4.h
index 356617f..a6ed838 100644
--- a/include/net/netns/ipv4.h
+++ b/include/net/netns/ipv4.h
@@ -48,5 +48,6 @@ struct netns_ipv4 {
  int sysctl_icmp_errors_use_inbound_ifaddr;

  struct timer_list rt_secret_timer;
+ atomic_t rt_genid;
 };
 #endif
diff --git a/net/ipv4/route.c b/net/ipv4/route.c
index e4e37ed..67c3ed7 100644
--- a/net/ipv4/route.c
+++ b/net/ipv4/route.c
@@ -250,7 +250,6 @@ static inline void rt_hash_lock_init(void)
 static struct rt_hash_bucket  *rt_hash_table __read_mostly;
 static unsigned   rt_hash_mask __read_mostly;
```

```
 static unsigned int  rt_hash_log  __read_mostly;
-static atomic_t   rt_genid __read_mostly;

 static DEFINE_PER_CPU(struct rt_cache_stat, rt_cache_stat);
 #define RT_CACHE_STAT_INC(field) \
@@ -265,6 +264,11 @@ static inline unsigned int rt_hash(__be32 daddr, __be32 saddr, int idx,
   & rt_hash_mask;
 }

+static inline int rt_genid(struct net *net)
+{
+ return atomic_read(&net->ipv4.rt_genid);
+}
+
 #ifdef CONFIG_PROC_FS
 struct rt_cache_iter_state {
  struct seq_net_private p;
@@ -334,7 +338,7 @@ static void *rt_cache_seq_start(struct seq_file *seq, loff_t *pos)
  struct rt_cache_iter_state *st = seq->private;
  if (*pos)
   return rt_cache_get_idx(seq, *pos - 1);
- st->genid = atomic_read(&rt_genid);
+ st->genid = rt_genid(seq_file_net(seq));
  return SEQ_START_TOKEN;
 }

@@ -681,6 +685,11 @@ static inline int compare_netns(struct rtable *rt1, struct rtable *rt2)
  return dev_net(rt1->u.dst.dev) == dev_net(rt2->u.dst.dev);
 }

+static inline int rt_is_expired(struct rtable *rth)
+{
+ return rth->rt_genid != rt_genid(dev_net(rth->u.dst.dev));
+}
+
 /*
  * Perform a full scan of hash table and free all entries.
  * Can be called by a softirq or a process.
@@ -736,7 +745,7 @@ static void rt_check_expire(void)
    continue;
   spin_lock_bh(rt_hash_lock_addr(i));
   while ((rth = *rthp) != NULL) {
-   if (rth->rt_genid != atomic_read(&rt_genid)) {
+   if (rt_is_expired(rth)) {
     *rthp = rth->u.dst.rt_next;
     rt_free(rth);
     continue;
@@ -784,7 +793,7 @@ static void rt_cache_invalidate(struct net *net)
```

```
  unsigned char shuffle;

  get_random_bytes(&shuffle, sizeof(shuffle));
- atomic_add(shuffle + 1U, &rt_genid);
+ atomic_add(shuffle + 1U, &net->ipv4.rt_genid);
 }

 /*
@@ -881,7 +890,7 @@ static int rt_garbage_collect(struct dst_ops *ops)
    rthp = &rt_hash_table[k].chain;
    spin_lock_bh(rt_hash_lock_addr(k));
    while ((rth = *rthp) != NULL) {
-    if (rth->rt_genid == atomic_read(&rt_genid) &&
+    if (!rt_is_expired(rth) &&
      !rt_may_expire(rth, tmo, expire)) {
      tmo >>= 1;
      rthp = &rth->u.dst.rt_next;
@@ -963,7 +972,7 @@ restart:

  spin_lock_bh(rt_hash_lock_addr(hash));
  while ((rth = *rthp) != NULL) {
-  if (rth->rt_genid != atomic_read(&rt_genid)) {
+  if (rt_is_expired(rth)) {
    *rthp = rth->u.dst.rt_next;
    rt_free(rth);
    continue;
@@ -1139,7 +1148,7 @@ static void rt_del(unsigned hash, struct rtable *rt)
  spin_lock_bh(rt_hash_lock_addr(hash));
  ip_rt_put(rt);
  while ((aux = *rthp) != NULL) {
-  if (aux == rt || (aux->rt_genid != atomic_read(&rt_genid))) {
+  if (aux == rt || rt_is_expired(aux)) {
    *rthp = aux->u.dst.rt_next;
    rt_free(aux);
    continue;
@@ -1182,7 +1191,7 @@ void ip_rt_redirect(__be32 old_gw, __be32 daddr, __be32 new_gw,
  for (i = 0; i < 2; i++) {
   for (k = 0; k < 2; k++) {
    unsigned hash = rt_hash(daddr, skeys[i], ikeys[k],
-      atomic_read(&rt_genid));
+      rt_genid(net));

    rthp=&rt_hash_table[hash].chain;

@@ -1194,7 +1203,7 @@ void ip_rt_redirect(__be32 old_gw, __be32 daddr, __be32 new_gw,
        rth->fl.fl4_src != skeys[i] ||
        rth->fl.oif != ikeys[k] ||
        rth->fl.iif != 0 ||
```

```
-       rth->rt_genid != atomic_read(&rt_genid) ||
+        rt_is_expired(rth) ||
         !net_eq(dev_net(rth->u.dst.dev), net)) {
      rthp = &rth->u.dst.rt_next;
      continue;
@@ -1233,7 +1242,7 @@ void ip_rt_redirect(__be32 old_gw, __be32 daddr, __be32 new_gw,
     rt->u.dst.neighbour = NULL;
     rt->u.dst.hh  = NULL;
     rt->u.dst.xfrm  = NULL;
-    rt->rt_genid  = atomic_read(&rt_genid);
+    rt->rt_genid  = rt_genid(net);
     rt->rt_flags  |= RTCF_REDIRECTED;

     /* Gateway is different ... */
@@ -1298,7 +1307,7 @@ static struct dst_entry *ipv4_negative_advice(struct dst_entry *dst)
       rt->u.dst.expires) {
    unsigned hash = rt_hash(rt->fl.fl4_dst, rt->fl.fl4_src,
       rt->fl.oif,
-      atomic_read(&rt_genid));
+      rt_genid(dev_net(dst->dev)));
 #if RT_CACHE_DEBUG >= 1
    printk(KERN_DEBUG "ipv4_negative_advice: redirect to "
       NIPQUAD_FMT "/%02x dropped\n",
@@ -1448,7 +1457,7 @@ unsigned short ip_rt_frag_needed(struct net *net, struct iphdr *iph,
  for (k = 0; k < 2; k++) {
   for (i = 0; i < 2; i++) {
    unsigned hash = rt_hash(daddr, skeys[i], ikeys[k],
-      atomic_read(&rt_genid));
+      rt_genid(net));

    rcu_read_lock();
    for (rth = rcu_dereference(rt_hash_table[hash].chain); rth;
@@ -1463,7 +1472,7 @@ unsigned short ip_rt_frag_needed(struct net *net, struct iphdr *iph,
         rth->fl.iif != 0 ||
         dst_metric_locked(&rth->u.dst, RTAX_MTU) ||
         !net_eq(dev_net(rth->u.dst.dev), net) ||
-        rth->rt_genid != atomic_read(&rt_genid))
+        !rt_is_expired(rth))
      continue;

     if (new_mtu < 68 || new_mtu >= old_mtu) {
@@ -1698,7 +1707,7 @@ static int ip_route_input_mc(struct sk_buff *skb, __be32 daddr,
__be32 saddr,
  rth->fl.oif = 0;
  rth->rt_gateway = daddr;
  rth->rt_spec_dst= spec_dst;
- rth->rt_genid = atomic_read(&rt_genid);
+ rth->rt_genid = rt_genid(dev_net(dev));
```

```
  rth->rt_flags = RTCF_MULTICAST;
  rth->rt_type = RTN_MULTICAST;
  if (our) {
@@ -1713,7 +1722,7 @@ static int ip_route_input_mc(struct sk_buff *skb, __be32 daddr,
__be32 saddr,
  RT_CACHE_STAT_INC(in_slow_mc);

  in_dev_put(in_dev);
- hash = rt_hash(daddr, saddr, dev->ifindex, atomic_read(&rt_genid));
+ hash = rt_hash(daddr, saddr, dev->ifindex, rt_genid(dev_net(dev)));
  return rt_intern_hash(hash, rth, &skb->rtable);

 e_nobufs:
@@ -1839,7 +1848,7 @@ static int __mkroute_input(struct sk_buff *skb,

  rth->u.dst.input = ip_forward;
  rth->u.dst.output = ip_output;
- rth->rt_genid = atomic_read(&rt_genid);
+ rth->rt_genid = rt_genid(dev_net(rth->u.dst.dev));

  rt_set_nexthop(rth, res, itag);

@@ -1874,7 +1883,8 @@ static int ip_mkroute_input(struct sk_buff *skb,
   return err;

  /* put it into the cache */
- hash = rt_hash(daddr, saddr, fl->iif, atomic_read(&rt_genid));
+ hash = rt_hash(daddr, saddr, fl->iif,
+         rt_genid(dev_net(rth->u.dst.dev)));
  return rt_intern_hash(hash, rth, &skb->rtable);
 }

@@ -2000,7 +2010,7 @@ local_input:
   goto e_nobufs;

  rth->u.dst.output= ip_rt_bug;
- rth->rt_genid = atomic_read(&rt_genid);
+ rth->rt_genid = rt_genid(net);

  atomic_set(&rth->u.dst.__refcnt, 1);
  rth->u.dst.flags= DST_HOST;
@@ -2030,7 +2040,7 @@ local_input:
   rth->rt_flags  &= ~RTCF_LOCAL;
  }
  rth->rt_type = res.type;
- hash = rt_hash(daddr, saddr, fl.iif, atomic_read(&rt_genid));
+ hash = rt_hash(daddr, saddr, fl.iif, rt_genid(net));
  err = rt_intern_hash(hash, rth, &skb->rtable);
```

```
  goto done;

@@ -2081,7 +2091,7 @@ int ip_route_input(struct sk_buff *skb, __be32 daddr, __be32 saddr,

  net = dev_net(dev);
  tos &= IPTOS_RT_MASK;
- hash = rt_hash(daddr, saddr, iif, atomic_read(&rt_genid));
+ hash = rt_hash(daddr, saddr, iif, rt_genid(net));

  rcu_read_lock();
  for (rth = rcu_dereference(rt_hash_table[hash].chain); rth;
@@ -2093,7 +2103,7 @@ int ip_route_input(struct sk_buff *skb, __be32 daddr, __be32 saddr,
        (rth->fl.fl4_tos ^ tos)) == 0 &&
       rth->fl.mark == skb->mark &&
       net_eq(dev_net(rth->u.dst.dev), net) &&
-      rth->rt_genid == atomic_read(&rt_genid)) {
+      !rt_is_expired(rth)) {
    dst_use(&rth->u.dst, jiffies);
    RT_CACHE_STAT_INC(in_hit);
    rcu_read_unlock();
@@ -2221,7 +2231,7 @@ static int __mkroute_output(struct rtable **result,
  rth->rt_spec_dst= fl->fl4_src;

  rth->u.dst.output=ip_output;
- rth->rt_genid = atomic_read(&rt_genid);
+ rth->rt_genid = rt_genid(dev_net(dev_out));

  RT_CACHE_STAT_INC(out_slow_tot);

@@ -2271,7 +2281,7 @@ static int ip_mkroute_output(struct rtable **rp,
  unsigned hash;
  if (err == 0) {
   hash = rt_hash(oldflp->fl4_dst, oldflp->fl4_src, oldflp->oif,
-         atomic_read(&rt_genid));
+         rt_genid(dev_net(dev_out)));
   err = rt_intern_hash(hash, rth, rp);
  }

@@ -2483,8 +2493,7 @@ int __ip_route_output_key(struct net *net, struct rtable **rp,
  unsigned hash;
  struct rtable *rth;

- hash = rt_hash(flp->fl4_dst, flp->fl4_src, flp->oif,
-        atomic_read(&rt_genid));
+ hash = rt_hash(flp->fl4_dst, flp->fl4_src, flp->oif, rt_genid(net));

  rcu_read_lock_bh();
  for (rth = rcu_dereference(rt_hash_table[hash].chain); rth;
```

```
@@ -2497,7 +2506,7 @@ int __ip_route_output_key(struct net *net, struct rtable **rp,
       !((rth->fl.fl4_tos ^ flp->fl4_tos) &
         (IPTOS_RT_MASK | RTO_ONLINK)) &&
       net_eq(dev_net(rth->u.dst.dev), net) &&
-      rth->rt_genid == atomic_read(&rt_genid)) {
+      !rt_is_expired(rth)) {
    dst_use(&rth->u.dst, jiffies);
    RT_CACHE_STAT_INC(out_hit);
    rcu_read_unlock_bh();
@@ -2528,7 +2537,7 @@ static struct dst_ops ipv4_dst_blackhole_ops = {
 };


-static int ipv4_dst_blackhole(struct rtable **rp, struct flowi *flp)
+static int ipv4_dst_blackhole(struct net *net, struct rtable **rp, struct flowi *flp)
 {
  struct rtable *ort = *rp;
  struct rtable *rt = (struct rtable *)
@@ -2552,7 +2561,7 @@ static int ipv4_dst_blackhole(struct rtable **rp, struct flowi *flp)
   rt->idev = ort->idev;
   if (rt->idev)
    in_dev_hold(rt->idev);
-  rt->rt_genid = atomic_read(&rt_genid);
+  rt->rt_genid = rt_genid(net);
   rt->rt_flags = ort->rt_flags;
   rt->rt_type = ort->rt_type;
   rt->rt_dst = ort->rt_dst;
@@ -2588,7 +2597,7 @@ int ip_route_output_flow(struct net *net, struct rtable **rp, struct flowi
*flp,
   err = __xfrm_lookup((struct dst_entry **)rp, flp, sk,
         flags ? XFRM_LOOKUP_WAIT : 0);
   if (err == -EREMOTE)
-   err = ipv4_dst_blackhole(rp, flp);
+   err = ipv4_dst_blackhole(net, rp, flp);

   return err;
  }
@@ -2807,7 +2816,7 @@ int ip_rt_dump(struct sk_buff *skb,  struct netlink_callback *cb)
        rt = rcu_dereference(rt->u.dst.rt_next), idx++) {
    if (!net_eq(dev_net(rt->u.dst.dev), net) || idx < s_idx)
     continue;
-   if (rt->rt_genid != atomic_read(&rt_genid))
+   if (rt_is_expired(rt))
     continue;
    skb->dst = dst_clone(&rt->u.dst);
    if (rt_fill_info(skb, NETLINK_CB(cb->skb).pid,
@@ -3081,6 +3090,10 @@ static __net_initdata struct pernet_operations sysctl_route_ops = {
```

```
 static __net_init int rt_secret_timer_init(struct net *net)
 {
+ atomic_set(&net->ipv4.rt_genid,
+    (int) ((num_physpages ^ (num_physpages>>8)) ^
+    (jiffies ^ (jiffies >> 7))));
+
   net->ipv4.rt_secret_timer.function = rt_secret_rebuild;
   net->ipv4.rt_secret_timer.data = (unsigned long)net;
   init_timer_deferrable(&net->ipv4.rt_secret_timer);
@@ -3121,9 +3134,6 @@ int __init ip_rt_init(void)
 {
   int rc = 0;

- atomic_set(&rt_genid, (int) ((num_physpages ^ (num_physpages>>8)) ^
-        (jiffies ^ (jiffies >> 7))));
-
 #ifdef CONFIG_NET_CLS_ROUTE
   ip_rt_acct = __alloc_percpu(256 * sizeof(struct ip_rt_acct));
   if (!ip_rt_acct)
--
1.5.3.rc5
```

Subject: [PATCH net-next 9/9] netns: selective flush of rt_cache
Posted by den on Fri, 04 Jul 2008 13:17:12 GMT
View Forum Message <> Reply to Message

dst cache is marked as expired on the per/namespace basis by previous
path. Right now we have to implement selective cache shrinking. This
procedure has been ported from older OpenVz codebase.

Signed-off-by: Denis V. Lunev <den@openvz.org>

```
---
 net/ipv4/route.c |   31 ++++++++++++++++++++++++++++++-
 1 files changed, 30 insertions(+), 1 deletions(-)

diff --git a/net/ipv4/route.c b/net/ipv4/route.c
index 67c3ed7..113cd25 100644
--- a/net/ipv4/route.c
+++ b/net/ipv4/route.c
@@ -699,6 +699,7 @@ static void rt_do_flush(int process_context)
 {
   unsigned int i;
```

```
  struct rtable *rth, *next;
+ struct rtable * tail;

  for (i = 0; i <= rt_hash_mask; i++) {
   if (process_context && need_resched())
@@ -708,11 +709,39 @@ static void rt_do_flush(int process_context)
    continue;

   spin_lock_bh(rt_hash_lock_addr(i));
+#ifdef CONFIG_NET_NS
+  {
+  struct rtable ** prev, * p;
+
+  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_is_expired(tail))
+    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_is_expired(p)) {
+    prev = &p->u.dst.rt_next;
+   } else {
+    *prev = next;
+    rt_free(p);
+   }
+  }
+  }
+#else
   rth = rt_hash_table[i].chain;
   rt_hash_table[i].chain = NULL;
+  tail = NULL;
+#endif
   spin_unlock_bh(rt_hash_lock_addr(i));

-  for (; rth; rth = next) {
+  for (; rth != tail; rth = next) {
    next = rth->u.dst.rt_next;
    rt_free(rth);
   }
--
1.5.3.rc5
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH net-next 0/9] selective (per/namespace) flush of rt_cache
Posted by davem on Sun, 06 Jul 2008 03:55:48 GMT
View Forum Message <> Reply to Message

From: "Denis V. Lunev" <den@openvz.org>
Date: Fri, 04 Jul 2008 17:16:00 +0400

> This series of patches implements selective rt cache flushing to make
> sure that in one namespace we'll not been able to affect the performance
> of other from the user space.

Applied and pushed out to net-next-2.6, thanks.

Although I wish patch 9 didn't have to be so ugly. :-/ Also, is it
really the right thing to do if another namespace's RT cache entries
are in fact chewing up all the slots in a hash chain? I think
the replacement garbage collection algorithm should be namespace
agnostic.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH net-next 3/9] ipv4: remove static flush_delay variable
Posted by ebiederm on Mon, 07 Jul 2008 08:43:06 GMT
View Forum Message <> Reply to Message

"Denis V. Lunev" <den@openvz.org> writes:

> flush delay is used as an external storage for net.ipv4.route.flush sysctl
> entry. It is write-only.
>
> The ctl_table->data for this entry is used once. Fix this case to point
> to the stack to remove global variable. Do this to avoid additional
> variable on struct net in the next patch.
>
> Possible race (as it was before) accessing this local variable is removed
> using flush_mutex.

FYI. You can avoid the locking entirely by defining a local struct ctl_table variable
on the stack.

Eric

Subject: Re: [PATCH net-next 0/9] selective (per/namespace) flush of rt_cache
Posted by ebiederm on Mon, 07 Jul 2008 10:29:23 GMT
View Forum Message <> Reply to Message

David Miller <davem@davemloft.net> writes:

> From: "Denis V. Lunev" <den@openvz.org>
> Date: Fri, 04 Jul 2008 17:16:00 +0400
>
>> This series of patches implements selective rt cache flushing to make
>> sure that in one namespace we'll not been able to affect the performance
>> of other from the user space.
>
> Applied and pushed out to net-next-2.6, thanks.
>
> Although I wish patch 9 didn't have to be so ugly. :-/ Also, is it
> really the right thing to do if another namespace's RT cache entries
> are in fact chewing up all the slots in a hash chain? I think
> the replacement garbage collection algorithm should be namespace
> agnostic.

My VM experience agrees. The requested flushes from inside the namespace
and for namespace exit really should just flush the routes for that namespace.
However for general route caching and expiry I don't see the point of making
the code per namespace, and my gut feel is that is likely to reduce average
case performance at the cost of better isolation between namespaces.

Denis did I read the patches right and you are making all route cache flushes
per namespace? Including the periodic expiry?

Eric

Subject: Re: [PATCH net-next 0/9] selective (per/namespace) flush of rt_cache
Posted by den on Mon, 07 Jul 2008 10:39:41 GMT
View Forum Message <> Reply to Message

On Mon, 2008-07-07 at 03:29 -0700, Eric W. Biederman wrote:
> David Miller <davem@davemloft.net> writes:
>
> > From: "Denis V. Lunev" <den@openvz.org>
> > Date: Fri, 04 Jul 2008 17:16:00 +0400
> >
> >> This series of patches implements selective rt cache flushing to make
> >> sure that in one namespace we'll not been able to affect the performance
> >> of other from the user space.
> >
> > Applied and pushed out to net-next-2.6, thanks.
> >
> > Although I wish patch 9 didn't have to be so ugly. :-/ Also, is it
> > really the right thing to do if another namespace's RT cache entries
> > are in fact chewing up all the slots in a hash chain? I think
> > the replacement garbage collection algorithm should be namespace
> > agnostic.
>
> My VM experience agrees. The requested flushes from inside the namespace
> and for namespace exit really should just flush the routes for that namespace.
> However for general route caching and expiry I don't see the point of making
> the code per namespace, and my gut feel is that is likely to reduce average
> case performance at the cost of better isolation between namespaces.
>
> Denis did I read the patches right and you are making all route cache flushes
> per namespace? Including the periodic expiry?
yes. The reason for the latter is that we can't pin namespace list in
the timer, so it is better to have a separate timer for each namespace.
The similar approach has been discussed for IPv6 flushing if my memory
does not fail me.

Though, generic rt cache garbage collecting is left untouched and from
my point of view it should be left as it is. At least we use this
approach in OpenVz.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH net-next 0/9] selective (per/namespace) flush of rt_cache
Posted by den on Mon, 07 Jul 2008 10:51:09 GMT
View Forum Message <> Reply to Message

On Sat, 2008-07-05 at 20:55 -0700, David Miller wrote:
> From: "Denis V. Lunev" <den@openvz.org>
> Date: Fri, 04 Jul 2008 17:16:00 +0400
>
> > This series of patches implements selective rt cache flushing to make
> > sure that in one namespace we'll not been able to affect the performance
> > of other from the user space.
>
> Applied and pushed out to net-next-2.6, thanks.
>
> Although I wish patch 9 didn't have to be so ugly. :-/ Also, is it
> really the right thing to do if another namespace's RT cache entries
> are in fact chewing up all the slots in a hash chain? I think
> the replacement garbage collection algorithm should be namespace
> agnostic.

hmmm... it looks like a name rt_is_expired is not good enough for the
case. May be rt_is_flushed should be better?

Subject: Re: [PATCH net-next 3/9] ipv4: remove static flush_delay variable
Posted by davem on Tue, 08 Jul 2008 10:05:59 GMT
View Forum Message <> Reply to Message

From: "Denis V. Lunev" <den@openvz.org>
Date: Mon, 07 Jul 2008 15:26:14 +0400

> On Mon, 2008-07-07 at 01:43 -0700, Eric W. Biederman wrote:
> > "Denis V. Lunev" <den@openvz.org> writes:
> >
> > > flush delay is used as an external storage for net.ipv4.route.flush sysctl
> > > entry. It is write-only.
> > >
> > > The ctl_table->data for this entry is used once. Fix this case to point
> > > to the stack to remove global variable. Do this to avoid additional
> > > variable on struct net in the next patch.
> > >
> > > Possible race (as it was before) accessing this local variable is removed
> > > using flush_mutex.
> >
> > FYI. You can avoid the locking entirely by defining a local struct ctl_table variable
> > on the stack.
>

> Dave, could you consider the patch attached?

Applied to net-next-2.6, thanks!

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers