Subject: [PATCH 1/5] net: Modify all rtnetlink methods to only work in the initial namespace
Posted by ebiederm on Sat, 29 Sep 2007 01:00:37 GMT
View Forum Message <> Reply to Message

Before I can enable rtnetlink to work in all network namespaces
I need to be certain that something won't break. So this
patch deliberately disables all of the rtnletlink methods in everything
except the initial network namespace. After the methods have been
audited this extra check can be disabled.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
net/bridge/br_netlink.c | 9 +++++++++
net/core/fib_rules.c | 11 +++++++++++
net/core/neighbour.c | 18 ++++++++++++++++++
net/core/rtnetlink.c | 19 +++++++++++++++++++
net/decnet/dn_dev.c | 12 ++++++++++++
net/decnet/dn_fib.c | 8 ++++++++
net/decnet/dn_route.c | 8 ++++++++
net/decnet/dn_table.c | 4 ++++
net/ipv4/devinet.c | 12 ++++++++++++
net/ipv4/fib_frontend.c | 12 ++++++++++++
net/ipv4/route.c | 4 ++++
net/ipv6/addrconf.c | 31 +++++++++++++++++++++++++++++++
net/ipv6/ip6_fib.c | 4 ++++
net/ipv6/route.c | 12 ++++++++++++
net/sched/act_api.c | 10 ++++++++++
net/sched/cls_api.c | 10 ++++++++++
net/sched/sch_api.c | 21 +++++++++++++++++++++
17 files changed, 205 insertions(+), 0 deletions(-)

```
diff --git a/net/bridge/br_netlink.c b/net/bridge/br_netlink.c
index 53ab8e0..a4ffa2b 100644
--- a/net/bridge/br_netlink.c
+++ b/net/bridge/br_netlink.c
@@ -13,6 +13,7 @@
 #include <linux/kernel.h>
 #include <net/rtnetlink.h>
 #include <net/net_namespace.h>
+#include <net/sock.h>
 #include "br_private.h"

 static inline size_t br_nlmsg_size(void)
@@ -107,9 +108,13 @@ errout:
  */
 static int br_dump_ifinfo(struct sk_buff *skb, struct netlink_callback *cb)
 {
```

```
+ struct net *net = skb->sk->sk_net;
  struct net_device *dev;
  int idx;

+ if (net != &init_net)
+  return 0;
+
  idx = 0;
  for_each_netdev(&init_net, dev) {
   /* not a bridge port */
@@ -135,12 +140,16 @@ skip:
  */
 static int br_rtm_setlink(struct sk_buff *skb,  struct nlmsghdr *nlh, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct ifinfomsg *ifm;
  struct nlattr *protinfo;
  struct net_device *dev;
  struct net_bridge_port *p;
  u8 new_state;

+ if (net != &init_net)
+  return -EINVAL;
+
  if (nlmsg_len(nlh) < sizeof(*ifm))
   return -EINVAL;

diff --git a/net/core/fib_rules.c b/net/core/fib_rules.c
index 13de6f5..357bfa0 100644
--- a/net/core/fib_rules.c
+++ b/net/core/fib_rules.c
@@ -206,6 +206,9 @@ static int fib_nl_newrule(struct sk_buff *skb, struct nlmsghdr* nlh, void
*arg)
  struct nlattr *tb[FRA_MAX+1];
  int err = -EINVAL, unresolved = 0;

+ if (net != &init_net)
+  return -EINVAL;
+
  if (nlh->nlmsg_len < nlmsg_msg_size(sizeof(*frh)))
   goto errout;

@@ -336,12 +339,16 @@ errout:

 static int fib_nl_delrule(struct sk_buff *skb, struct nlmsghdr* nlh, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct fib_rule_hdr *frh = nlmsg_data(nlh);
```

```
  struct fib_rules_ops *ops = NULL;
  struct fib_rule *rule, *tmp;
  struct nlattr *tb[FRA_MAX+1];
  int err = -EINVAL;

+ if (net != &init_net)
+  return -EINVAL;
+
  if (nlh->nlmsg_len < nlmsg_msg_size(sizeof(*frh)))
   goto errout;

@@ -517,9 +524,13 @@ skip:

 static int fib_nl_dumprule(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  struct fib_rules_ops *ops;
  int idx = 0, family;

+ if (net != &init_net)
+  return -EINVAL;
+
  family = rtnl_msg_family(cb->nlh);
  if (family != AF_UNSPEC) {
   /* Protocol specific dump request */
diff --git a/net/core/neighbour.c b/net/core/neighbour.c
index c52df85..27001db 100644
--- a/net/core/neighbour.c
+++ b/net/core/neighbour.c
@@ -1448,6 +1448,9 @@ static int neigh_delete(struct sk_buff *skb, struct nlmsghdr *nlh, void
*arg)
  struct net_device *dev = NULL;
  int err = -EINVAL;

+ if (net != &init_net)
+  return -EINVAL;
+
  if (nlmsg_len(nlh) < sizeof(*ndm))
   goto out;

@@ -1514,6 +1517,9 @@ static int neigh_add(struct sk_buff *skb, struct nlmsghdr *nlh, void
*arg)
  struct net_device *dev = NULL;
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
```

```
  err = nlmsg_parse(nlh, sizeof(*ndm), tb, NDA_MAX, NULL);
  if (err < 0)
   goto out;
@@ -1788,11 +1794,15 @@ static const struct nla_policy nl_ntbl_parm_policy[NDTPA_MAX+1]
= {

 static int neightbl_set(struct sk_buff *skb, struct nlmsghdr *nlh, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct neigh_table *tbl;
  struct ndtmsg *ndtmsg;
  struct nlattr *tb[NDTA_MAX+1];
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
  err = nlmsg_parse(nlh, sizeof(*ndtmsg), tb, NDTA_MAX,
      nl_neightbl_policy);
  if (err < 0)
@@ -1912,11 +1922,15 @@ errout:

 static int neightbl_dump_info(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  int family, tidx, nidx = 0;
  int tbl_skip = cb->args[0];
  int neigh_skip = cb->args[1];
  struct neigh_table *tbl;

+ if (net != &init_net)
+  return 0;
+
  family = ((struct rtgenmsg *) nlmsg_data(cb->nlh))->rtgen_family;

  read_lock(&neigh_tbl_lock);
@@ -2041,9 +2055,13 @@ out:

 static int neigh_dump_info(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  struct neigh_table *tbl;
  int t, family, s_t;

+ if (net != &init_net)
+  return 0;
+
  read_lock(&neigh_tbl_lock);
```

```
  family = ((struct rtgenmsg *) nlmsg_data(cb->nlh))->rtgen_family;
  s_t = cb->args[0];
diff --git a/net/core/rtnetlink.c b/net/core/rtnetlink.c
index 8bc68e6..56fc4f9 100644
--- a/net/core/rtnetlink.c
+++ b/net/core/rtnetlink.c
@@ -707,6 +707,9 @@ static int rtnl_dump_ifinfo(struct sk_buff *skb, struct netlink_callback *cb)
  int s_idx = cb->args[0];
  struct net_device *dev;

+ if (net != &init_net)
+  return 0;
+
  idx = 0;
  for_each_netdev(net, dev) {
   if (idx < s_idx)
@@ -909,6 +912,9 @@ static int rtnl_setlink(struct sk_buff *skb, struct nlmsghdr *nlh, void *arg)
  struct nlattr *tb[IFLA_MAX+1];
  char ifname[IFNAMSIZ];

+ if (net != &init_net)
+  return -EINVAL;
+
  err = nlmsg_parse(nlh, sizeof(*ifm), tb, IFLA_MAX, ifla_policy);
  if (err < 0)
   goto errout;
@@ -957,6 +963,9 @@ static int rtnl_dellink(struct sk_buff *skb, struct nlmsghdr *nlh, void *arg)
  struct nlattr *tb[IFLA_MAX+1];
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
  err = nlmsg_parse(nlh, sizeof(*ifm), tb, IFLA_MAX, ifla_policy);
  if (err < 0)
   return err;
@@ -1038,6 +1047,9 @@ static int rtnl_newlink(struct sk_buff *skb, struct nlmsghdr *nlh, void
*arg)
  struct nlattr *linkinfo[IFLA_INFO_MAX+1];
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
 #ifdef CONFIG_KMOD
 replay:
 #endif
@@ -1164,6 +1176,9 @@ static int rtnl_getlink(struct sk_buff *skb, struct nlmsghdr* nlh, void
```

```
*arg)
  struct sk_buff *nskb;
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
  err = nlmsg_parse(nlh, sizeof(*ifm), tb, IFLA_MAX, ifla_policy);
  if (err < 0)
   return err;
@@ -1199,9 +1214,13 @@ errout:

 static int rtnl_dump_all(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  int idx;
  int s_idx = cb->family;

+ if (net != &init_net)
+  return 0;
+
  if (s_idx == 0)
   s_idx = 1;
  for (idx=1; idx<NPROTO; idx++) {
diff --git a/net/decnet/dn_dev.c b/net/decnet/dn_dev.c
index 26130af..6e82f7a 100644
--- a/net/decnet/dn_dev.c
+++ b/net/decnet/dn_dev.c
@@ -647,12 +647,16 @@ static const struct nla_policy dn_ifa_policy[IFA_MAX+1] = {

 static int dn_nl_deladdr(struct sk_buff *skb, struct nlmsghdr *nlh, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct nlattr *tb[IFA_MAX+1];
  struct dn_dev *dn_db;
  struct ifaddrmsg *ifm;
  struct dn_ifaddr *ifa, **ifap;
  int err = -EADDRNOTAVAIL;

+ if (net != &init_net)
+  goto errout;
+
  err = nlmsg_parse(nlh, sizeof(*ifm), tb, IFA_MAX, dn_ifa_policy);
  if (err < 0)
   goto errout;
@@ -679,6 +683,7 @@ errout:

 static int dn_nl_newaddr(struct sk_buff *skb, struct nlmsghdr *nlh, void *arg)
```

```
 {
+ struct net *net = skb->sk->sk_net;
  struct nlattr *tb[IFA_MAX+1];
  struct net_device *dev;
  struct dn_dev *dn_db;
@@ -686,6 +691,9 @@ static int dn_nl_newaddr(struct sk_buff *skb, struct nlmsghdr *nlh, void
*arg)
  struct dn_ifaddr *ifa;
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
  err = nlmsg_parse(nlh, sizeof(*ifm), tb, IFA_MAX, dn_ifa_policy);
  if (err < 0)
   return err;
@@ -791,11 +799,15 @@ errout:

 static int dn_nl_dump_ifaddr(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  int idx, dn_idx = 0, skip_ndevs, skip_naddr;
  struct net_device *dev;
  struct dn_dev *dn_db;
  struct dn_ifaddr *ifa;

+ if (net != &init_net)
+  return 0;
+
  skip_ndevs = cb->args[0];
  skip_naddr = cb->args[1];

diff --git a/net/decnet/dn_fib.c b/net/decnet/dn_fib.c
index 3760a20..5413e1b 100644
--- a/net/decnet/dn_fib.c
+++ b/net/decnet/dn_fib.c
@@ -506,10 +506,14 @@ static int dn_fib_check_attr(struct rtmsg *r, struct rtattr **rta)

 static int dn_fib_rtm_delroute(struct sk_buff *skb, struct nlmsghdr *nlh, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct dn_fib_table *tb;
  struct rtattr **rta = arg;
  struct rtmsg *r = NLMSG_DATA(nlh);

+ if (net != &init_net)
+  return -EINVAL;
+
```

```
  if (dn_fib_check_attr(r, rta))
   return -EINVAL;

@@ -522,10 +526,14 @@ static int dn_fib_rtm_delroute(struct sk_buff *skb, struct nlmsghdr *nlh,
void *

 static int dn_fib_rtm_newroute(struct sk_buff *skb, struct nlmsghdr *nlh, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct dn_fib_table *tb;
  struct rtattr **rta = arg;
  struct rtmsg *r = NLMSG_DATA(nlh);

+ if (net != &init_net)
+  return -EINVAL;
+
  if (dn_fib_check_attr(r, rta))
   return -EINVAL;

diff --git a/net/decnet/dn_route.c b/net/decnet/dn_route.c
index b7ebf99..a16374b 100644
--- a/net/decnet/dn_route.c
+++ b/net/decnet/dn_route.c
@@ -1530,6 +1530,7 @@ rtattr_failure:
  */
 static int dn_cache_getroute(struct sk_buff *in_skb, struct nlmsghdr *nlh, void *arg)
 {
+ struct net *net = in_skb->sk->sk_net;
  struct rtattr **rta = arg;
  struct rtmsg *rtm = NLMSG_DATA(nlh);
  struct dn_route *rt = NULL;
@@ -1538,6 +1539,9 @@ static int dn_cache_getroute(struct sk_buff *in_skb, struct nlmsghdr
*nlh, void
  struct sk_buff *skb;
  struct flowi fl;

+ if (net != &init_net)
+  return -EINVAL;
+
  memset(&fl, 0, sizeof(fl));
  fl.proto = DNPROTO_NSP;

@@ -1615,10 +1619,14 @@ out_free:
  */
 int dn_cache_dump(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  struct dn_route *rt;
```

```
  int h, s_h;
  int idx, s_idx;

+ if (net != &init_net)
+  return 0;
+
  if (NLMSG_PAYLOAD(cb->nlh, 0) < sizeof(struct rtmsg))
   return -EINVAL;
  if (!(((struct rtmsg *)NLMSG_DATA(cb->nlh))->rtm_flags&RTM_F_CLONED))
diff --git a/net/decnet/dn_table.c b/net/decnet/dn_table.c
index fda0772..a3bdb8d 100644
--- a/net/decnet/dn_table.c
+++ b/net/decnet/dn_table.c
@@ -463,12 +463,16 @@ static int dn_fib_table_dump(struct dn_fib_table *tb, struct sk_buff
*skb,

 int dn_fib_dump(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  unsigned int h, s_h;
  unsigned int e = 0, s_e;
  struct dn_fib_table *tb;
  struct hlist_node *node;
  int dumped = 0;

+ if (net != &init_net)
+  return 0;
+
  if (NLMSG_PAYLOAD(cb->nlh, 0) >= sizeof(struct rtmsg) &&
   ((struct rtmsg *)NLMSG_DATA(cb->nlh))->rtm_flags&RTM_F_CLONED)
    return dn_cache_dump(skb, cb);
diff --git a/net/ipv4/devinet.c b/net/ipv4/devinet.c
index 55d199e..0ff2ef6 100644
--- a/net/ipv4/devinet.c
+++ b/net/ipv4/devinet.c
@@ -441,6 +441,7 @@ struct in_ifaddr *inet_ifa_byprefix(struct in_device *in_dev, __be32 prefix,

 static int inet_rtm_deladdr(struct sk_buff *skb, struct nlmsghdr *nlh, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct nlattr *tb[IFA_MAX+1];
  struct in_device *in_dev;
  struct ifaddrmsg *ifm;
@@ -449,6 +450,9 @@ static int inet_rtm_deladdr(struct sk_buff *skb, struct nlmsghdr *nlh, void
*arg

  ASSERT_RTNL();
```

```
+ if (net != &init_net)
+  return -EINVAL;
+
  err = nlmsg_parse(nlh, sizeof(*ifm), tb, IFA_MAX, ifa_ipv4_policy);
  if (err < 0)
   goto errout;
@@ -561,10 +565,14 @@ errout:

 static int inet_rtm_newaddr(struct sk_buff *skb, struct nlmsghdr *nlh, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct in_ifaddr *ifa;

  ASSERT_RTNL();

+ if (net != &init_net)
+  return -EINVAL;
+
  ifa = rtm_to_ifaddr(nlh);
  if (IS_ERR(ifa))
   return PTR_ERR(ifa);
@@ -1175,12 +1183,16 @@ nla_put_failure:

 static int inet_dump_ifaddr(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  int idx, ip_idx;
  struct net_device *dev;
  struct in_device *in_dev;
  struct in_ifaddr *ifa;
  int s_ip_idx, s_idx = cb->args[0];

+ if (net != &init_net)
+  return 0;
+
  s_ip_idx = ip_idx = cb->args[1];
  idx = 0;
  for_each_netdev(&init_net, dev) {
diff --git a/net/ipv4/fib_frontend.c b/net/ipv4/fib_frontend.c
index f823ca3..84d688a 100644
--- a/net/ipv4/fib_frontend.c
+++ b/net/ipv4/fib_frontend.c
@@ -527,10 +527,14 @@ errout:

 static int inet_rtm_delroute(struct sk_buff *skb, struct nlmsghdr* nlh, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct fib_config cfg;
```

```
  struct fib_table *tb;
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
  err = rtm_to_fib_config(skb, nlh, &cfg);
  if (err < 0)
   goto errout;
@@ -548,10 +552,14 @@ errout:

 static int inet_rtm_newroute(struct sk_buff *skb, struct nlmsghdr* nlh, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct fib_config cfg;
  struct fib_table *tb;
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
  err = rtm_to_fib_config(skb, nlh, &cfg);
  if (err < 0)
   goto errout;
@@ -569,12 +577,16 @@ errout:

 static int inet_dump_fib(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  unsigned int h, s_h;
  unsigned int e = 0, s_e;
  struct fib_table *tb;
  struct hlist_node *node;
  int dumped = 0;

+ if (net != &init_net)
+  return 0;
+
  if (nlmsg_len(cb->nlh) >= sizeof(struct rtmsg) &&
      ((struct rtmsg *) nlmsg_data(cb->nlh))->rtm_flags & RTM_F_CLONED)
   return ip_rt_dump(skb, cb);
diff --git a/net/ipv4/route.c b/net/ipv4/route.c
index c5a06e6..a538a30 100644
--- a/net/ipv4/route.c
+++ b/net/ipv4/route.c
@@ -2552,6 +2552,7 @@ nla_put_failure:

 static int inet_rtm_getroute(struct sk_buff *in_skb, struct nlmsghdr* nlh, void *arg)
```

```
 {
+ struct net *net = in_skb->sk->sk_net;
  struct rtmsg *rtm;
  struct nlattr *tb[RTA_MAX+1];
  struct rtable *rt = NULL;
@@ -2561,6 +2562,9 @@ static int inet_rtm_getroute(struct sk_buff *in_skb, struct nlmsghdr*
nlh, void
  int err;
  struct sk_buff *skb;

+ if (net != &init_net)
+  return -EINVAL;
+
  err = nlmsg_parse(nlh, sizeof(*rtm), tb, RTA_MAX, rtm_ipv4_policy);
  if (err < 0)
   goto errout;
diff --git a/net/ipv6/addrconf.c b/net/ipv6/addrconf.c
index 6d5c3c2..bdc183e 100644
--- a/net/ipv6/addrconf.c
+++ b/net/ipv6/addrconf.c
@@ -3003,11 +3003,15 @@ static const struct nla_policy ifa_ipv6_policy[IFA_MAX+1] = {
 static int
 inet6_rtm_deladdr(struct sk_buff *skb, struct nlmsghdr *nlh, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct ifaddrmsg *ifm;
  struct nlattr *tb[IFA_MAX+1];
  struct in6_addr *pfx;
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
  err = nlmsg_parse(nlh, sizeof(*ifm), tb, IFA_MAX, ifa_ipv6_policy);
  if (err < 0)
   return err;
@@ -3060,6 +3064,7 @@ static int inet6_addr_modify(struct inet6_ifaddr *ifp, u8 ifa_flags,
 static int
 inet6_rtm_newaddr(struct sk_buff *skb, struct nlmsghdr *nlh, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct ifaddrmsg *ifm;
  struct nlattr *tb[IFA_MAX+1];
  struct in6_addr *pfx;
@@ -3069,6 +3074,9 @@ inet6_rtm_newaddr(struct sk_buff *skb, struct nlmsghdr *nlh, void
*arg)
  u8 ifa_flags;
  int err;
```

```
+ if (net != &init_net)
+  return -EINVAL;
+
  err = nlmsg_parse(nlh, sizeof(*ifm), tb, IFA_MAX, ifa_ipv6_policy);
  if (err < 0)
   return err;
@@ -3342,26 +3350,42 @@ done:

 static int inet6_dump_ifaddr(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  enum addr_type_t type = UNICAST_ADDR;
+
+ if (net != &init_net)
+  return 0;
+
  return inet6_dump_addr(skb, cb, type);
 }

 static int inet6_dump_ifmcaddr(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  enum addr_type_t type = MULTICAST_ADDR;
+
+ if (net != &init_net)
+  return 0;
+
  return inet6_dump_addr(skb, cb, type);
 }


 static int inet6_dump_ifacaddr(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  enum addr_type_t type = ANYCAST_ADDR;
+
+ if (net != &init_net)
+  return 0;
+
  return inet6_dump_addr(skb, cb, type);
 }

 static int inet6_rtm_getaddr(struct sk_buff *in_skb, struct nlmsghdr* nlh,
        void *arg)
 {
+ struct net *net = in_skb->sk->sk_net;
  struct ifaddrmsg *ifm;
```

```
  struct nlattr *tb[IFA_MAX+1];
  struct in6_addr *addr = NULL;
@@ -3370,6 +3394,9 @@ static int inet6_rtm_getaddr(struct sk_buff *in_skb, struct nlmsghdr*
nlh,
  struct sk_buff *skb;
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
  err = nlmsg_parse(nlh, sizeof(*ifm), tb, IFA_MAX, ifa_ipv6_policy);
  if (err < 0)
   goto errout;
@@ -3587,11 +3614,15 @@ nla_put_failure:

 static int inet6_dump_ifinfo(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  int idx, err;
  int s_idx = cb->args[0];
  struct net_device *dev;
  struct inet6_dev *idev;

+ if (net != &init_net)
+  return 0;
+
  read_lock(&dev_base_lock);
  idx = 0;
  for_each_netdev(&init_net, dev) {
diff --git a/net/ipv6/ip6_fib.c b/net/ipv6/ip6_fib.c
index 6a612a7..a731812 100644
--- a/net/ipv6/ip6_fib.c
+++ b/net/ipv6/ip6_fib.c
@@ -361,6 +361,7 @@ end:

 static int inet6_dump_fib(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  unsigned int h, s_h;
  unsigned int e = 0, s_e;
  struct rt6_rtnl_dump_arg arg;
@@ -369,6 +370,9 @@ static int inet6_dump_fib(struct sk_buff *skb, struct netlink_callback *cb)
  struct hlist_node *node;
  int res = 0;

+ if (net != &init_net)
+  return 0;
+
```

```
  s_h = cb->args[0];
  s_e = cb->args[1];

diff --git a/net/ipv6/route.c b/net/ipv6/route.c
index a7db84c..f0cf11c 100644
--- a/net/ipv6/route.c
+++ b/net/ipv6/route.c
@@ -2080,9 +2080,13 @@ errout:

 static int inet6_rtm_delroute(struct sk_buff *skb, struct nlmsghdr* nlh, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct fib6_config cfg;
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
  err = rtm_to_fib6_config(skb, nlh, &cfg);
  if (err < 0)
   return err;
@@ -2092,9 +2096,13 @@ static int inet6_rtm_delroute(struct sk_buff *skb, struct nlmsghdr* nlh,
void *a

 static int inet6_rtm_newroute(struct sk_buff *skb, struct nlmsghdr* nlh, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct fib6_config cfg;
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
  err = rtm_to_fib6_config(skb, nlh, &cfg);
  if (err < 0)
   return err;
@@ -2229,6 +2237,7 @@ int rt6_dump_route(struct rt6_info *rt, void *p_arg)

 static int inet6_rtm_getroute(struct sk_buff *in_skb, struct nlmsghdr* nlh, void *arg)
 {
+ struct net *net = in_skb->sk->sk_net;
  struct nlattr *tb[RTA_MAX+1];
  struct rt6_info *rt;
  struct sk_buff *skb;
@@ -2236,6 +2245,9 @@ static int inet6_rtm_getroute(struct sk_buff *in_skb, struct nlmsghdr*
nlh, void
  struct flowi fl;
  int err, iif = 0;
```

```
+ if (net != &init_net)
+  return -EINVAL;
+
  err = nlmsg_parse(nlh, sizeof(*rtm), tb, RTA_MAX, rtm_ipv6_policy);
  if (err < 0)
   goto errout;
diff --git a/net/sched/act_api.c b/net/sched/act_api.c
index 72cdb0f..8528291 100644
--- a/net/sched/act_api.c
+++ b/net/sched/act_api.c
@@ -18,6 +18,8 @@
 #include <linux/skbuff.h>
 #include <linux/init.h>
 #include <linux/kmod.h>
+#include <net/net_namespace.h>
+#include <net/sock.h>
 #include <net/sch_generic.h>
 #include <net/act_api.h>
 #include <net/netlink.h>
@@ -924,10 +926,14 @@ done:

 static int tc_ctl_action(struct sk_buff *skb, struct nlmsghdr *n, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct rtattr **tca = arg;
  u32 pid = skb ? NETLINK_CB(skb).pid : 0;
  int ret = 0, ovr = 0;

+ if (net != &init_net)
+  return -EINVAL;
+
  if (tca[TCA_ACT_TAB-1] == NULL) {
   printk("tc_ctl_action: received NO action attribs\n");
   return -EINVAL;
@@ -997,6 +1003,7 @@ find_dump_kind(struct nlmsghdr *n)
 static int
 tc_dump_action(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  struct nlmsghdr *nlh;
  unsigned char *b = skb_tail_pointer(skb);
  struct rtattr *x;
@@ -1006,6 +1013,9 @@ tc_dump_action(struct sk_buff *skb, struct netlink_callback *cb)
  struct tcamsg *t = (struct tcamsg *) NLMSG_DATA(cb->nlh);
  struct rtattr *kind = find_dump_kind(cb->nlh);

+ if (net != &init_net)
```

```
+  return 0;
+
  if (kind == NULL) {
   printk("tc_dump_action: action bad kind\n");
   return 0;
diff --git a/net/sched/cls_api.c b/net/sched/cls_api.c
index 0365797..2754e50 100644
--- a/net/sched/cls_api.c
+++ b/net/sched/cls_api.c
@@ -23,6 +23,8 @@
 #include <linux/init.h>
 #include <linux/kmod.h>
 #include <linux/netlink.h>
+#include <net/net_namespace.h>
+#include <net/sock.h>
 #include <net/netlink.h>
 #include <net/pkt_sched.h>
 #include <net/pkt_cls.h>
@@ -119,6 +121,7 @@ static __inline__ u32 tcf_auto_prio(struct tcf_proto *tp)

 static int tc_ctl_tfilter(struct sk_buff *skb, struct nlmsghdr *n, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct rtattr **tca;
  struct tcmsg *t;
  u32 protocol;
@@ -135,6 +138,9 @@ static int tc_ctl_tfilter(struct sk_buff *skb, struct nlmsghdr *n, void *arg)
  unsigned long fh;
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
 replay:
  tca = arg;
  t = NLMSG_DATA(n);
@@ -375,6 +381,7 @@ static int tcf_node_dump(struct tcf_proto *tp, unsigned long n, struct
tcf_walke

 static int tc_dump_tfilter(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  int t;
  int s_t;
  struct net_device *dev;
@@ -385,6 +392,9 @@ static int tc_dump_tfilter(struct sk_buff *skb, struct netlink_callback *cb)
  struct Qdisc_class_ops *cops;
  struct tcf_dump_args arg;
```

```
+ if (net != &init_net)
+  return 0;
+
  if (cb->nlh->nlmsg_len < NLMSG_LENGTH(sizeof(*tcm)))
   return skb->len;
  if ((dev = dev_get_by_index(&init_net, tcm->tcm_ifindex)) == NULL)
diff --git a/net/sched/sch_api.c b/net/sched/sch_api.c
index 39d3278..95eb0a8 100644
--- a/net/sched/sch_api.c
+++ b/net/sched/sch_api.c
@@ -29,6 +29,7 @@
 #include <linux/hrtimer.h>

 #include <net/net_namespace.h>
+#include <net/sock.h>
 #include <net/netlink.h>
 #include <net/pkt_sched.h>

@@ -599,6 +600,7 @@ check_loop_fn(struct Qdisc *q, unsigned long cl, struct qdisc_walker *w)

 static int tc_get_qdisc(struct sk_buff *skb, struct nlmsghdr *n, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct tcmsg *tcm = NLMSG_DATA(n);
  struct rtattr **tca = arg;
  struct net_device *dev;
@@ -607,6 +609,9 @@ static int tc_get_qdisc(struct sk_buff *skb, struct nlmsghdr *n, void *arg)
  struct Qdisc *p = NULL;
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
  if ((dev = __dev_get_by_index(&init_net, tcm->tcm_ifindex)) == NULL)
   return -ENODEV;

@@ -660,6 +665,7 @@ static int tc_get_qdisc(struct sk_buff *skb, struct nlmsghdr *n, void *arg)

 static int tc_modify_qdisc(struct sk_buff *skb, struct nlmsghdr *n, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct tcmsg *tcm;
  struct rtattr **tca;
  struct net_device *dev;
@@ -667,6 +673,9 @@ static int tc_modify_qdisc(struct sk_buff *skb, struct nlmsghdr *n, void
*arg)
  struct Qdisc *q, *p;
```

```
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
 replay:
  /* Reinit, just in case something touches this. */
  tcm = NLMSG_DATA(n);
@@ -872,11 +881,15 @@ err_out:

 static int tc_dump_qdisc(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
  int idx, q_idx;
  int s_idx, s_q_idx;
  struct net_device *dev;
  struct Qdisc *q;

+ if (net != &init_net)
+  return 0;
+
  s_idx = cb->args[0];
  s_q_idx = q_idx = cb->args[1];
  read_lock(&dev_base_lock);
@@ -920,6 +933,7 @@ done:

 static int tc_ctl_tclass(struct sk_buff *skb, struct nlmsghdr *n, void *arg)
 {
+ struct net *net = skb->sk->sk_net;
  struct tcmsg *tcm = NLMSG_DATA(n);
  struct rtattr **tca = arg;
  struct net_device *dev;
@@ -932,6 +946,9 @@ static int tc_ctl_tclass(struct sk_buff *skb, struct nlmsghdr *n, void *arg)
  u32 qid = TC_H_MAJ(clid);
  int err;

+ if (net != &init_net)
+  return -EINVAL;
+
  if ((dev = __dev_get_by_index(&init_net, tcm->tcm_ifindex)) == NULL)
   return -ENODEV;

@@ -1106,6 +1123,7 @@ static int qdisc_class_dump(struct Qdisc *q, unsigned long cl, struct
qdisc_walk

 static int tc_dump_tclass(struct sk_buff *skb, struct netlink_callback *cb)
 {
+ struct net *net = skb->sk->sk_net;
```

```
 int t;
 int s_t;
 struct net_device *dev;
@@ -1113,6 +1131,9 @@ static int tc_dump_tclass(struct sk_buff *skb, struct netlink_callback *cb)
 struct tcmsg *tcm = (struct tcmsg*)NLMSG_DATA(cb->nlh);
 struct qdisc_dump_args arg;

+ if (net != &init_net)
+  return 0;
+
 if (cb->nlh->nlmsg_len < NLMSG_LENGTH(sizeof(*tcm)))
  return 0;
 if ((dev = dev_get_by_index(&init_net, tcm->tcm_ifindex)) == NULL)
--
1.5.3.rc6.17.g1911
```

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

---

Subject: [PATCH 2/5] net: Make rtnetlink infrastructure network namespace aware
Posted by ebiederm on Sat, 29 Sep 2007 01:02:11 GMT

View Forum Message <> Reply to Message

After this patch none of the netlink callback support anything
except the initial network namespace but the rtnetlink infrastructure
now handles multiple network namespaces.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
---
 include/linux/rtnetlink.h   |    8 ++--
 include/net/net_namespace.h |    3 +
 net/bridge/br_netlink.c     |    4 +-
 net/core/fib_rules.c        |    4 +-
 net/core/neighbour.c        |    4 +-
 net/core/rtnetlink.c        |   96 +++++++++++++++++++++++++++++++++------------
 net/decnet/dn_dev.c         |    4 +-
 net/decnet/dn_route.c       |    2 +-
 net/decnet/dn_table.c       |    4 +-
 net/ipv4/devinet.c          |    4 +-
 net/ipv4/fib_semantics.c    |    4 +-
 net/ipv4/ipmr.c             |    4 +-
 net/ipv4/route.c            |    2 +-
 net/ipv6/addrconf.c         |   14 +++---
 net/ipv6/route.c            |    6 +-
```

```
 net/sched/act_api.c         |    8 ++--
 net/sched/cls_api.c         |    2 +-
 net/sched/sch_api.c         |    4 +-
 net/wireless/wext.c         |    5 ++-
 19 files changed, 129 insertions(+), 53 deletions(-)

diff --git a/include/linux/rtnetlink.h b/include/linux/rtnetlink.h
index 9c21e45..518247e 100644
--- a/include/linux/rtnetlink.h
+++ b/include/linux/rtnetlink.h
@@ -582,11 +582,11 @@ extern int __rtattr_parse_nested_compat(struct rtattr *tb[], int maxattr,
 ({ data = RTA_PAYLOAD(rta) >= len ? RTA_DATA(rta) : NULL; \
   __rtattr_parse_nested_compat(tb, max, rta, len); })

-extern int rtnetlink_send(struct sk_buff *skb, u32 pid, u32 group, int echo);
-extern int rtnl_unicast(struct sk_buff *skb, u32 pid);
-extern int rtnl_notify(struct sk_buff *skb, u32 pid, u32 group,
+extern int rtnetlink_send(struct sk_buff *skb, struct net *net, u32 pid, u32 group, int echo);
+extern int rtnl_unicast(struct sk_buff *skb, struct net *net, u32 pid);
+extern int rtnl_notify(struct sk_buff *skb, struct net *net, u32 pid, u32 group,
         struct nlmsghdr *nlh, gfp_t flags);
-extern void rtnl_set_sk_err(u32 group, int error);
+extern void rtnl_set_sk_err(struct net *net, u32 group, int error);
 extern int rtnetlink_put_metrics(struct sk_buff *skb, u32 *metrics);
 extern int rtnl_put_cacheinfo(struct sk_buff *skb, struct dst_entry *dst,
         u32 id, u32 ts, u32 tsage, long expires,
diff --git a/include/net/net_namespace.h b/include/net/net_namespace.h
index 934c840..f75607a 100644
--- a/include/net/net_namespace.h
+++ b/include/net/net_namespace.h
@@ -10,6 +10,7 @@

 struct proc_dir_entry;
 struct net_device;
+struct sock;
 struct net {
  atomic_t  count;  /* To decided when the network
       *  namespace should be freed.
@@ -29,6 +30,8 @@ struct net {
  struct list_head  dev_base_head;
  struct hlist_head  *dev_name_head;
  struct hlist_head *dev_index_head;
+
+ struct sock   *rtnl;   /* rtnetlink socket */
 };

 #ifdef CONFIG_NET
diff --git a/net/bridge/br_netlink.c b/net/bridge/br_netlink.c
```

```
index a4ffa2b..f5d6933 100644
--- a/net/bridge/br_netlink.c
+++ b/net/bridge/br_netlink.c
@@ -97,10 +97,10 @@ void br_ifinfo_notify(int event, struct net_bridge_port *port)
   kfree_skb(skb);
   goto errout;
  }
- err = rtnl_notify(skb, 0, RTNLGRP_LINK, NULL, GFP_ATOMIC);
+ err = rtnl_notify(skb, &init_net,0, RTNLGRP_LINK, NULL, GFP_ATOMIC);
 errout:
  if (err < 0)
-  rtnl_set_sk_err(RTNLGRP_LINK, err);
+  rtnl_set_sk_err(&init_net, RTNLGRP_LINK, err);
 }

 /*
diff --git a/net/core/fib_rules.c b/net/core/fib_rules.c
index 357bfa0..03c803c 100644
--- a/net/core/fib_rules.c
+++ b/net/core/fib_rules.c
@@ -577,10 +577,10 @@ static void notify_rule_change(int event, struct fib_rule *rule,
   kfree_skb(skb);
   goto errout;
  }
- err = rtnl_notify(skb, pid, ops->nlgroup, nlh, GFP_KERNEL);
+ err = rtnl_notify(skb, &init_net, pid, ops->nlgroup, nlh, GFP_KERNEL);
 errout:
  if (err < 0)
-  rtnl_set_sk_err(ops->nlgroup, err);
+  rtnl_set_sk_err(&init_net, ops->nlgroup, err);
 }

 static void attach_rules(struct list_head *rules, struct net_device *dev)
diff --git a/net/core/neighbour.c b/net/core/neighbour.c
index 27001db..c452584 100644
--- a/net/core/neighbour.c
+++ b/net/core/neighbour.c
@@ -2466,10 +2466,10 @@ static void __neigh_notify(struct neighbour *n, int type, int flags)
   kfree_skb(skb);
   goto errout;
  }
- err = rtnl_notify(skb, 0, RTNLGRP_NEIGH, NULL, GFP_ATOMIC);
+ err = rtnl_notify(skb, &init_net, 0, RTNLGRP_NEIGH, NULL, GFP_ATOMIC);
 errout:
  if (err < 0)
-  rtnl_set_sk_err(RTNLGRP_NEIGH, err);
+  rtnl_set_sk_err(&init_net, RTNLGRP_NEIGH, err);
 }
```

```
 #ifdef CONFIG_ARPD
diff --git a/net/core/rtnetlink.c b/net/core/rtnetlink.c
index 56fc4f9..fc49104 100644
--- a/net/core/rtnetlink.c
+++ b/net/core/rtnetlink.c
@@ -60,7 +60,6 @@ struct rtnl_link
 };

 static DEFINE_MUTEX(rtnl_mutex);
-static struct sock *rtnl;

 void rtnl_lock(void)
 {
@@ -74,9 +73,38 @@ void __rtnl_unlock(void)

 void rtnl_unlock(void)
 {
- mutex_unlock(&rtnl_mutex);
- if (rtnl && rtnl->sk_receive_queue.qlen)
+ struct net *net;
+
+ /*
+  * Loop through all of the rtnl sockets until none of them (in
+  * a live network namespace) have queue packets.
+  *
+  * We have to be careful with the locking here as
+  * sk_data_ready aka rtnetlink_rcv takes the rtnl_mutex.
+  *
+  * To ensure the network namespace does not exit while
+  * we are processing packets on it's rtnl socket we
+  * grab a reference to the network namespace, ignoring
+  * it if the network namespace has already exited.
+  */
+retry:
+ for_each_net(net) {
+  struct sock *rtnl = net->rtnl;
+
+  if (!rtnl || !rtnl->sk_receive_queue.qlen)
+   continue;
+
+  if (!maybe_get_net(net))
+   continue;
+
+  mutex_unlock(&rtnl_mutex);
   rtnl->sk_data_ready(rtnl, 0);
+  mutex_lock(&rtnl_mutex);
+  put_net(net);
```

```
+  goto retry;
+ }
+ mutex_unlock(&rtnl_mutex);
+
  netdev_run_todo();
 }

@@ -462,8 +490,9 @@ size_t rtattr_strlcpy(char *dest, const struct rtattr *rta, size_t size)
  return ret;
 }

-int rtnetlink_send(struct sk_buff *skb, u32 pid, unsigned group, int echo)
+int rtnetlink_send(struct sk_buff *skb, struct net *net, u32 pid, unsigned group, int echo)
 {
+ struct sock *rtnl = net->rtnl;
  int err = 0;

  NETLINK_CB(skb).dst_group = group;
@@ -475,14 +504,17 @@ int rtnetlink_send(struct sk_buff *skb, u32 pid, unsigned group, int
echo)
  return err;
 }

-int rtnl_unicast(struct sk_buff *skb, u32 pid)
+int rtnl_unicast(struct sk_buff *skb, struct net *net, u32 pid)
 {
+ struct sock *rtnl = net->rtnl;
+
  return nlmsg_unicast(rtnl, skb, pid);
 }

-int rtnl_notify(struct sk_buff *skb, u32 pid, u32 group,
+int rtnl_notify(struct sk_buff *skb, struct net *net, u32 pid, u32 group,
    struct nlmsghdr *nlh, gfp_t flags)
 {
+ struct sock *rtnl = net->rtnl;
  int report = 0;

  if (nlh)
@@ -491,8 +523,10 @@ int rtnl_notify(struct sk_buff *skb, u32 pid, u32 group,
  return nlmsg_notify(rtnl, skb, pid, group, report, flags);
 }

-void rtnl_set_sk_err(u32 group, int error)
+void rtnl_set_sk_err(struct net *net, u32 group, int error)
 {
+ struct sock *rtnl = net->rtnl;
+
```

```
  netlink_set_err(rtnl, 0, group, error);
 }

@@ -1205,7 +1239,7 @@ static int rtnl_getlink(struct sk_buff *skb, struct nlmsghdr* nlh, void
*arg)
    kfree_skb(nskb);
    goto errout;
   }
- err = rtnl_unicast(nskb, NETLINK_CB(skb).pid);
+ err = rtnl_unicast(nskb, net, NETLINK_CB(skb).pid);
 errout:
   dev_put(dev);

@@ -1256,10 +1290,10 @@ void rtmsg_ifinfo(int type, struct net_device *dev, unsigned change)
    kfree_skb(skb);
    goto errout;
   }
- err = rtnl_notify(skb, 0, RTNLGRP_LINK, NULL, GFP_KERNEL);
+ err = rtnl_notify(skb, &init_net, 0, RTNLGRP_LINK, NULL, GFP_KERNEL);
 errout:
   if (err < 0)
-   rtnl_set_sk_err(RTNLGRP_LINK, err);
+   rtnl_set_sk_err(&init_net, RTNLGRP_LINK, err);
 }

 /* Protected by RTNL sempahore.  */
@@ -1270,6 +1304,7 @@ static int rtattr_max;

 static int rtnetlink_rcv_msg(struct sk_buff *skb, struct nlmsghdr *nlh)
 {
+ struct net *net = skb->sk->sk_net;
   rtnl_doit_func doit;
   int sz_idx, kind;
   int min_len;
@@ -1298,6 +1333,7 @@ static int rtnetlink_rcv_msg(struct sk_buff *skb, struct nlmsghdr *nlh)
    return -EPERM;

   if (kind == 2 && nlh->nlmsg_flags&NLM_F_DUMP) {
+   struct sock *rtnl;
    rtnl_dumpit_func dumpit;

    dumpit = rtnl_get_dumpit(family, type);
@@ -1305,6 +1341,7 @@ static int rtnetlink_rcv_msg(struct sk_buff *skb, struct nlmsghdr *nlh)
     return -EOPNOTSUPP;

    __rtnl_unlock();
+   rtnl = net->rtnl;
    err = netlink_dump_start(rtnl, skb, nlh, dumpit, NULL);
```

```
   rtnl_lock();
   return err;
@@ -1383,6 +1420,40 @@ static struct notifier_block rtnetlink_dev_notifier = {
  .notifier_call = rtnetlink_event,
 };

+
+static int rtnetlink_net_init(struct net *net)
+{
+ struct sock *sk;
+ sk = netlink_kernel_create(net, NETLINK_ROUTE, RTNLGRP_MAX,
+       rtnetlink_rcv, &rtnl_mutex, THIS_MODULE);
+ if (!sk)
+  return -ENOMEM;
+
+ /* Don't hold an extra reference on the namespace */
+ put_net(sk->sk_net);
+ net->rtnl = sk;
+ return 0;
+}
+
+static void rtnetlink_net_exit(struct net *net)
+{
+ struct sock *sk = net->rtnl;
+ if (sk) {
+  /* At the last minute lie and say this is a socket for the
+   * initial network namespace.  So the socket will be safe to
+   * free.
+   */
+  sk->sk_net = get_net(&init_net);
+  sock_put(sk);
+  net->rtnl = NULL;
+ }
+}
+
+static struct pernet_operations rtnetlink_net_ops = {
+ .init = rtnetlink_net_init,
+ .exit = rtnetlink_net_exit,
+};
+
 void __init rtnetlink_init(void)
 {
  int i;
@@ -1395,10 +1466,9 @@ void __init rtnetlink_init(void)
  if (!rta_buf)
   panic("rtnetlink_init: cannot allocate rta_buf\n");

- rtnl = netlink_kernel_create(&init_net, NETLINK_ROUTE, RTNLGRP_MAX,
```

```
-         rtnetlink_rcv, &rtnl_mutex, THIS_MODULE);
- if (rtnl == NULL)
+ if (register_pernet_subsys(&rtnetlink_net_ops))
    panic("rtnetlink_init: cannot initialize rtnetlink\n");
+
   netlink_set_nonroot(NETLINK_ROUTE, NL_NONROOT_RECV);
   register_netdevice_notifier(&rtnetlink_dev_notifier);

diff --git a/net/decnet/dn_dev.c b/net/decnet/dn_dev.c
index 6e82f7a..8b34b24 100644
--- a/net/decnet/dn_dev.c
+++ b/net/decnet/dn_dev.c
@@ -791,10 +791,10 @@ static void dn_ifaddr_notify(int event, struct dn_ifaddr *ifa)
    kfree_skb(skb);
    goto errout;
   }
- err = rtnl_notify(skb, 0, RTNLGRP_DECnet_IFADDR, NULL, GFP_KERNEL);
+ err = rtnl_notify(skb, &init_net, 0, RTNLGRP_DECnet_IFADDR, NULL, GFP_KERNEL);
  errout:
   if (err < 0)
-   rtnl_set_sk_err(RTNLGRP_DECnet_IFADDR, err);
+   rtnl_set_sk_err(&init_net, RTNLGRP_DECnet_IFADDR, err);
  }

  static int dn_nl_dump_ifaddr(struct sk_buff *skb, struct netlink_callback *cb)
diff --git a/net/decnet/dn_route.c b/net/decnet/dn_route.c
index a16374b..a5df108 100644
--- a/net/decnet/dn_route.c
+++ b/net/decnet/dn_route.c
@@ -1606,7 +1606,7 @@ static int dn_cache_getroute(struct sk_buff *in_skb, struct nlmsghdr
*nlh, void
    goto out_free;
   }

- return rtnl_unicast(skb, NETLINK_CB(in_skb).pid);
+ return rtnl_unicast(skb, &init_net, NETLINK_CB(in_skb).pid);

  out_free:
   kfree_skb(skb);
diff --git a/net/decnet/dn_table.c b/net/decnet/dn_table.c
index a3bdb8d..e09d915 100644
--- a/net/decnet/dn_table.c
+++ b/net/decnet/dn_table.c
@@ -375,10 +375,10 @@ static void dn_rtmsg_fib(int event, struct dn_fib_node *f, int z, u32
tb_id,
    kfree_skb(skb);
    goto errout;
   }
```

```
- err = rtnl_notify(skb, pid, RTNLGRP_DECnet_ROUTE, nlh, GFP_KERNEL);
+ err = rtnl_notify(skb, &init_net, pid, RTNLGRP_DECnet_ROUTE, nlh, GFP_KERNEL);
 errout:
  if (err < 0)
-  rtnl_set_sk_err(RTNLGRP_DECnet_ROUTE, err);
+  rtnl_set_sk_err(&init_net, RTNLGRP_DECnet_ROUTE, err);
 }

 static __inline__ int dn_hash_dump_bucket(struct sk_buff *skb,
diff --git a/net/ipv4/devinet.c b/net/ipv4/devinet.c
index 0ff2ef6..1ae7dcf 100644
--- a/net/ipv4/devinet.c
+++ b/net/ipv4/devinet.c
@@ -1241,10 +1241,10 @@ static void rtmsg_ifa(int event, struct in_ifaddr* ifa, struct nlmsghdr
*nlh,
   kfree_skb(skb);
   goto errout;
  }
- err = rtnl_notify(skb, pid, RTNLGRP_IPV4_IFADDR, nlh, GFP_KERNEL);
+ err = rtnl_notify(skb, &init_net, pid, RTNLGRP_IPV4_IFADDR, nlh, GFP_KERNEL);
 errout:
  if (err < 0)
-  rtnl_set_sk_err(RTNLGRP_IPV4_IFADDR, err);
+  rtnl_set_sk_err(&init_net, RTNLGRP_IPV4_IFADDR, err);
 }

 #ifdef CONFIG_SYSCTL
diff --git a/net/ipv4/fib_semantics.c b/net/ipv4/fib_semantics.c
index 1351a26..33ec960 100644
--- a/net/ipv4/fib_semantics.c
+++ b/net/ipv4/fib_semantics.c
@@ -320,11 +320,11 @@ void rtmsg_fib(int event, __be32 key, struct fib_alias *fa,
   kfree_skb(skb);
   goto errout;
  }
- err = rtnl_notify(skb, info->pid, RTNLGRP_IPV4_ROUTE,
+ err = rtnl_notify(skb, &init_net, info->pid, RTNLGRP_IPV4_ROUTE,
      info->nlh, GFP_KERNEL);
 errout:
  if (err < 0)
-  rtnl_set_sk_err(RTNLGRP_IPV4_ROUTE, err);
+  rtnl_set_sk_err(&init_net, RTNLGRP_IPV4_ROUTE, err);
 }

 /* Return the first fib alias matching TOS with
diff --git a/net/ipv4/ipmr.c b/net/ipv4/ipmr.c
index b8b4b49..c077cc0 100644
--- a/net/ipv4/ipmr.c
```

```
+++ b/net/ipv4/ipmr.c
@@ -321,7 +321,7 @@ static void ipmr_destroy_unres(struct mfc_cache *c)
    e->error = -ETIMEDOUT;
    memset(&e->msg, 0, sizeof(e->msg));

-   rtnl_unicast(skb, NETLINK_CB(skb).pid);
+   rtnl_unicast(skb, &init_net, NETLINK_CB(skb).pid);
   } else
    kfree_skb(skb);
  }
@@ -533,7 +533,7 @@ static void ipmr_cache_resolve(struct mfc_cache *uc, struct mfc_cache
*c)
     memset(&e->msg, 0, sizeof(e->msg));
    }

-   rtnl_unicast(skb, NETLINK_CB(skb).pid);
+   rtnl_unicast(skb, &init_net, NETLINK_CB(skb).pid);
   } else
    ip_mr_forward(skb, c, 0);
  }
diff --git a/net/ipv4/route.c b/net/ipv4/route.c
index a538a30..a710b48 100644
--- a/net/ipv4/route.c
+++ b/net/ipv4/route.c
@@ -2635,7 +2635,7 @@ static int inet_rtm_getroute(struct sk_buff *in_skb, struct nlmsghdr*
nlh, void
  if (err <= 0)
   goto errout_free;

- err = rtnl_unicast(skb, NETLINK_CB(in_skb).pid);
+ err = rtnl_unicast(skb, &init_net, NETLINK_CB(in_skb).pid);
 errout:
  return err;

diff --git a/net/ipv6/addrconf.c b/net/ipv6/addrconf.c
index bdc183e..7dd3f4f 100644
--- a/net/ipv6/addrconf.c
+++ b/net/ipv6/addrconf.c
@@ -3429,7 +3429,7 @@ static int inet6_rtm_getaddr(struct sk_buff *in_skb, struct nlmsghdr*
nlh,
   kfree_skb(skb);
   goto errout_ifa;
  }
- err = rtnl_unicast(skb, NETLINK_CB(in_skb).pid);
+ err = rtnl_unicast(skb, &init_net, NETLINK_CB(in_skb).pid);
 errout_ifa:
  in6_ifa_put(ifa);
 errout:
```

```
@@ -3452,10 +3452,10 @@ static void inet6_ifa_notify(int event, struct inet6_ifaddr *ifa)
   kfree_skb(skb);
   goto errout;
  }
- err = rtnl_notify(skb, 0, RTNLGRP_IPV6_IFADDR, NULL, GFP_ATOMIC);
+ err = rtnl_notify(skb, &init_net, 0, RTNLGRP_IPV6_IFADDR, NULL, GFP_ATOMIC);
 errout:
  if (err < 0)
-  rtnl_set_sk_err(RTNLGRP_IPV6_IFADDR, err);
+  rtnl_set_sk_err(&init_net, RTNLGRP_IPV6_IFADDR, err);
 }

 static inline void ipv6_store_devconf(struct ipv6_devconf *cnf,
@@ -3660,10 +3660,10 @@ void inet6_ifinfo_notify(int event, struct inet6_dev *idev)
   kfree_skb(skb);
   goto errout;
  }
- err = rtnl_notify(skb, 0, RTNLGRP_IPV6_IFADDR, NULL, GFP_ATOMIC);
+ err = rtnl_notify(skb, &init_net, 0, RTNLGRP_IPV6_IFADDR, NULL, GFP_ATOMIC);
 errout:
  if (err < 0)
-  rtnl_set_sk_err(RTNLGRP_IPV6_IFADDR, err);
+  rtnl_set_sk_err(&init_net, RTNLGRP_IPV6_IFADDR, err);
 }

 static inline size_t inet6_prefix_nlmsg_size(void)
@@ -3729,10 +3729,10 @@ static void inet6_prefix_notify(int event, struct inet6_dev *idev,
   kfree_skb(skb);
   goto errout;
  }
- err = rtnl_notify(skb, 0, RTNLGRP_IPV6_PREFIX, NULL, GFP_ATOMIC);
+ err = rtnl_notify(skb, &init_net, 0, RTNLGRP_IPV6_PREFIX, NULL, GFP_ATOMIC);
 errout:
  if (err < 0)
-  rtnl_set_sk_err(RTNLGRP_IPV6_PREFIX, err);
+  rtnl_set_sk_err(&init_net, RTNLGRP_IPV6_PREFIX, err);
 }

 static void __ipv6_ifa_notify(int event, struct inet6_ifaddr *ifp)
diff --git a/net/ipv6/route.c b/net/ipv6/route.c
index f0cf11c..849dfd1 100644
--- a/net/ipv6/route.c
+++ b/net/ipv6/route.c
@@ -2307,7 +2307,7 @@ static int inet6_rtm_getroute(struct sk_buff *in_skb, struct nlmsghdr*
nlh, void
   goto errout;
  }
```

```
- err = rtnl_unicast(skb, NETLINK_CB(in_skb).pid);
+ err = rtnl_unicast(skb, &init_net, NETLINK_CB(in_skb).pid);
 errout:
  return err;
 }
@@ -2337,10 +2337,10 @@ void inet6_rt_notify(int event, struct rt6_info *rt, struct nl_info *info)
   kfree_skb(skb);
   goto errout;
  }
- err = rtnl_notify(skb, pid, RTNLGRP_IPV6_ROUTE, nlh, gfp_any());
+ err = rtnl_notify(skb, &init_net, pid, RTNLGRP_IPV6_ROUTE, nlh, gfp_any());
 errout:
  if (err < 0)
-  rtnl_set_sk_err(RTNLGRP_IPV6_ROUTE, err);
+  rtnl_set_sk_err(&init_net, RTNLGRP_IPV6_ROUTE, err);
 }

 /*
diff --git a/net/sched/act_api.c b/net/sched/act_api.c
index 8528291..8150647 100644
--- a/net/sched/act_api.c
+++ b/net/sched/act_api.c
@@ -660,7 +660,7 @@ act_get_notify(u32 pid, struct nlmsghdr *n, struct tc_action *a, int event)
   return -EINVAL;
  }

- return rtnl_unicast(skb, pid);
+ return rtnl_unicast(skb, &init_net, pid);
 }

 static struct tc_action *
@@ -781,7 +781,7 @@ static int tca_action_flush(struct rtattr *rta, struct nlmsghdr *n, u32 pid)
  nlh->nlmsg_flags |= NLM_F_ROOT;
  module_put(a->ops->owner);
  kfree(a);
- err = rtnetlink_send(skb, pid, RTNLGRP_TC, n->nlmsg_flags&NLM_F_ECHO);
+ err = rtnetlink_send(skb, &init_net, pid, RTNLGRP_TC, n->nlmsg_flags&NLM_F_ECHO);
  if (err > 0)
   return 0;

@@ -844,7 +844,7 @@ tca_action_gd(struct rtattr *rta, struct nlmsghdr *n, u32 pid, int event)

   /* now do the delete */
   tcf_action_destroy(head, 0);
-  ret = rtnetlink_send(skb, pid, RTNLGRP_TC,
+  ret = rtnetlink_send(skb, &init_net, pid, RTNLGRP_TC,
         n->nlmsg_flags&NLM_F_ECHO);
   if (ret > 0)
```

```
    return 0;
@@ -888,7 +888,7 @@ static int tcf_add_notify(struct tc_action *a, u32 pid, u32 seq, int event,
  nlh->nlmsg_len = skb_tail_pointer(skb) - b;
  NETLINK_CB(skb).dst_group = RTNLGRP_TC;

- err = rtnetlink_send(skb, pid, RTNLGRP_TC, flags&NLM_F_ECHO);
+ err = rtnetlink_send(skb, &init_net, pid, RTNLGRP_TC, flags&NLM_F_ECHO);
  if (err > 0)
   err = 0;
  return err;
diff --git a/net/sched/cls_api.c b/net/sched/cls_api.c
index 2754e50..9dcdc88 100644
--- a/net/sched/cls_api.c
+++ b/net/sched/cls_api.c
@@ -361,7 +361,7 @@ static int tfilter_notify(struct sk_buff *oskb, struct nlmsghdr *n,
   return -EINVAL;
  }

- return rtnetlink_send(skb, pid, RTNLGRP_TC, n->nlmsg_flags&NLM_F_ECHO);
+ return rtnetlink_send(skb, &init_net, pid, RTNLGRP_TC, n->nlmsg_flags&NLM_F_ECHO);
 }

 struct tcf_dump_args
diff --git a/net/sched/sch_api.c b/net/sched/sch_api.c
index 95eb0a8..ddbae5a 100644
--- a/net/sched/sch_api.c
+++ b/net/sched/sch_api.c
@@ -872,7 +872,7 @@ static int qdisc_notify(struct sk_buff *oskb, struct nlmsghdr *n,
  }

  if (skb->len)
-  return rtnetlink_send(skb, pid, RTNLGRP_TC, n->nlmsg_flags&NLM_F_ECHO);
+  return rtnetlink_send(skb, &init_net, pid, RTNLGRP_TC, n->nlmsg_flags&NLM_F_ECHO);

 err_out:
  kfree_skb(skb);
@@ -1103,7 +1103,7 @@ static int tclass_notify(struct sk_buff *oskb, struct nlmsghdr *n,
   return -EINVAL;
  }

- return rtnetlink_send(skb, pid, RTNLGRP_TC, n->nlmsg_flags&NLM_F_ECHO);
+ return rtnetlink_send(skb, &init_net, pid, RTNLGRP_TC, n->nlmsg_flags&NLM_F_ECHO);
 }

 struct qdisc_dump_args
diff --git a/net/wireless/wext.c b/net/wireless/wext.c
index 85e5f9d..85805f2 100644
--- a/net/wireless/wext.c
```

```
+++ b/net/wireless/wext.c
@@ -1137,7 +1137,7 @@ static void wireless_nlevent_process(unsigned long data)
  struct sk_buff *skb;

  while ((skb = skb_dequeue(&wireless_nlevent_queue)))
-  rtnl_notify(skb, 0, RTNLGRP_LINK, NULL, GFP_ATOMIC);
+  rtnl_notify(skb, &init_net, 0, RTNLGRP_LINK, NULL, GFP_ATOMIC);
 }

 static DECLARE_TASKLET(wireless_nlevent_tasklet, wireless_nlevent_process, 0);
@@ -1189,6 +1189,9 @@ static void rtmsg_iwinfo(struct net_device *dev, char *event, int
event_len)
  struct sk_buff *skb;
  int err;

+ if (dev->nd_net != &init_net)
+  return;
+
  skb = nlmsg_new(NLMSG_DEFAULT_SIZE, GFP_ATOMIC);
  if (!skb)
   return;
--
1.5.3.rc6.17.g1911
```

Subject: [PATCH 3/5] net: Make the netlink methods in rtnetlink handle multiple network namespaces

Posted by ebiederm on Sat, 29 Sep 2007 01:04:35 GMT

View Forum Message <> Reply to Message

After the previous prep work this just consists of removing checks limiting the code to work in the initial network namespace, and updating rtmsg_ifinfo so we can generate events for devices in something other then the initial network namespace.

Referring to network other network devices like the IFLA_LINK and IFLA_MASTER attributes do, gets interesting if those network devices happen to be in other network namespaces. Currently ifindex numbers are allocated globally so I have taken the path of least resistance and not still report the information even though the devices they are talking about are invisible.

If applications start getting confused or when ifindex

numbers become local to the network namespace we may need
to do something different in the future.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 net/core/rtnetlink.c | 27 +++------------------------
 1 files changed, 3 insertions(+), 24 deletions(-)

diff --git a/net/core/rtnetlink.c b/net/core/rtnetlink.c
index fc49104..809a9fb 100644
--- a/net/core/rtnetlink.c
+++ b/net/core/rtnetlink.c
@@ -741,9 +741,6 @@ static int rtnl_dump_ifinfo(struct sk_buff *skb, struct netlink_callback *cb)
  int s_idx = cb->args[0];
  struct net_device *dev;

- if (net != &init_net)
- return 0;
-
  idx = 0;
  for_each_netdev(net, dev) {
   if (idx < s_idx)
@@ -946,9 +943,6 @@ static int rtnl_setlink(struct sk_buff *skb, struct nlmsghdr *nlh, void *arg)
  struct nlattr *tb[IFLA_MAX+1];
  char ifname[IFNAMSIZ];

- if (net != &init_net)
- return -EINVAL;
-
  err = nlmsg_parse(nlh, sizeof(*ifm), tb, IFLA_MAX, ifla_policy);
  if (err < 0)
   goto errout;
@@ -997,9 +991,6 @@ static int rtnl_dellink(struct sk_buff *skb, struct nlmsghdr *nlh, void *arg)
  struct nlattr *tb[IFLA_MAX+1];
  int err;

- if (net != &init_net)
- return -EINVAL;
-
  err = nlmsg_parse(nlh, sizeof(*ifm), tb, IFLA_MAX, ifla_policy);
  if (err < 0)
   return err;
@@ -1081,9 +1072,6 @@ static int rtnl_newlink(struct sk_buff *skb, struct nlmsghdr *nlh, void
*arg)
  struct nlattr *linkinfo[IFLA_INFO_MAX+1];
  int err;

- if (net != &init_net)

```
-  return -EINVAL;
-
 #ifdef CONFIG_KMOD
 replay:
 #endif
@@ -1210,9 +1198,6 @@ static int rtnl_getlink(struct sk_buff *skb, struct nlmsghdr* nlh, void
*arg)
  struct sk_buff *nskb;
  int err;

- if (net != &init_net)
-  return -EINVAL;
-
  err = nlmsg_parse(nlh, sizeof(*ifm), tb, IFLA_MAX, ifla_policy);
  if (err < 0)
   return err;
@@ -1248,13 +1233,9 @@ errout:

 static int rtnl_dump_all(struct sk_buff *skb, struct netlink_callback *cb)
 {
- struct net *net = skb->sk->sk_net;
  int idx;
  int s_idx = cb->family;

- if (net != &init_net)
-  return 0;
-
  if (s_idx == 0)
   s_idx = 1;
  for (idx=1; idx<NPROTO; idx++) {
@@ -1276,6 +1257,7 @@ static int rtnl_dump_all(struct sk_buff *skb, struct netlink_callback *cb)

 void rtmsg_ifinfo(int type, struct net_device *dev, unsigned change)
 {
+ struct net *net = dev->nd_net;
  struct sk_buff *skb;
  int err = -ENOBUFS;

@@ -1290,10 +1272,10 @@ void rtmsg_ifinfo(int type, struct net_device *dev, unsigned change)
   kfree_skb(skb);
   goto errout;
  }
- err = rtnl_notify(skb, &init_net, 0, RTNLGRP_LINK, NULL, GFP_KERNEL);
+ err = rtnl_notify(skb, net, 0, RTNLGRP_LINK, NULL, GFP_KERNEL);
 errout:
  if (err < 0)
-  rtnl_set_sk_err(&init_net, RTNLGRP_LINK, err);
+  rtnl_set_sk_err(net, RTNLGRP_LINK, err);
```

```
 }

 /* Protected by RTNL sempahore.  */
@@ -1392,9 +1374,6 @@ static int rtnetlink_event(struct notifier_block *this, unsigned long
event, voi
 {
  struct net_device *dev = ptr;

- if (dev->nd_net != &init_net)
-  return NOTIFY_DONE;
-
  switch (event) {
  case NETDEV_UNREGISTER:
   rtmsg_ifinfo(RTM_DELLINK, dev, ~0U);
--
1.5.3.rc6.17.g1911
```

Subject: [PATCH 4/5] net: Make AF_PACKET handle multiple network namespaces
Posted by ebiederm on Sat, 29 Sep 2007 01:07:05 GMT
View Forum Message <> Reply to Message

This is done by making packet_sklist_lock and packet_sklist per
network namespace and adding an additional filter condition on
received packets to ensure they came from the proper network
namespace.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 include/net/net_namespace.h | 4 +
 net/packet/af_packet.c | 129 ++++++++++++++++++++++++++++++++----------
 2 files changed, 87 insertions(+), 46 deletions(-)

diff --git a/include/net/net_namespace.h b/include/net/net_namespace.h
index f75607a..5e9fb47 100644
--- a/include/net/net_namespace.h
+++ b/include/net/net_namespace.h
@@ -32,6 +32,10 @@ struct net {
  struct hlist_head *dev_index_head;

  struct sock *rtnl; /* rtnetlink socket */
+
+ /* List of all packet sockets. */

```
+ rwlock_t  packet_sklist_lock;
+ struct hlist_head packet_sklist;
 };

 #ifdef CONFIG_NET
diff --git a/net/packet/af_packet.c b/net/packet/af_packet.c
index e11000a..1c3a5a8 100644
--- a/net/packet/af_packet.c
+++ b/net/packet/af_packet.c
@@ -135,10 +135,6 @@ dev->hard_header == NULL (ll header is added by device, we cannot
control it)
    packet classifier depends on it.
  */

-/* List of all packet sockets. */
-static HLIST_HEAD(packet_sklist);
-static DEFINE_RWLOCK(packet_sklist_lock);
-
 static atomic_t packet_socks_nr;


@@ -252,9 +248,6 @@ static int packet_rcv_spkt(struct sk_buff *skb, struct net_device *dev,
struct
  struct sock *sk;
  struct sockaddr_pkt *spkt;

- if (dev->nd_net != &init_net)
-  goto out;
-
  /*
   * When we registered the protocol we saved the socket in the data
   * field for just this event.
@@ -276,6 +269,9 @@ static int packet_rcv_spkt(struct sk_buff *skb, struct net_device *dev,
struct
  if (skb->pkt_type == PACKET_LOOPBACK)
   goto out;

+ if (dev->nd_net != sk->sk_net)
+  goto out;
+
  if ((skb = skb_share_check(skb, GFP_ATOMIC)) == NULL)
   goto oom;

@@ -347,7 +343,7 @@ static int packet_sendmsg_spkt(struct kiocb *iocb, struct socket *sock,
   */

  saddr->spkt_device[13] = 0;
- dev = dev_get_by_name(&init_net, saddr->spkt_device);
```

```
+ dev = dev_get_by_name(sk->sk_net, saddr->spkt_device);
  err = -ENODEV;
  if (dev == NULL)
   goto out_unlock;
@@ -455,15 +451,15 @@ static int packet_rcv(struct sk_buff *skb, struct net_device *dev, struct
packet
  int skb_len = skb->len;
  unsigned int snaplen, res;

- if (dev->nd_net != &init_net)
-  goto drop;
-
  if (skb->pkt_type == PACKET_LOOPBACK)
   goto drop;

  sk = pt->af_packet_priv;
  po = pkt_sk(sk);

+ if (dev->nd_net != sk->sk_net)
+  goto drop;
+
  skb->dev = dev;

  if (dev->header_ops) {
@@ -572,15 +568,15 @@ static int tpacket_rcv(struct sk_buff *skb, struct net_device *dev, struct
packe
  struct sk_buff *copy_skb = NULL;
  struct timeval tv;

- if (dev->nd_net != &init_net)
-  goto drop;
-
  if (skb->pkt_type == PACKET_LOOPBACK)
   goto drop;

  sk = pt->af_packet_priv;
  po = pkt_sk(sk);

+ if (dev->nd_net != sk->sk_net)
+  goto drop;
+
  if (dev->header_ops) {
   if (sk->sk_type != SOCK_DGRAM)
    skb_push(skb, skb->data - skb_mac_header(skb));
@@ -738,7 +734,7 @@ static int packet_sendmsg(struct kiocb *iocb, struct socket *sock,
  }
```

```
- dev = dev_get_by_index(&init_net, ifindex);
+ dev = dev_get_by_index(sk->sk_net, ifindex);
  err = -ENXIO;
  if (dev == NULL)
   goto out_unlock;
@@ -805,15 +801,17 @@ static int packet_release(struct socket *sock)
 {
  struct sock *sk = sock->sk;
  struct packet_sock *po;
+ struct net *net;

  if (!sk)
   return 0;

+ net = sk->sk_net;
  po = pkt_sk(sk);

- write_lock_bh(&packet_sklist_lock);
+ write_lock_bh(&net->packet_sklist_lock);
  sk_del_node_init(sk);
- write_unlock_bh(&packet_sklist_lock);
+ write_unlock_bh(&net->packet_sklist_lock);

  /*
   * Unhook packet receive handler.
@@ -927,7 +925,7 @@ static int packet_bind_spkt(struct socket *sock, struct sockaddr *uaddr,
int add
   return -EINVAL;
  strlcpy(name,uaddr->sa_data,sizeof(name));

- dev = dev_get_by_name(&init_net, name);
+ dev = dev_get_by_name(sk->sk_net, name);
  if (dev) {
   err = packet_do_bind(sk, dev, pkt_sk(sk)->num);
   dev_put(dev);
@@ -954,7 +952,7 @@ static int packet_bind(struct socket *sock, struct sockaddr *uaddr, int
addr_len

  if (sll->sll_ifindex) {
   err = -ENODEV;
-  dev = dev_get_by_index(&init_net, sll->sll_ifindex);
+  dev = dev_get_by_index(sk->sk_net, sll->sll_ifindex);
   if (dev == NULL)
    goto out;
  }
@@ -983,9 +981,6 @@ static int packet_create(struct net *net, struct socket *sock, int protocol)
  __be16 proto = (__force __be16)protocol; /* weird, but documented */
  int err;
```

```
- if (net != &init_net)
-  return -EAFNOSUPPORT;
-
  if (!capable(CAP_NET_RAW))
   return -EPERM;
  if (sock->type != SOCK_DGRAM && sock->type != SOCK_RAW &&
@@ -1031,9 +1026,9 @@ static int packet_create(struct net *net, struct socket *sock, int
protocol)
   po->running = 1;
  }

- write_lock_bh(&packet_sklist_lock);
- sk_add_node(sk, &packet_sklist);
- write_unlock_bh(&packet_sklist_lock);
+ write_lock_bh(&net->packet_sklist_lock);
+ sk_add_node(sk, &net->packet_sklist);
+ write_unlock_bh(&net->packet_sklist_lock);
  return(0);
 out:
  return err;
@@ -1151,7 +1146,7 @@ static int packet_getname_spkt(struct socket *sock, struct sockaddr
*uaddr,
   return -EOPNOTSUPP;

  uaddr->sa_family = AF_PACKET;
- dev = dev_get_by_index(&init_net, pkt_sk(sk)->ifindex);
+ dev = dev_get_by_index(sk->sk_net, pkt_sk(sk)->ifindex);
  if (dev) {
   strlcpy(uaddr->sa_data, dev->name, 15);
   dev_put(dev);
@@ -1176,7 +1171,7 @@ static int packet_getname(struct socket *sock, struct sockaddr *uaddr,
  sll->sll_family = AF_PACKET;
  sll->sll_ifindex = po->ifindex;
  sll->sll_protocol = po->num;
- dev = dev_get_by_index(&init_net, po->ifindex);
+ dev = dev_get_by_index(sk->sk_net, po->ifindex);
  if (dev) {
   sll->sll_hatype = dev->type;
   sll->sll_halen = dev->addr_len;
@@ -1228,7 +1223,7 @@ static int packet_mc_add(struct sock *sk, struct packet_mreq_max
*mreq)
  rtnl_lock();

  err = -ENODEV;
- dev = __dev_get_by_index(&init_net, mreq->mr_ifindex);
+ dev = __dev_get_by_index(sk->sk_net, mreq->mr_ifindex);
  if (!dev)
```

```
   goto done;

@@ -1282,7 +1277,7 @@ static int packet_mc_drop(struct sock *sk, struct packet_mreq_max
*mreq)
    if (--ml->count == 0) {
     struct net_device *dev;
     *mlp = ml->next;
-    dev = dev_get_by_index(&init_net, ml->ifindex);
+    dev = dev_get_by_index(sk->sk_net, ml->ifindex);
     if (dev) {
      packet_dev_mc(dev, ml, -1);
      dev_put(dev);
@@ -1310,7 +1305,7 @@ static void packet_flush_mclist(struct sock *sk)
   struct net_device *dev;

   po->mclist = ml->next;
-  if ((dev = dev_get_by_index(&init_net, ml->ifindex)) != NULL) {
+  if ((dev = dev_get_by_index(sk->sk_net, ml->ifindex)) != NULL) {
    packet_dev_mc(dev, ml, -1);
    dev_put(dev);
   }
@@ -1466,12 +1461,10 @@ static int packet_notifier(struct notifier_block *this, unsigned long
msg, void
  struct sock *sk;
  struct hlist_node *node;
  struct net_device *dev = data;
+ struct net *net = dev->nd_net;

- if (dev->nd_net != &init_net)
-  return NOTIFY_DONE;
-
- read_lock(&packet_sklist_lock);
- sk_for_each(sk, node, &packet_sklist) {
+ read_lock(&net->packet_sklist_lock);
+ sk_for_each(sk, node, &net->packet_sklist) {
   struct packet_sock *po = pkt_sk(sk);

   switch (msg) {
@@ -1510,7 +1503,7 @@ static int packet_notifier(struct notifier_block *this, unsigned long msg,
void
    break;
   }
  }
- read_unlock(&packet_sklist_lock);
+ read_unlock(&net->packet_sklist_lock);
  return NOTIFY_DONE;
 }
```

```
@@ -1878,12 +1871,12 @@ static struct notifier_block packet_netdev_notifier = {
 };

 #ifdef CONFIG_PROC_FS
-static inline struct sock *packet_seq_idx(loff_t off)
+static inline struct sock *packet_seq_idx(struct net *net, loff_t off)
 {
  struct sock *s;
  struct hlist_node *node;

- sk_for_each(s, node, &packet_sklist) {
+ sk_for_each(s, node, &net->packet_sklist) {
   if (!off--)
    return s;
  }
@@ -1892,21 +1885,24 @@ static inline struct sock *packet_seq_idx(loff_t off)

 static void *packet_seq_start(struct seq_file *seq, loff_t *pos)
 {
- read_lock(&packet_sklist_lock);
- return *pos ? packet_seq_idx(*pos - 1) : SEQ_START_TOKEN;
+ struct net *net = seq->private;
+ read_lock(&net->packet_sklist_lock);
+ return *pos ? packet_seq_idx(net, *pos - 1) : SEQ_START_TOKEN;
 }

 static void *packet_seq_next(struct seq_file *seq, void *v, loff_t *pos)
 {
+ struct net *net = seq->private;
  ++*pos;
  return  (v == SEQ_START_TOKEN)
-  ? sk_head(&packet_sklist)
+  ? sk_head(&net->packet_sklist)
   : sk_next((struct sock*)v) ;
 }

 static void packet_seq_stop(struct seq_file *seq, void *v)
 {
- read_unlock(&packet_sklist_lock);
+ struct net *net = seq->private;
+ read_unlock(&net->packet_sklist_lock);
 }

 static int packet_seq_show(struct seq_file *seq, void *v)
@@ -1942,7 +1938,26 @@ static const struct seq_operations packet_seq_ops = {

 static int packet_seq_open(struct inode *inode, struct file *file)
 {
```

```
- return seq_open(file, &packet_seq_ops);
+ struct seq_file *seq;
+ int res;
+ res = seq_open(file, &packet_seq_ops);
+ if (!res) {
+  seq = file->private_data;
+  seq->private = get_proc_net(inode);
+  if (!seq->private) {
+   seq_release(inode, file);
+   res = -ENXIO;
+  }
+ }
+ return res;
+}
+
+static int packet_seq_release(struct inode *inode, struct file *file)
+{
+ struct seq_file *seq=  file->private_data;
+ struct net *net = seq->private;
+ put_net(net);
+ return seq_release(inode, file);
 }

 static const struct file_operations packet_seq_fops = {
@@ -1950,15 +1965,37 @@ static const struct file_operations packet_seq_fops = {
  .open  = packet_seq_open,
  .read  = seq_read,
  .llseek  = seq_lseek,
- .release = seq_release,
+ .release = packet_seq_release,
 };

 #endif

+static int packet_net_init(struct net *net)
+{
+ rwlock_init(&net->packet_sklist_lock);
+ INIT_HLIST_HEAD(&net->packet_sklist);
+
+ if (!proc_net_fops_create(net, "packet", 0, &packet_seq_fops))
+  return -ENOMEM;
+
+ return 0;
+}
+
+static void packet_net_exit(struct net *net)
+{
+ proc_net_remove(net, "packet");
```

```
+}
+
+static struct pernet_operations packet_net_ops = {
+ .init = packet_net_init,
+ .exit = packet_net_exit,
+};
+
+
 static void __exit packet_exit(void)
 {
- proc_net_remove(&init_net, "packet");
  unregister_netdevice_notifier(&packet_netdev_notifier);
+ unregister_pernet_subsys(&packet_net_ops);
  sock_unregister(PF_PACKET);
  proto_unregister(&packet_proto);
 }
@@ -1971,8 +2008,8 @@ static int __init packet_init(void)
   goto out;

  sock_register(&packet_family_ops);
+ register_pernet_subsys(&packet_net_ops);
  register_netdevice_notifier(&packet_netdev_notifier);
- proc_net_fops_create(&init_net, "packet", 0, &packet_seq_fops);
 out:
  return rc;
 }
--
1.5.3.rc6.17.g1911
```

Subject: [PATCH 5/5] net: Make AF_UNIX per network namespace safe.
Posted by ebiederm on Sat, 29 Sep 2007 01:08:05 GMT
View Forum Message <> Reply to Message

Because of the global nature of garbage collection, and because of the cost of per namespace hash tables unix_socket_table has been kept global. With a filter added on lookups so we don't see sockets from the wrong namespace.

Currently I don't fold the namesapce into the hash so multiple namespaces using the same socket name will be guaranteed a hash collision.

```
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 net/unix/af_unix.c |  113 ++++++++++++++++++++++++++++++++++++++------------
 1 files changed, 86 insertions(+), 27 deletions(-)

diff --git a/net/unix/af_unix.c b/net/unix/af_unix.c
index 10e7312..2792ef2 100644
--- a/net/unix/af_unix.c
+++ b/net/unix/af_unix.c
@@ -270,7 +270,8 @@ static inline void unix_insert_socket(struct hlist_head *list, struct sock
*sk)
  spin_unlock(&unix_table_lock);
 }

-static struct sock *__unix_find_socket_byname(struct sockaddr_un *sunname,
+static struct sock *__unix_find_socket_byname(struct net *net,
+           struct sockaddr_un *sunname,
           int len, int type, unsigned hash)
 {
  struct sock *s;
@@ -279,6 +280,9 @@ static struct sock *__unix_find_socket_byname(struct sockaddr_un
*sunname,
  sk_for_each(s, node, &unix_socket_table[hash ^ type]) {
   struct unix_sock *u = unix_sk(s);

+  if (s->sk_net != net)
+   continue;
+
   if (u->addr->len == len &&
       !memcmp(u->addr->name, sunname, len))
    goto found;
@@ -288,21 +292,22 @@ found:
  return s;
 }

-static inline struct sock *unix_find_socket_byname(struct sockaddr_un *sunname,
+static inline struct sock *unix_find_socket_byname(struct net *net,
+         struct sockaddr_un *sunname,
          int len, int type,
          unsigned hash)
 {
  struct sock *s;

  spin_lock(&unix_table_lock);
- s = __unix_find_socket_byname(sunname, len, type, hash);
+ s = __unix_find_socket_byname(net, sunname, len, type, hash);
  if (s)
   sock_hold(s);
```

```
  spin_unlock(&unix_table_lock);
  return s;
 }

-static struct sock *unix_find_socket_byinode(struct inode *i)
+static struct sock *unix_find_socket_byinode(struct net *net, struct inode *i)
 {
  struct sock *s;
  struct hlist_node *node;
@@ -312,6 +317,9 @@ static struct sock *unix_find_socket_byinode(struct inode *i)
       &unix_socket_table[i->i_ino & (UNIX_HASH_SIZE - 1)]) {
   struct dentry *dentry = unix_sk(s)->dentry;

+  if (s->sk_net != net)
+   continue;
+
   if(dentry && dentry->d_inode == i)
   {
    sock_hold(s);
@@ -630,9 +638,6 @@ out:

 static int unix_create(struct net *net, struct socket *sock, int protocol)
 {
- if (net != &init_net)
-  return -EAFNOSUPPORT;
-
  if (protocol && protocol != PF_UNIX)
   return -EPROTONOSUPPORT;

@@ -676,6 +681,7 @@ static int unix_release(struct socket *sock)
 static int unix_autobind(struct socket *sock)
 {
  struct sock *sk = sock->sk;
+ struct net *net = sk->sk_net;
  struct unix_sock *u = unix_sk(sk);
  static u32 ordernum = 1;
  struct unix_address * addr;
@@ -702,7 +708,7 @@ retry:
  spin_lock(&unix_table_lock);
  ordernum = (ordernum+1)&0xFFFFF;

- if (__unix_find_socket_byname(addr->name, addr->len, sock->type,
+ if (__unix_find_socket_byname(net, addr->name, addr->len, sock->type,
           addr->hash)) {
   spin_unlock(&unix_table_lock);
   /* Sanity yield. It is unusual case, but yet... */
@@ -722,7 +728,8 @@ out: mutex_unlock(&u->readlock);
  return err;
```

```
 }

-static struct sock *unix_find_other(struct sockaddr_un *sunname, int len,
+static struct sock *unix_find_other(struct net *net,
+        struct sockaddr_un *sunname, int len,
         int type, unsigned hash, int *error)
 {
  struct sock *u;
@@ -740,7 +747,7 @@ static struct sock *unix_find_other(struct sockaddr_un *sunname, int len,
   err = -ECONNREFUSED;
   if (!S_ISSOCK(nd.dentry->d_inode->i_mode))
    goto put_fail;
-  u=unix_find_socket_byinode(nd.dentry->d_inode);
+  u=unix_find_socket_byinode(net, nd.dentry->d_inode);
   if (!u)
    goto put_fail;

@@ -756,7 +763,7 @@ static struct sock *unix_find_other(struct sockaddr_un *sunname, int len,
   }
  } else {
   err = -ECONNREFUSED;
-  u=unix_find_socket_byname(sunname, len, type, hash);
+  u=unix_find_socket_byname(net, sunname, len, type, hash);
   if (u) {
    struct dentry *dentry;
    dentry = unix_sk(u)->dentry;
@@ -778,6 +785,7 @@ fail:
 static int unix_bind(struct socket *sock, struct sockaddr *uaddr, int addr_len)
 {
  struct sock *sk = sock->sk;
+ struct net *net = sk->sk_net;
  struct unix_sock *u = unix_sk(sk);
  struct sockaddr_un *sunaddr=(struct sockaddr_un *)uaddr;
  struct dentry * dentry = NULL;
@@ -852,7 +860,7 @@ static int unix_bind(struct socket *sock, struct sockaddr *uaddr, int
addr_len)

  if (!sunaddr->sun_path[0]) {
   err = -EADDRINUSE;
-  if (__unix_find_socket_byname(sunaddr, addr_len,
+  if (__unix_find_socket_byname(net, sunaddr, addr_len,
          sk->sk_type, hash)) {
    unix_release_addr(addr);
    goto out_unlock;
@@ -918,6 +926,7 @@ static int unix_dgram_connect(struct socket *sock, struct sockaddr *addr,
         int alen, int flags)
 {
  struct sock *sk = sock->sk;
```

```
+ struct net *net = sk->sk_net;
  struct sockaddr_un *sunaddr=(struct sockaddr_un*)addr;
  struct sock *other;
  unsigned hash;
@@ -934,7 +943,7 @@ static int unix_dgram_connect(struct socket *sock, struct sockaddr *addr,
    goto out;

 restart:
-  other=unix_find_other(sunaddr, alen, sock->type, hash, &err);
+  other=unix_find_other(net, sunaddr, alen, sock->type, hash, &err);
   if (!other)
    goto out;

@@ -1014,6 +1023,7 @@ static int unix_stream_connect(struct socket *sock, struct sockaddr
*uaddr,
 {
  struct sockaddr_un *sunaddr=(struct sockaddr_un *)uaddr;
  struct sock *sk = sock->sk;
+ struct net *net = sk->sk_net;
  struct unix_sock *u = unix_sk(sk), *newu, *otheru;
  struct sock *newsk = NULL;
  struct sock *other = NULL;
@@ -1053,7 +1063,7 @@ static int unix_stream_connect(struct socket *sock, struct sockaddr
*uaddr,

 restart:
  /*  Find listening sock. */
- other = unix_find_other(sunaddr, addr_len, sk->sk_type, hash, &err);
+ other = unix_find_other(net, sunaddr, addr_len, sk->sk_type, hash, &err);
  if (!other)
   goto out;

@@ -1329,6 +1339,7 @@ static int unix_dgram_sendmsg(struct kiocb *kiocb, struct socket
*sock,
 {
  struct sock_iocb *siocb = kiocb_to_siocb(kiocb);
  struct sock *sk = sock->sk;
+ struct net *net = sk->sk_net;
  struct unix_sock *u = unix_sk(sk);
  struct sockaddr_un *sunaddr=msg->msg_name;
  struct sock *other = NULL;
@@ -1392,7 +1403,7 @@ restart:
   if (sunaddr == NULL)
    goto out_free;

-  other = unix_find_other(sunaddr, namelen, sk->sk_type,
+  other = unix_find_other(net, sunaddr, namelen, sk->sk_type,
      hash, &err);
```

```
   if (other==NULL)
    goto out_free;
@@ -1998,12 +2009,18 @@ static unsigned int unix_poll(struct file * file, struct socket *sock,
poll_tabl


 #ifdef CONFIG_PROC_FS
-static struct sock *unix_seq_idx(int *iter, loff_t pos)
+struct unix_iter_state {
+ struct net *net;
+ int i;
+};
+static struct sock *unix_seq_idx(struct unix_iter_state *iter, loff_t pos)
 {
  loff_t off = 0;
  struct sock *s;

- for (s = first_unix_socket(iter); s; s = next_unix_socket(iter, s)) {
+ for (s = first_unix_socket(&iter->i); s; s = next_unix_socket(&iter->i, s)) {
+  if (s->sk_net != iter->net)
+   continue;
   if (off == pos)
    return s;
   ++off;
@@ -2014,17 +2031,24 @@ static struct sock *unix_seq_idx(int *iter, loff_t pos)

 static void *unix_seq_start(struct seq_file *seq, loff_t *pos)
 {
+ struct unix_iter_state *iter = seq->private;
  spin_lock(&unix_table_lock);
- return *pos ? unix_seq_idx(seq->private, *pos - 1) : ((void *) 1);
+ return *pos ? unix_seq_idx(iter, *pos - 1) : ((void *) 1);
 }

 static void *unix_seq_next(struct seq_file *seq, void *v, loff_t *pos)
 {
+ struct unix_iter_state *iter = seq->private;
+ struct sock *sk = v;
  ++*pos;

  if (v == (void *)1)
-  return first_unix_socket(seq->private);
- return next_unix_socket(seq->private, v);
+  sk = first_unix_socket(&iter->i);
+ else
+  sk = next_unix_socket(&iter->i, sk);
+ while (sk && (sk->sk_net != iter->net))
+  sk = next_unix_socket(&iter->i, sk);
```

```
+ return sk;
 }

 static void unix_seq_stop(struct seq_file *seq, void *v)
@@ -2088,7 +2112,7 @@ static int unix_seq_open(struct inode *inode, struct file *file)
 {
  struct seq_file *seq;
  int rc = -ENOMEM;
- int *iter = kmalloc(sizeof(int), GFP_KERNEL);
+ struct unix_iter_state *iter = kmalloc(sizeof(*iter), GFP_KERNEL);

  if (!iter)
   goto out;
@@ -2099,7 +2123,12 @@ static int unix_seq_open(struct inode *inode, struct file *file)

  seq      = file->private_data;
  seq->private = iter;
- *iter = 0;
+ iter->i = 0;
+ iter->net = get_proc_net(inode);
+ if (!iter->net) {
+  seq_release_private(inode, file);
+  rc = -ENXIO;
+ }
 out:
  return rc;
 out_kfree:
@@ -2107,12 +2136,20 @@ out_kfree:
  goto out;
 }

+static int unix_seq_release(struct inode *inode, struct file *file)
+{
+ struct seq_file *seq = file->private_data;
+ struct unix_iter_state *iter = seq->private;
+ put_net(iter->net);
+ return seq_release_private(inode, file);
+}
+
 static const struct file_operations unix_seq_fops = {
  .owner  = THIS_MODULE,
  .open  = unix_seq_open,
  .read  = seq_read,
  .llseek  = seq_lseek,
- .release = seq_release_private,
+ .release = unix_seq_release,
 };
```

```
 #endif
@@ -2123,6 +2160,30 @@ static struct net_proto_family unix_family_ops = {
  .owner = THIS_MODULE,
 };

+
+static int unix_net_init(struct net *net)
+{
+ int error = -ENOMEM;
+
+#ifdef CONFIG_PROC_FS
+ if (!proc_net_fops_create(net, "unix", 0, &unix_seq_fops))
+  goto out;
+#endif
+ error = 0;
+out:
+ return 0;
+}
+
+static void unix_net_exit(struct net *net)
+{
+ proc_net_remove(net, "unix");
+}
+
+static struct pernet_operations unix_net_ops = {
+ .init = unix_net_init,
+ .exit = unix_net_exit,
+};
+
 static int __init af_unix_init(void)
 {
  int rc = -1;
@@ -2138,9 +2199,7 @@ static int __init af_unix_init(void)
  }

  sock_register(&unix_family_ops);
-#ifdef CONFIG_PROC_FS
- proc_net_fops_create(&init_net, "unix", 0, &unix_seq_fops);
-#endif
+ register_pernet_subsys(&unix_net_ops);
  unix_sysctl_register();
 out:
  return rc;
@@ -2150,8 +2209,8 @@ static void __exit af_unix_exit(void)
 {
  sock_unregister(PF_UNIX);
  unix_sysctl_unregister();
- proc_net_remove(&init_net, "unix");
```

```
  proto_unregister(&unix_proto);
+ unregister_pernet_subsys(&unix_net_ops);
 }

 module_init(af_unix_init);
--
1.5.3.rc6.17.g1911
```

Subject: Re: [PATCH 2/5] net: Make rtnetlink infrastructure network namespace aware
Posted by Patrick McHardy on Sat, 29 Sep 2007 15:44:35 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:

```
>  void rtnl_unlock(void)
>  {
> - mutex_unlock(&rtnl_mutex);
> - if (rtnl && rtnl->sk_receive_queue.qlen)
> + struct net *net;
> +
> + /*
> +  * Loop through all of the rtnl sockets until none of them (in
> +  * a live network namespace) have queue packets.
> +  *
> +  * We have to be careful with the locking here as
> +  * sk_data_ready aka rtnetlink_rcv takes the rtnl_mutex.
> +  *
> +  * To ensure the network namespace does not exit while
> +  * we are processing packets on it's rtnl socket we
> +  * grab a reference to the network namespace, ignoring
> +  * it if the network namespace has already exited.
> +  */
> +retry:
> + for_each_net(net) {
> +  struct sock *rtnl = net->rtnl;
> +
> +  if (!rtnl || !rtnl->sk_receive_queue.qlen)
> +   continue;
> +
> +  if (!maybe_get_net(net))
> +   continue;
> +
```

> + mutex_unlock(&rtnl_mutex);
>   rtnl->sk_data_ready(rtnl, 0);
> + mutex_lock(&rtnl_mutex);
> + put_net(net);
> + goto retry;
> + }
> + mutex_unlock(&rtnl_mutex);
> +
>   netdev_run_todo();
> }

I'm wondering why this receive queue processing on unlock is still
necessary today, we don't do trylock in rtnetlink_rcv anymore, so
all senders will simply wait until the lock is released and then
process the queue.

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

---

Subject: Re: [PATCH 5/5] net: Make AF_UNIX per network namespace safe.
Posted by Patrick McHardy on Sat, 29 Sep 2007 15:47:56 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Because of the global nature of garbage collection, and because of the
> cost of per namespace hash tables unix_socket_table has been kept
> global. With a filter added on lookups so we don't see sockets from
> the wrong namespace.
>
> Currently I don't fold the namesapce into the hash so multiple
> namespaces using the same socket name will be guaranteed a hash
> collision.

That doesn't sound like a good thing :) Is there a reason for
not avoiding the collisions?

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

---

Subject: Re: [PATCH 2/5] net: Make rtnetlink infrastructure network namespace aware

Posted by ebiederm on Sat, 29 Sep 2007 16:51:14 GMT
View Forum Message <> Reply to Message

Patrick McHardy <kaber@trash.net> writes:

> I'm wondering why this receive queue processing on unlock is still
> necessary today, we don't do trylock in rtnetlink_rcv anymore, so
> all senders will simply wait until the lock is released and then
> process the queue.

Good question, I should probably look. I was lazy and didn't go back and audit why we were doing this. I just coded a routine that I was certain would work. It does appear that we are processing the queue with sk_data_read when we add a message, so this may be completely unnecessary. I will go back and look. If we can remove this bit things should be simpler.

Eric

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

Subject: Re: [PATCH 5/5] net: Make AF_UNIX per network namespace safe.
Posted by ebiederm on Sat, 29 Sep 2007 17:03:05 GMT
View Forum Message <> Reply to Message

Patrick McHardy <kaber@trash.net> writes:

> Eric W. Biederman wrote:
>> Because of the global nature of garbage collection, and because of the
>> cost of per namespace hash tables unix_socket_table has been kept
>> global. With a filter added on lookups so we don't see sockets from
>> the wrong namespace.
>>
>> Currently I don't fold the namesapce into the hash so multiple
>> namespaces using the same socket name will be guaranteed a hash
>> collision.
>
>
> That doesn't sound like a good thing :) Is there a reason for
> not avoiding the collisions?

Two reasons. Minimizing the size of the changes to make review easier, and I don't know if hash collisions are likely in practice or if they matter. I don't believe we can't physically collide and

have the same inode because we make a node in the filesystem. The
abstract domain is local to linux and so people don't use it as much.

All of which boils down to. I don't see it matter a heck of a lot
especially initially. So I did the traditional unix thing and started
with a simple and stupid implementation. But it didn't quite feel
right to me either so I documented it.

Whipping up a patch to take the namespace into account in mkname
doesn't look to hard though.

Eric

_____

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

---

Subject: Re: [PATCH 2/5] net: Make rtnetlink infrastructure network namespace aware
Posted by Patrick McHardy on Sat, 29 Sep 2007 17:48:03 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Patrick McHardy <kaber@trash.net> writes:
>
>
>>I'm wondering why this receive queue processing on unlock is still
>>necessary today, we don't do trylock in rtnetlink_rcv anymore, so
>>all senders will simply wait until the lock is released and then
>>process the queue.
>
>
> Good question, I should probably look. I was lazy and didn't go back
> and audit why we were doing this. I just coded a routine that I was
> certain would work. It does appear that we are processing the queue
> with sk_data_read when we add a message, so this may be completely
> unnecessary. I will go back and look. If we can remove this bit
> things should be simpler.

Maybe I can save you some time: we used to do down_trylock()
for the rtnl mutex, so senders would simply return if someone
else was already processing the queue *or* the rtnl was locked
for some other reason. In the first case the process already
processing the queue would also process the new messages, but
if it the rtnl was locked for some other reason (for example
during module registration) the message would sit in the

queue until the next rtnetlink sendmsg call, which is why
rtnl_unlock does queue processing. Commit 6756ae4b changed
the down_trylock to mutex_lock, so senders will now simply wait
until the mutex is released and then call netlink_run_queue
themselves. This means its not needed anymore.

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

---

Subject: Re: [PATCH 5/5] net: Make AF_UNIX per network namespace safe.
Posted by Patrick McHardy on Sat, 29 Sep 2007 17:50:14 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Patrick McHardy <kaber@trash.net> writes:
>
>>>>Currently I don't fold the namesapce into the hash so multiple
>>>>namespaces using the same socket name will be guaranteed a hash
>>>>collision.
>>
>>
>>That doesn't sound like a good thing :) Is there a reason for
>>not avoiding the collisions?
>
>
> Two reasons. Minimizing the size of the changes to make review
> easier, and I don't know if hash collisions are likely in practice
> or if they matter. I don't believe we can't physically collide and
> have the same inode because we make a node in the filesystem. The
> abstract domain is local to linux and so people don't use it as much.
>
> All of which boils down to. I don't see it matter a heck of a lot
> especially initially. So I did the traditional unix thing and started
> with a simple and stupid implementation. But it didn't quite feel
> right to me either so I documented it.
>
> Whipping up a patch to take the namespace into account in mkname
> doesn't look to hard though.

It doesn't look like it would increase patch size significantly
(about 4 more changed lines), but it could of course be done in
a follow-up patch.

_____

Subject: Re: [PATCH 2/5] net: Make rtnetlink infrastructure network namespace aware
Posted by ebiederm on Sat, 29 Sep 2007 21:00:48 GMT
View Forum Message <> Reply to Message

Patrick McHardy <kaber@trash.net> writes:

> Maybe I can save you some time: we used to do down_trylock()
> for the rtnl mutex, so senders would simply return if someone
> else was already processing the queue *or* the rtnl was locked
> for some other reason. In the first case the process already
> processing the queue would also process the new messages, but
> if it the rtnl was locked for some other reason (for example
> during module registration) the message would sit in the
> queue until the next rtnetlink sendmsg call, which is why
> rtnl_unlock does queue processing. Commit 6756ae4b changed
> the down_trylock to mutex_lock, so senders will now simply wait
> until the mutex is released and then call netlink_run_queue
> themselves. This means its not needed anymore.

Sounds reasonable.

I started looking through the code paths and I currently cannot
see anything that would leave a message on a kernel rtnl socket.

However I did a quick test adding a WARN_ON if there were any messages
found in the queue during rtnl_unlock and I found this code path
getting invoked from linkwatch_event. So there is clearly something I
don't understand, and it sounds at odds just a bit from your
description.

If we can remove the extra queue processing that would be great,
as it looks like a nice way to simplify the locking and the special
cases in the code.

Eric

_____

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

Subject: Re: Re: [PATCH 2/5] net: Make rtnetlink infrastructure network namespace aware
Posted by dlunev on Sun, 30 Sep 2007 13:13:15 GMT
View Forum Message <> Reply to Message

Hmm, so it looks like we do not need this queue processing at all...

Regards,
 Den

Eric W. Biederman wrote:
> Patrick McHardy <kaber@trash.net> writes:
>
>> Maybe I can save you some time: we used to do down_trylock()
>> for the rtnl mutex, so senders would simply return if someone
>> else was already processing the queue *or* the rtnl was locked
>> for some other reason. In the first case the process already
>> processing the queue would also process the new messages, but
>> if it the rtnl was locked for some other reason (for example
>> during module registration) the message would sit in the
>> queue until the next rtnetlink sendmsg call, which is why
>> rtnl_unlock does queue processing. Commit 6756ae4b changed
>> the down_trylock to mutex_lock, so senders will now simply wait
>> until the mutex is released and then call netlink_run_queue
>> themselves. This means its not needed anymore.
>
> Sounds reasonable.
>
> I started looking through the code paths and I currently cannot
> see anything that would leave a message on a kernel rtnl socket.
>
> However I did a quick test adding a WARN_ON if there were any messages
> found in the queue during rtnl_unlock and I found this code path
> getting invoked from linkwatch_event. So there is clearly something I
> don't understand, and it sounds at odds just a bit from your
> description.
>
> If we can remove the extra queue processing that would be great,
> as it looks like a nice way to simplify the locking and the special
> cases in the code.
>
> Eric

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

Subject: Re: [PATCH 2/5] net: Make rtnetlink infrastructure network namespace aware
Posted by Patrick McHardy on Sun, 30 Sep 2007 15:39:54 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Patrick McHardy <kaber@trash.net> writes:
>
>
>>Maybe I can save you some time: we used to do down_trylock()
>>for the rtnl mutex, so senders would simply return if someone
>>else was already processing the queue *or* the rtnl was locked
>>for some other reason. In the first case the process already
>>processing the queue would also process the new messages, but
>>if it the rtnl was locked for some other reason (for example
>>during module registration) the message would sit in the
>>queue until the next rtnetlink sendmsg call, which is why
>>rtnl_unlock does queue processing. Commit 6756ae4b changed
>>the down_trylock to mutex_lock, so senders will now simply wait
>>until the mutex is released and then call netlink_run_queue
>>themselves. This means its not needed anymore.
>
>
> Sounds reasonable.
>
> I started looking through the code paths and I currently cannot
> see anything that would leave a message on a kernel rtnl socket.
>
> However I did a quick test adding a WARN_ON if there were any messages
> found in the queue during rtnl_unlock and I found this code path
> getting invoked from linkwatch_event. So there is clearly something I
> don't understand, and it sounds at odds just a bit from your
> description.

That sounds like a bug. Did you place the WARN_ON before or after
the mutex_unlock()?

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

Subject: Re: Re: [PATCH 2/5] net: Make rtnetlink infrastructure network namespace aware
Posted by den on Mon, 01 Oct 2007 08:26:52 GMT
View Forum Message <> Reply to Message

Patrick McHardy wrote:
> Eric W. Biederman wrote:
>> Patrick McHardy <kaber@trash.net> writes:
>>
>>
>>> Maybe I can save you some time: we used to do down_trylock()
>>> for the rtnl mutex, so senders would simply return if someone
>>> else was already processing the queue *or* the rtnl was locked
>>> for some other reason. In the first case the process already
>>> processing the queue would also process the new messages, but
>>> if it the rtnl was locked for some other reason (for example
>>> during module registration) the message would sit in the
>>> queue until the next rtnetlink sendmsg call, which is why
>>> rtnl_unlock does queue processing. Commit 6756ae4b changed
>>> the down_trylock to mutex_lock, so senders will now simply wait
>>> until the mutex is released and then call netlink_run_queue
>>> themselves. This means its not needed anymore.
>>
>> Sounds reasonable.
>>
>> I started looking through the code paths and I currently cannot
>> see anything that would leave a message on a kernel rtnl socket.
>>
>> However I did a quick test adding a WARN_ON if there were any messages
>> found in the queue during rtnl_unlock and I found this code path
>> getting invoked from linkwatch_event. So there is clearly something I
>> don't understand, and it sounds at odds just a bit from your
>> description.
>
>
> That sounds like a bug. Did you place the WARN_ON before or after
> the mutex_unlock()?

The presence of the message in the queue during rtnl_unlock is quite
possible as normal user->kernel message processing path for rtnl is the
following:

```
netlink_sendmsg
  netlink_unicast
    netlink_sendskb
      skb_queue_tail
      netlink_data_ready
        rtnetlink_rcv
          mutex_lock(&rtnl_mutex);
          netlink_run_queue(sk, qlen, &rtnetlink_rcv_msg);
          mutex_unlock(&rtnl_mutex);
```

so, the presence of the packet in the rtnl queue on rtnl_unlock is

normal race with a rtnetlink_rcv for me.

Regards,
Den

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

---

Subject: Re: Re: [PATCH 2/5] net: Make rtnetlink infrastructure network namespace aware
Posted by ebiederm on Mon, 01 Oct 2007 08:45:07 GMT
View Forum Message <> Reply to Message

"Denis V. Lunev" <den@sw.ru> writes:

> The presence of the message in the queue during rtnl_unlock is quite
> possible as normal user->kernel message processing path for rtnl is the
> following:
>
> netlink_sendmsg
> netlink_unicast
> netlink_sendskb
> skb_queue_tail
> netlink_data_ready
> rtnetlink_rcv
> mutex_lock(&rtnl_mutex);
> netlink_run_queue(sk, qlen, &rtnetlink_rcv_msg);
> mutex_unlock(&rtnl_mutex);
>
> so, the presence of the packet in the rtnl queue on rtnl_unlock is
> normal race with a rtnetlink_rcv for me.

Yes. That is what I saw in practice as well.
Thanks for confirming this.

It happened to reproducible because I had a dhcp client asking
for a list of links in parallel with the actual link coming up
during boot.

Looking at netlink_unicast and netlink_broadcast I am generally
convinced that we can remove the call of sk_data_ready in
rtnl_unlock. I think those are the only two possible paths
through there and I don't see how we could miss a processing a
packet on the way through there.

What would be nice is if we could figure out how to eliminate

this race. As that would allow netlink packets to be processed
synchronously and we could actually use current for security
checks, and for getting the context of the calling process.

Right now we are 99% of the way there but because of the above
race the code must all be written as if netlink packets were coming
in completely asynchronously. Which is unfortunate and a pain.

Eric

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

---

Subject: Re: [PATCH 1/5] net: Modify all rtnetlink methods to only work in the initial namespace
Posted by den on Wed, 10 Oct 2007 12:33:50 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Before I can enable rtnetlink to work in all network namespaces
> I need to be certain that something won't break. So this
> patch deliberately disables all of the rtnletlink methods in everything
> except the initial network namespace. After the methods have been
> audited this extra check can be disabled.
>
[...]
> static int br_dump_ifinfo(struct sk_buff *skb, struct netlink_callback *cb)
> {
> + struct net *net = skb->sk->sk_net;
>   struct net_device *dev;
>   int idx;
>

I've read some code today greping 'init_net.loopback_dev' and found
interesting non-trivial for me issue.

Network namespace is extracted from the packet in two different ways in
TCP. This is a socket for outgoing path and a device for incoming.
Though, there are some places called uniformly both from incoming and
outgoing path.

Typical example is netfilters. They are called uniformly all around the
code. The prototype is the following:

static unsigned int reject6_target(struct sk_buff **pskb,
                   const struct net_device *in,

const struct net_device *out,
unsigned int hooknum,
const struct xt_target *target,
const void *targinfo);

So, we are bound to the following options:
- perform additional non-uniform hacks around to place 'struct net' into
  other and other structures like xt_target
- add 7th parameter here and over
- introduce an skb_net field in the 'struct sk_buff' making all code
  uniform, at least when we have an skb

I think that this is not the last place with such a parameter list and
we should make a decision at this point when the code in not mainline yet.

As far as I understand, netfilters are not touched by the Eric and we
can face some non-trivial problems there.

So, if my point about uniformity is valid, this patchset looks wrong and
should be re-worked :(

Regards,
Den

Subject: Re: [PATCH 1/5] net: Modify all rtnetlink methods to only work in the initial namespace
Posted by Daniel Lezcano on Wed, 10 Oct 2007 14:05:26 GMT
View Forum Message <> Reply to Message

Denis V. Lunev wrote:
> Eric W. Biederman wrote:
>> Before I can enable rtnetlink to work in all network namespaces
>> I need to be certain that something won't break. So this
>> patch deliberately disables all of the rtnletlink methods in everything
>> except the initial network namespace. After the methods have been
>> audited this extra check can be disabled.
>>
> [...]
>> static int br_dump_ifinfo(struct sk_buff *skb, struct netlink_callback *cb)
>> {
>> + struct net *net = skb->sk->sk_net;
>> struct net_device *dev;
>> int idx;
>>
>
> I've read some code today greping 'init_net.loopback_dev' and found
> interesting non-trivial for me issue.

>
> Network namespace is extracted from the packet in two different ways in
> TCP. This is a socket for outgoing path and a device for incoming.
> Though, there are some places called uniformly both from incoming and
> outgoing path.
>
> Typical example is netfilters. They are called uniformly all around the
> code. The prototype is the following:
>
> static unsigned int reject6_target(struct sk_buff **pskb,
>   const struct net_device *in,
>   const struct net_device *out,
>   unsigned int hooknum,
>   const struct xt_target *target,
>   const void *targinfo);
>

Thanks Denis for auditing the code.

As far as I see, struct net_device *in is NULL for outgoing traffic and
struct net_device *out is NULL for ingress traffic. Except for the
FORWARD rules where both are filled. If we are following network
namespace semantic, we should not have two network devices belonging to
two differents namespaces, right ?
In this case, the following line of code should be sufficient to
retrieve the network namespace, no ?

struct net *net = in?in->nd_net:out->nd_net;

> So, we are bound to the following options:
> - perform additional non-uniform hacks around to place 'struct net' into
>   other and other structures like xt_target
> - add 7th parameter here and over
> - introduce an skb_net field in the 'struct sk_buff' making all code
>   uniform, at least when we have an skb
>
> I think that this is not the last place with such a parameter list and
> we should make a decision at this point when the code in not mainline yet.
>
> As far as I understand, netfilters are not touched by the Eric and we
> can face some non-trivial problems there.

In Eric's git tree:
http://git.kernel.org/?p=linux/kernel/git/ebiederm/linux-2.6-netns.git

There are some modifications concerning
net/ipv4/netfiler/iptable_filter.c and at the ipt_hook function, there is:

struct net *net = (in?in:out)->nd_net;

> So, if my point about uniformity is valid, this patchset looks wrong and
> should be re-worked :(

As Eric said, we want to build the network namespace step by step,
taking care of not breaking the init network namespace.

If you want to make iptables per namespace or catch problems before the
code goes to Dave's tree, IMHO it will be more convenient to post to
containers@ the patches against netns49, where the modifications will be
in a network namespace big picture.

Regards.

-- Daniel

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

---

Subject: Re: [PATCH 1/5] net: Modify all rtnetlink methods to only work in the initial namespace
Posted by den on Wed, 10 Oct 2007 14:29:05 GMT
View Forum Message <> Reply to Message

Daniel Lezcano wrote:
> struct net *net = in?in->nd_net:out->nd_net;
>
>> So, we are bound to the following options:
>> - perform additional non-uniform hacks around to place 'struct net' into
>> other and other structures like xt_target
>> - add 7th parameter here and over
>> - introduce an skb_net field in the 'struct sk_buff' making all code
>> uniform, at least when we have an skb
>>
>> I think that this is not the last place with such a parameter list and
>> we should make a decision at this point when the code in not mainline
>> yet.
>>
>> As far as I understand, netfilters are not touched by the Eric and we
>> can face some non-trivial problems there.
>
> In Eric's git tree:
> http://git.kernel.org/?p=linux/kernel/git/ebiederm/linux-2.6-netns.git
>

> There are some modifications concerning
> net/ipv4/netfiler/iptable_filter.c and at the ipt_hook function, there is:
>
> struct net *net = (in?in:out)->nd_net;
>
>> So, if my point about uniformity is valid, this patchset looks wrong and
>> should be re-worked :(
>
> As Eric said, we want to build the network namespace step by step,
> taking care of not breaking the init network namespace.
>
> If you want to make iptables per namespace or catch problems before the
> code goes to Dave's tree, IMHO it will be more convenient to post to
> containers@ the patches against netns49, where the modifications will be
> in a network namespace big picture.
>

my point is somewhat another. Yes, this is enough for that place. If so,
I must scatter these checks all around in the netfilters code. Brr.

In forward chain the situation is different for Layer3 switching. Let's
assume that we have an OpenVZ scheme, where the packet flows from socket
to device and after that from device to device via forwarding path. You
can't call skb_orphan on namespace switching as this breaks UDP flow
regulation. Virtual network device is fast while real Ethernet is slow,
packets will be dropped on queue in real device. So, the situation with
packet on send path with a socket from other namespace is possible :(

I just pray for uniformity to concentrate on the code rather than on
guesses on which path we are :(

Regards,
 Den

Subject: Re: [PATCH 1/5] net: Modify all rtnetlink methods to only work in the initial namespace
Posted by ebiederm on Wed, 10 Oct 2007 19:37:04 GMT
View Forum Message <> Reply to Message

"Denis V. Lunev" <den@sw.ru> writes:

> Eric W. Biederman wrote:
>> Before I can enable rtnetlink to work in all network namespaces

>> I need to be certain that something won't break. So this
>> patch deliberately disables all of the rtnletlink methods in everything
>> except the initial network namespace. After the methods have been
>> audited this extra check can be disabled.
>>
> [...]
>> static int br_dump_ifinfo(struct sk_buff *skb, struct netlink_callback *cb)
>> {
>> + struct net *net = skb->sk->sk_net;
>>   struct net_device *dev;
>>   int idx;
>>
>
> I've read some code today greping 'init_net.loopback_dev' and found
> interesting non-trivial for me issue.
>
> Network namespace is extracted from the packet in two different ways in
> TCP. This is a socket for outgoing path and a device for incoming.
> Though, there are some places called uniformly both from incoming and
> outgoing path.
>
> Typical example is netfilters. They are called uniformly all around the
> code. The prototype is the following:
>
> static unsigned int reject6_target(struct sk_buff **pskb,
>   const struct net_device *in,
>   const struct net_device *out,
>   unsigned int hooknum,
>   const struct xt_target *target,
>   const void *targinfo);
>
> So, we are bound to the following options:
> - perform additional non-uniform hacks around to place 'struct net' into
>   other and other structures like xt_target
> - add 7th parameter here and over

> - introduce an skb_net field in the 'struct sk_buff' making all code
>   uniform, at least when we have an skb

No. That bloats a sk_buff, changes the semantics of moving a skb
around, and decreases performance (because we have to maintain the
field on a fast path).

There will not be a skb_net field.

The entire concept of skb_net is a maintenance disaster.

> I think that this is not the last place with such a parameter list and

> we should make a decision at this point when the code in not mainline yet.

Certainly that is what I have a proof of concept tree for. So we can see how these things look before we merge them.

> As far as I understand, netfilters are not touched by the Eric and we
> can face some non-trivial problems there.

No. In my proof of concept tree I should have working per network namespace netfilter code. My intention was to just do enough to see what the impact would be so most of the netfilter code (in my tree) insists on running in the initial network namespace. But there are a few pieces that are fully converted. Please take a look.

> So, if my point about uniformity is valid, this patchset looks wrong and
> should be re-worked :(

This patchset does need to get rebased on top of net-2.6.25 when it opens and hopefully your patchset to remove the unnecessary work in rtnl_unlock, and to really process netlink requests in process context. I see a need for the more fundamental change you seem to be advocating.

Differentiating between the incoming and the outgoing code paths is something we already do permission checking, for locking, for sleeping, etc. Modifying the code requires reading and understanding it in context. That is the nature of code.

This does make large patches going across the entire networking stack making something a network namespace parameter difficult, but it should not cause any problem for maintenance or other work on the code. As shown by the fact that even outside the tree rebasing my network namespace patches has not been all that difficult.

So no I don't think uniformity, or beauty or elegance is what we are after right now. Trying to hard in that direction ultimately obfuscates the code.

What we want is something that is simple, straight forward, and doesn't require you to be an expert in network namespaces to understand the code or the patches.

In the particular case of the netfilter hooks we don't have a network namespace parameter laying around before we call NF_HOOK, and the idiom "net = (in?in:out)->nd_net" seems perfectly accurate so it seems reasonable to me to derive the network namespace that way in generic code. Although thinking about this. We know which hooks we are being called from so we may in fact actually know

if which of in or out must be valid when we get to the netfilter
hook.

Eric

Subject: Re: [PATCH 1/5] net: Modify all rtnetlink methods to only work in the initial namespace
Posted by den on Thu, 11 Oct 2007 06:35:58 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
[..]
>> - introduce an skb_net field in the 'struct sk_buff' making all code
>> uniform, at least when we have an skb
>
> No. That bloats a sk_buff, changes the semantics of moving a skb
> around, and decreases performance (because we have to maintain the
> field on a fast path).
>
> There will not be a skb_net field.
>
> The entire concept of skb_net is a maintenance disaster.
>
>> I think that this is not the last place with such a parameter list and
>> we should make a decision at this point when the code in not mainline yet.
>
> Certainly that is what I have a proof of concept tree for. So we can
> see how these things look before we merge them.
>
>> As far as I understand, netfilters are not touched by the Eric and we
>> can face some non-trivial problems there.
>
> No. In my proof of concept tree I should have working per network
> namespace netfilter code. My intention was to just do enough to see
> what the impact would be so most of the netfilter code (in my tree)
> insists on running in the initial network namespace. But there are
> a few pieces that are fully converted. Please take a look.
>
>> So, if my point about uniformity is valid, this patchset looks wrong and
>> should be re-worked :(
>
> This patchset does need to get rebased on top of net-2.6.25 when it
> opens and hopefully your patchset to remove the unnecessary work in
> rtnl_unlock, and to really process netlink requests in process
> context. I see a need for the more fundamental change you seem to
> be advocating.

I see that current patchset of RTNL code is attached. I'll start the
next piece of work next week after some reaction from people.

[..]
> In the particular case of the netfilter hooks we don't have a
> network namespace parameter laying around before we call NF_HOOK,
> and the idiom "net = (in?in:out)->nd_net" seems perfectly accurate
> so it seems reasonable to me to derive the network namespace that
> way in generic code. Although thinking about this. We know which
> hooks we are being called from so we may in fact actually know
> if which of in or out must be valid when we get to the netfilter
> hook.

I understand your position. But still have some points :)

First. A real-life usecase we have recently fixed in the OpenVZ. I have
described it in the previous post, but (may be) not in great details.

UDP buffers management for outgoing traffic.
The packet carries over a destructor, which is called in skb_orphan in
the real networking device. This destructor allows to queue more
packets. There is no problem in the current implementation. But there is
one after namespace introduction in the following configuration:
    [NS1] <-> [NS2] <-> [world]
There are two namespaces on the host. One of them is connected to the
outside world via another and the packet follows usual forwarding path
in NS2.

The problem: if we call skb_orphan in [NS1] outgoing device, there is a
great possibility to have a really huge packet drop in [NS2] on queuing
operation.

In OpenVZ we have added skb_orphan into receive path (tcp_v4_rcv,
__udp4_lib_rcv etc) and removed one from our virtual devices. As
unfortunate side effect we have packet in NS2 with a socket from NS1.

So, we should have an architectural solution for this from a beginning.
It will be too late to hack around and change namespace lookup scheme.

I see that we should adopt a generic approach:
- each concrete packet belongs to a concrete namespace
- if function has a packet as a parameter, we should get namespace from
packet
- if skb->dev is present we should get namespace as skb->dev->nd_net
- otherwise we should get skb->sk->sk_net. If skb->sk is NULL -> this is
 a bug

So, for the case of all netfilter calls we'll have a pskb->dev->nd_net

defined correctly.

Regards,
Den

Subject: Re: [PATCH 1/5] net: Modify all rtnetlink methods to only work in the initial namespace
Posted by ebiederm on Thu, 11 Oct 2007 07:57:11 GMT
View Forum Message <> Reply to Message

"Denis V. Lunev" <den@sw.ru> writes:

>> This patchset does need to get rebased on top of net-2.6.25 when it
>> opens and hopefully your patchset to remove the unnecessary work in
>> rtnl_unlock, and to really process netlink requests in process
>> context. I see a need for the more fundamental change you seem to
>> be advocating.

Grr. That last sentence should have been I do not see a need for the more
fundamental change you seem to be advocating.

> I see that current patchset of RTNL code is attached. I'll start the
> next piece of work next week after some reaction from people.

> I understand your position. But still have some points :)
>
> First. A real-life usecase we have recently fixed in the OpenVZ. I have
> described it in the previous post, but (may be) not in great details.
>
> UDP buffers management for outgoing traffic.
> The packet carries over a destructor, which is called in skb_orphan in
> the real networking device. This destructor allows to queue more
> packets. There is no problem in the current implementation. But there is
> one after namespace introduction in the following configuration:
> [NS1] <-> [NS2] <-> [world]
> There are two namespaces on the host. One of them is connected to the
> outside world via another and the packet follows usual forwarding path
> in NS2.

Yes. We can't quite do that today because of the missing ipv4 support,
but the basic infrastructure exists to describe this is already
merged.

> The problem: if we call skb_orphan in [NS1] outgoing device, there is a
> great possibility to have a really huge packet drop in [NS2] on queuing
> operation.

Yes. Removing skb_orphan from veth to solve this looks like a
pretty substantial hack. A clean solution is more likely to resemble
ethernet pause frames so we temporarily plug the virtual device.
Although there are issues with that as currently virtual devices
don't have queues.

> In OpenVZ we have added skb_orphan into receive path (tcp_v4_rcv,
> __udp4_lib_rcv etc) and removed one from our virtual devices. As
> unfortunate side effect we have packet in NS2 with a socket from NS1.

That also does some really nasty things to accounting. Using
the macvlan devices solve all of this rather neatly and with
higher performance.

> So, we should have an architectural solution for this from a beginning.
> It will be too late to hack around and change namespace lookup scheme.

However what you are specifically concerned about seems to be
using skb->sk->sk_net. Currently the only place I use this
is in the rtnetlink code because the functions that process packets
are not also passed a socket. Those functions we could easily
pass in an explicit namespace or a socket parameter, and so those
functions would not care.

> I see that we should adopt a generic approach:
> - each concrete packet belongs to a concrete namespace
> - if function has a packet as a parameter, we should get namespace from
> packet

This approach when suggested in earlier review was pretty
substantially shot down. Shrinking the size of struct sk_buff is
currently an ongoing todo for the linux networking stack.

> - if skb->dev is present we should get namespace as skb->dev->nd_net
> - otherwise we should get skb->sk->sk_net. If skb->sk is NULL -> this is
> a bug

Currently killing skb->dev is a todo list item, so using it more is
probably the wrong approach.

I do agree the duplication information between fields on the skb
and fields passed to functions is a pain. Moving more onto the
skb seems contrary to the how the rest of the networking stack would
like to go, so it is likely better to simply remove skb->dev and pass
an explicit dev parameter instead. Please note that parameters passed
explicitly will live in registers and will be quite fast.

> So, for the case of all netfilter calls we'll have a pskb->dev->nd_net
> defined correctly.

That would certainly be a nice extra, but that really seems the wrong
way to go. I would rather add an explicit struct net *net parameter
to nf_hookfn.

Eric

Subject: Re: [PATCH 1/5] net: Modify all rtnetlink methods to only work in the initial namespace
Posted by den on Thu, 11 Oct 2007 12:17:19 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> "Denis V. Lunev" <den@sw.ru> writes:
>
>>> This patchset does need to get rebased on top of net-2.6.25 when it
>>> opens and hopefully your patchset to remove the unnecessary work in
>>> rtnl_unlock, and to really process netlink requests in process
>>> context. I see a need for the more fundamental change you seem to
>>> be advocating.
>
> Grr. That last sentence should have been I do not see a need for the more
> fundamental change you seem to be advocating.

why?

>> First. A real-life usecase we have recently fixed in the OpenVZ. I have
>> described it in the previous post, but (may be) not in great details.
>>
>> UDP buffers management for outgoing traffic.
>> The packet carries over a destructor, which is called in skb_orphan in
>> the real networking device. This destructor allows to queue more
>> packets. There is no problem in the current implementation. But there is
>> one after namespace introduction in the following configuration:
>>   [NS1] <-> [NS2] <-> [world]
>> There are two namespaces on the host. One of them is connected to the
>> outside world via another and the packet follows usual forwarding path
>> in NS2.
>
> Yes. We can't quite do that today because of the missing ipv4 support,
> but the basic infrastructure exists to describe this is already
> merged.
>
>> The problem: if we call skb_orphan in [NS1] outgoing device, there is a
>> great possibility to have a really huge packet drop in [NS2] on queuing

>> operation.
>
> Yes. Removing skb_orphan from veth to solve this looks like a
> pretty substantial hack. A clean solution is more likely to resemble
> ethernet pause frames so we temporarily plug the virtual device.
> Although there are issues with that as currently virtual devices
> don't have queues.
>
>> In OpenVZ we have added skb_orphan into receive path (tcp_v4_rcv,
>> __udp4_lib_rcv etc) and removed one from our virtual devices. As
>> unfortunate side effect we have packet in NS2 with a socket from NS1.
>
> That also does some really nasty things to accounting. Using
> the macvlan devices solve all of this rather neatly and with
> higher performance.

veth is not mainly affected, if it is connected to a bridge there will
be no problem. You are talking about decision to switch/choose VE on
level 2, I am told about layer 3, i.e. at the moment of routing/based on
IP. So,

I do not understand how this will help me in my usecase. The macvlan
device transmit the packet into real device. OK. But it does not help if
I want to setup router in one namespace for another.

Basically, queue stop will not help, as routing namespace can have two
interfaces, one fast and one slow. In your case we should stop accepting
the message based on the slowest one?

>> So, we should have an architectural solution for this from a beginning.
>> It will be too late to hack around and change namespace lookup scheme.
>
> However what you are specifically concerned about seems to be
> using skb->sk->sk_net. Currently the only place I use this
> is in the rtnetlink code because the functions that process packets
> are not also passed a socket. Those functions we could easily
> pass in an explicit namespace or a socket parameter, and so those
> functions would not care.
>
>> I see that we should adopt a generic approach:
>> - each concrete packet belongs to a concrete namespace
>> - if function has a packet as a parameter, we should get namespace from
>> packet
>
> This approach when suggested in earlier review was pretty
> substantially shot down. Shrinking the size of struct sk_buff is
> currently an ongoing todo for the linux networking stack.
>

>> - if skb->dev is present we should get namespace as skb->dev->nd_net
>> - otherwise we should get skb->sk->sk_net. If skb->sk is NULL -> this is
>> a bug
>
> Currently killing skb->dev is a todo list item, so using it more is
> probably the wrong approach.
>
> I do agree the duplication information between fields on the skb
> and fields passed to functions is a pain. Moving more onto the
> skb seems contrary to the how the rest of the networking stack would
> like to go, so it is likely better to simply remove skb->dev and pass
> an explicit dev parameter instead. Please note that parameters passed
> explicitly will live in registers and will be quite fast.

how much registers do you have on i386 for this? the call found in my
original letter already has 6. Additionally, we can be shot by embedded
people arguing about 1 more argument and additional not needed for them
code.... And namespace code becomes unremovable one by usual ifdef way.

Regards,
 Den

Subject: Re: [PATCH 1/5] net: Modify all rtnetlink methods to only work in the initial namespace
Posted by ebiederm on Thu, 11 Oct 2007 17:08:51 GMT
View Forum Message <> Reply to Message

"Denis V. Lunev" <den@sw.ru> writes:

>> Grr. That last sentence should have been I do not see a need for the more
>> fundamental change you seem to be advocating.
>
> why?

Because I do not see the need. There are certainly details that can be
improved, but you seem to be talking about ripping everything out, ignoring
the reviews and the acceptance that has happened and starting from
scratch all over again.

I don't see the need to start again from scratch.

> veth is not mainly affected, if it is connected to a bridge there will
> be no problem. You are talking about decision to switch/choose VE on
> level 2, I am told about layer 3, i.e. at the moment of routing/based on
> IP. So,
>
> I do not understand how this will help me in my usecase. The macvlan

> device transmit the packet into real device. OK. But it does not help if
> I want to setup router in one namespace for another.
>
> Basically, queue stop will not help, as routing namespace can have two
> interfaces, one fast and one slow. In your case we should stop accepting
> the message based on the slowest one?

You are proposing a solution that only works for namespaces, while the problem continues to exist in all other routing cases. Solving this in a namespace specific way seems to reduce the value of using namespaces for testing weird aspects of the networking stack. And except for OpenVZ this is not a common case for anyone yet, so it seems a seriously premature optimization. Given that it could only be one path having something like ECN for UDP to slow the transmitter down would be nice.

The point of the macvlan example is that I believe that will be our common case when things network namespaces are deployed in the real world. Both because macvlans are easier to setup then routing or bridging, and because they perform better.

Further this problem feels like an application bug to me. Going from a fast to a slow network when you are saturating the fast network with UDP packets will cause packets to be dropped. If this is a problem the application should slow down it's packet transmission rate or at least wait for an ACK periodically.

> how much registers do you have on i386 for this? the call found in my
> original letter already has 6. Additionally, we can be shot by embedded
> people arguing about 1 more argument and additional not needed for them
> code.... And namespace code becomes unremovable one by usual ifdef
> way.

Mucking with data on the skb has all kinds of potential for slowing down the fast path of the networking stack, and if the networking stack is slow the embedded people won't even use it. So as much as I am sympathetic to preventing code size growth, maintenance and performance are higher priorities. Especially when you are advocating growth in the fast path, while I am only advocating growth in the slow path.

As for i386 stack accesses are optimized almost as well as registers, so function parameters are still not a bad deal.

>From the first couple of rounds of review the clear message was: Maintainable and comprehensible code and don't touch the fast path.

If to achieve the above I have to use a solution that distresses
the embedded people my apologies.

Eric

Subject: Re: [PATCH 1/5] net: Modify all rtnetlink methods to only work in the initial namespace
Posted by dlunev on Thu, 11 Oct 2007 17:26:55 GMT
View Forum Message <> Reply to Message

> If to achieve the above I have to use a solution that distresses
> the embedded people my apologies.

Sounds good for me :)

Regards,
Den
