LINUX PLUMBERS CONFERENCE 2024

Per Netns RTNL

Kuniyuki Iwashima

Kernel Development Engineer
Amazon Web Services

Agenda

What is RTNL ?

Recent Updates for RTNL

Per Netns RTNL

What is RTNL ?

rtnetlink

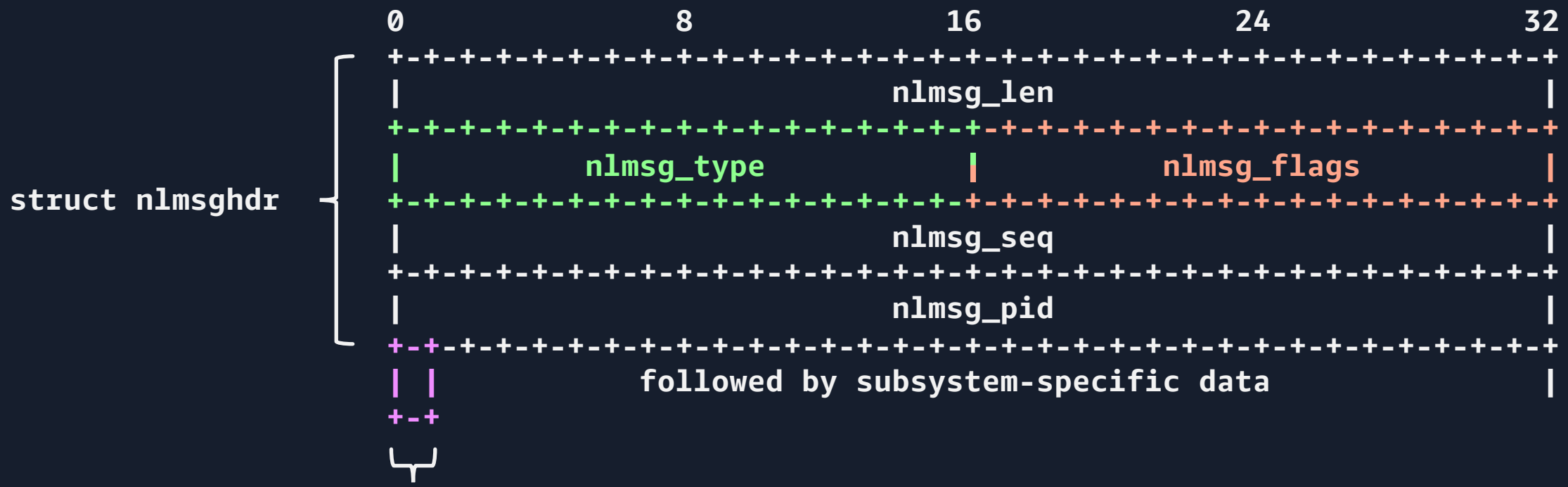
- A kind of Netlink family (**R**outing **T**able **Netlink**)

```
$ man 7 rtnetlink
rtnetlink_socket = socket(AF_NETLINK, int socket_type, NETLINK_ROUTE);
```

- Used to communicate between various subsystems
  - Network interfaces
  - IPv4 / IPv6 addresses
  - Routes
  - Neighbour entries
  - Tunnels
  - Qdisc
  - MPLS
  - MCTP

  ... and more

rtnetlink message




```
((struct rtgenmsg *)NLMSG_DATA(nlmsghdr))->rtgen_family
```

rtnetlink message

rtgen_family

- PF_UNSPEC
- PF_INET
- PF_INET6
- PF_BRIDGE

  ...

nlmsg_type

- RTM_NEW* (4n)
- RTM_DEL* (4n + 1)
- RTM_GET* (4n + 2)
- RTM_SET* (4n + 3)

  n >= 4

nlmsg_flags

- NLM_F_REQUEST
- NLM_F_ACK
- NLM_F_DUMP
- NLM_F_CREATE

  ...

rtnetlink message

rtgen_family

- PF_UNSPEC
- PF_INET
- PF_INET6
- PF_BRIDGE

  ...

nlmsg_type

- RTM_NEW* (4n)
- RTM_DEL* (4n + 1)
- RTM_GET* (4n + 2)
- RTM_SET* (4n + 3)

  n >= 4

nlmsg_flags

- NLM_F_REQUEST
- NLM_F_ACK
- NLM_F_DUMP
- NLM_F_CREATE

  ...

e.g.)

```
$ ip link add ... => (PF_UNSPEC, RTM_NEWLINK, NLM_F_REQUEST | NLM_F_CREATE)
$ ip -6 addr show => (PF_INET6 , RTM_GETADDR, NLM_F_REQUEST | NLM_F_DUMP)
```

rtnetlink message handler

- Registered by **`rtnl_register()`** or **`rtnl_register_module()`**

```
void rtnl_register(int protocol, int msgtype,
                   rtnl_doit_func doit, rtnl_dumpit_func dumpit,
                   unsigned int flags)
```

- Called based on **`rt_genfamily`**, **`nlmsg_type`**, and **`nlmsg_flag`**

pseudocode

```
rtnl_msg_handlers[protocol][msgtype - RTM_BASE] = (strcut rtnl_link *)&{
        .doit   = doit,         /* for each request */
        .dumpit = dumpit,       /* for each dump request (RTM_GET* && NLM_F_DUMP) */
        .flags  = flags,
};
```

How is doit() called ?

```
static int rtnetlink_rcv_msg(struct sk_buff *skb, struct nlmsghdr *nlh,
                             struct netlink_ext_ack *extack)
{
...
        type = nlh->nlmsg_type - RTM_BASE;
        family = ((struct rtgenmsg *)nlmsg_data(nlh))->rtgen_family;
...
        rtnl_lock();
        link = rtnl_get_link(family, type);
        if (link && link->doit)
                err = link->doit(skb, nlh, extack);
        rtnl_unlock();
...
```

How is doit() called ?

```
static int rtnetlink_rcv_msg(struct sk_buff *skb, struct nlmsghdr *nlh,
                             struct netlink_ext_ack *extack)
{
...
        type = nlh->nlmsg_type - RTM_BASE;
        family = ((struct rtgenmsg *)nlmsg_data(nlh))->rtgen_family;
...
        rtnl_lock();
        link = rtnl_get_link(family, type);
        if (link && link->doit)
                err = link->doit(skb, nlh, extack);
        rtnl_unlock();
...
```

→ Called under **rtnl_lock()**, so-called **RTNL**

What is RTNL ?

“Big Kernel Lock” for the networking slow path

```
DEFINE_MUTEX(rtnl_mutex);

void rtnl_lock(void)
{
        mutex_lock(&rtnl_mutex);
}
```

- Serialise*d all* rtnetlink requests

rtnetlink message handler flags

- Added in v4.14

```
void rtnl_register(int protocol, int msgtype,
                   rtnl_doit_func doit, rtnl_dumpit_func dumpit,
                   unsigned int flags)
```

- Control how each handler is called

pseudocode

```
rtnl_msg_handlers[protocol][msgtype - RTM_BASE] = (strcut rtnl_link *)&{
        .doit   = doit,         /* for each request */
        .dumpit = dumpit,       /* for each dump request (RTM_GET* && NLM_F_DUMP) */
        .flags  = flags,
};
```

RTNL_FLAG_DOIT_UNLOCKED

- Added in v4.14

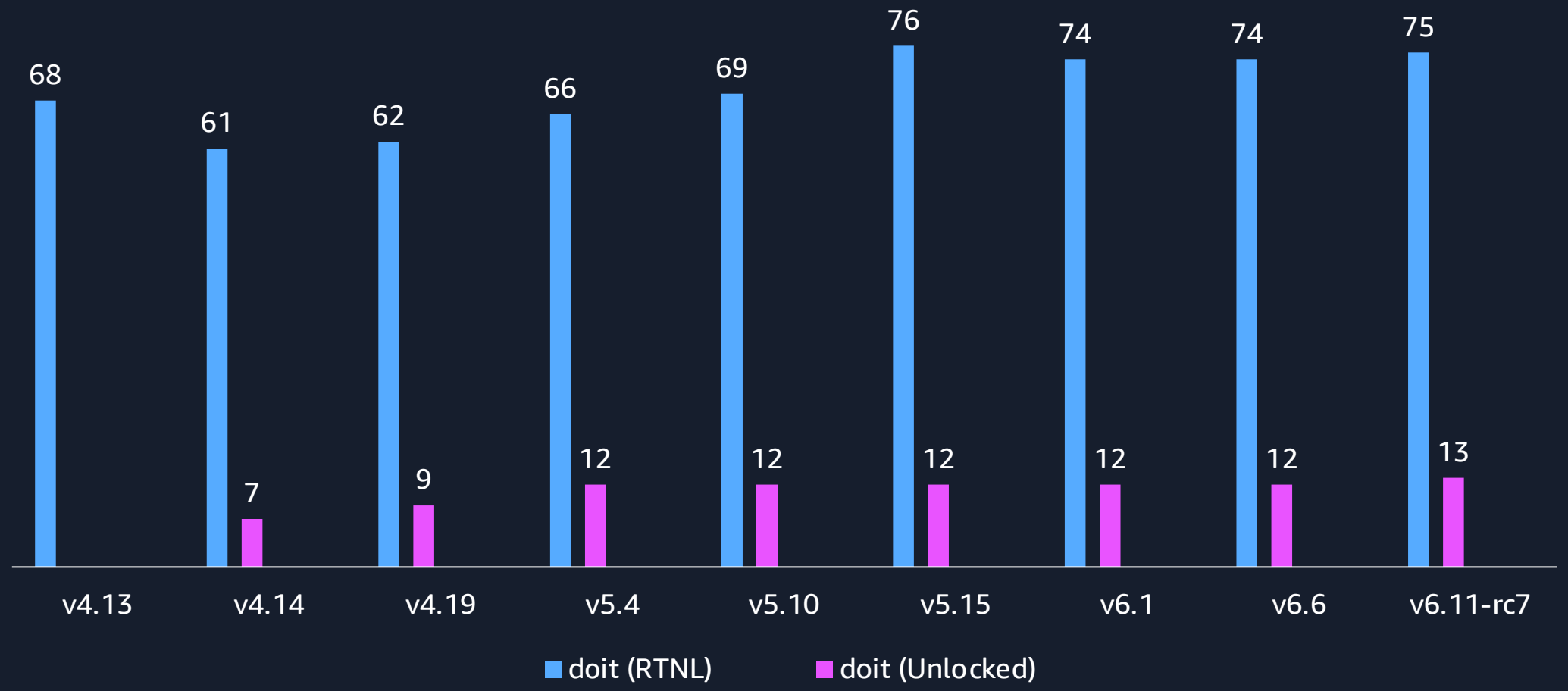
```
enum rtnl_link_flags {
        RTNL_FLAG_DOIT_UNLOCKED       = BIT(0),
};
```

- Used to call **doit()** without RTNL
  - Push RTNL down to **doit()**
  - Convert **doit()** to RCU

How is doit() called without RTNL ?

```
static int rtnetlink_rcv_msg(struct sk_buff *skb, struct nlmsghdr *nlh,
                             struct netlink_ext_ack *extack)
{
...
        rcu_read_lock();
        link = rtnl_get_link(family, type);
        owner = link->owner;
        if (!try_module_get(owner)) {...}
...
        if (link->flags & RTNL_FLAG_DOIT_UNLOCKED) {
                doit = link->doit;
                rcu_read_unlock();
                if (doit)
                        err = doit(skb, nlh, extack);
                module_put(owner);
                return err;
        }
```

Conversion of `doit()`



Recent Updates for RTNL

Lower RTNL pressure

- `RTNL_FLAG_DUMP_UNLOCKED`
- `exit_batch_rtnl()`

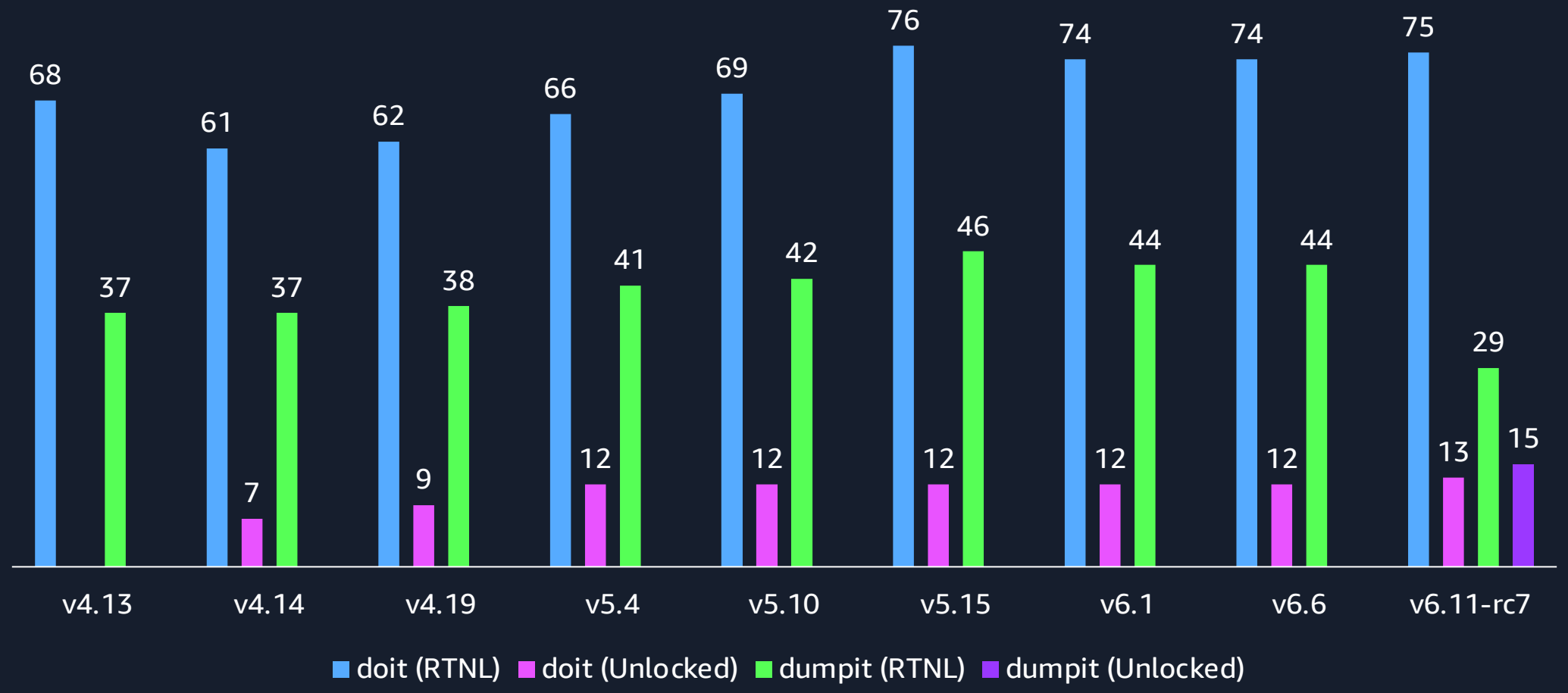
RTNL_FLAG_DUMP_UNLOCKED

- Added in v6.9

```
struct rtnl_link_flags {
        RTNL_FLAG_DOIT_UNLOCKED         = BIT(0),
...
        RTNL_FLAG_DUMP_UNLOCKED         = BIT(2),
};
```

- Used to call **`dumpit()`** without RTNL
  - Convert **`dumpit()`** to RCU

Conversion of `doit()` and `dumpit()`



exit_batch_rtnl()

- Added in v6.9

```
struct pernet_operations {
...
        /* Following method is called with RTNL held. */
        void (*exit_batch_rtnl)(struct list_head *net_exit_list,
                                struct list_head *dev_kill_list);
```

- Called with a list of (dying) netns and a list to queue dying network interfaces

How to remove network interface

1. Acquire RTNL
2. Queue network interfaces to a list
3. Pass the list to **`unregister_netdevice_many()`**
   - Call **`flush_all_backlogs()`**
   - Call **`synchronize_net()`** twice
4. Release RTNL
   - **`rtnl_unlock()`** waits until each network interface's refcnt drop to 1

Common pattern in exit_batch()

Bonding, Bridge, Geneve, IPIP, GRE, etc

```
static void __net_exit geneve_exit_batch_net(struct list_head *net_list)
{
        struct net *net;
        LIST_HEAD(list);

        rtnl_lock();
        list_for_each_entry(net, net_list, exit_list) {
                for_each_netdev_safe(net, dev, aux) {
                        if (dev->rtnl_link_ops == &geneve_link_ops)
                                unregister_netdevice_queue(dev, &list);
                }
        }
        unregister_netdevice_many(&list);
        rtnl_unlock();
```

How is `exit_batch_rtnl()` called ?

Factorise `rtnl_lock()` and `unregister_netdevice_many()`

```
static void cleanup_net(struct work_struct *work)
{
...
        rtnl_lock();
        list_for_each_entry_reverse(ops, &pernet_list, list) {
        if (ops->exit_batch_rtnl)
                ops->exit_batch_rtnl(&net_exit_list, &dev_kill_list);
        }
        unregister_netdevice_many(&dev_kill_list);
        rtnl_unlock();
...
```

exit_batch_rtnl()

- Added in v6.9

```
struct pernet_operations {
...
        /* Following method is called with RTNL held. */
        void (*exit_batch_rtnl)(struct list_head *net_exit_list,
                                struct list_head *dev_kill_list);
```

- Called with a list of (dying) netns and a list to queue dying network interfaces
- Dismantle netns faster
  - Reduce RTNL lock dance
  - Remove network interfaces in a batch

Pet Netns RTNL

RTNL vs Unlocked

RTNL : 104

Unlocked : 27

doit() : 87

dumpit() : 44

75

doit (RTNL)

12

doit (Unlocked)

15

dumpit (Unlocked)

29

dumpit (RTNL)

Unlocked dumpit()

protocol	msgtype	dumpit()	command
PF_UNSPEC	RTM_GETNEIGH	neigh_dump_info	ip neigh show
PF_UNSPEC	RTM_GETRULE	fib_nl_dumprule	ip rule show
PF_UNSPEC	RTM_GETNSID	rtnl_net_dumpid	ip netns list-id
PF_INET	RTM_GETADDR	inet_dump_ifaddr	ip -4 addr show
PF_INET	RTM_GETROUTE	inet_dump_fib	ip -4 route show
PF_INET	RTM_GETNETCONF	inet_netconf_dump_devconf	ip -4 netconf show
PF_INET6	RTM_GETLINK	inet6_dump_ifinfo	
PF_INET6	RTM_GETADDR	inet6_dump_ifaddr	ip -6 addr show
PF_INET6	RTM_GETROUTE	inet6_dump_fib	ip -6 route show
PF_INET6	RTM_GETMULTICAST	inet6_dump_ifmcaddr	
PF_INET6	RTM_GETANYCAST	inet6_dump_ifacaddr	
PF_INET6	RTM_GETADDRLABEL	ip6addrlbl_dump	ip addrlabel
PF_INET6	RTM_GETNETCONF	inet6_netconf_dump_devconf	ip -6 netconf show
PF_MPLS	RTM_GETNETCONF	mpls_netconf_dump_devconf	
PF_PHONET	RTM_GETROUTE	route_dumpit	

Unlocked doit()

protocol	msgtype	doit()	command
PF_UNSPEC	RTM_NEWTFILTER	tc_new_tfilter	tc filter add ...
PF_UNSPEC	RTM_DELTFILTER	tc_del_tfilter	tc filter del ...
PF_UNSPEC	RTM_GETTFILTER	tc_get_tfilter	tc filter get ...
PF_UNSPEC	RTM_NEWNSID	rtnl_net_newid	ip netns set ...
PF_UNSPEC	RTM_GETNSID	rtnl_net_getid	ip netns list-id ...
PF_INET	RTM_GETROUTE	inet_rtm_getroute	ip -4 route get ...
PF_INET	RTM_GETNETCONF	inet_netconf_get_devconf	
PF_INET6	RTM_GETADDR	inet6_rtm_getaddr	ip -6 addr show ...
PF_INET6	RTM_GETROUTE	inet6_rtm_getroute	ip -6 route get ...
PF_INET6	RTM_NEWADDRLABEL	ip6addrlbl_newdel	ip addrlabel add ...
PF_INET6	RTM_DELADDRLABEL	ip6addrlbl_newdel	ip addrlabel del ...
PF_INET6	RTM_GETADDRLABEL	ip6addrlbl_get	
PF_INET6	RTM_GETNETCONF	inet6_netconf_get_devconf	

One of our services

- Create 2K netns and ~10 network interfaces in each netns
  - `RTM_NEWLINK`
  - `RTM_NEWADDR`
  - `RTM_NEWROUTE`
  - `RTM_NEW...`
- Require 10+ minutes for setup
- Need more granular locking

Per Netns RTNL

Small RTNL mutex to freeze a single netns

```
void rtnl_net_lock(struct net *net)
{
        mutex_lock(&net->rtnl_mutex);
}
```

What is RTNL ?

“Big Kernel Lock” for the networking slow path

```
DEFINE_MUTEX(rtnl_mutex);

void rtnl_lock(void)
{
        mutex_lock(&rtnl_mutex);
}
```

- Serialise most *writer* rtnetlink requests
- Serialise similar uAPI (ioctl(), sysfs)
- and … ???

How big is RTNL ?

```
$ grep -rnI "rtnl_lock();" | wc -l
```

How big is RTNL ?

```
$ grep -rnI "rtnl_lock();" | wc -l
847
```

How big is RTNL ?

```
$ grep -rnI "rtnl_lock();" | wc -l
847

$ grep -rnI "rtnl_lock();" net/ | wc -l
298

$ grep -rnI "rtnl_lock();" drivers/ | wc -l
532

$ grep -rnI "ASSERT_RTNL();" | wc -l
501

$ grep -rnI "rtnl_dereference" | wc -l
655

$ grep -rnI rtnl | wc -l
5685
```

Per Netns RTNL

Small RTNL mutex to freeze a single netns

```
void rtnl_net_lock(struct net *net)
{
        mutex_lock(&net->rtnl_mutex);
}
```

- How to replace 800+ RTNL instances ?
- How many netns need to be frozen for each path ?

How to replace RTNL ?

For all RTNL scopes :

1. Replace **`rtnl_lock()`** with **`rtnl_net_lock()`**
2. Replace RTNL helpers with per-netns alternatives

```
void rtnl_net_lock(struct net *net)
{
        rtnl_lock();
#ifdef CONFIG_DEBUG_NET_SMALL_RTNL
        mutex_lock(&net->rtnl_mutex);
#endif
}
```

Once completed, remove **`rtnl_lock()`** and the config guard in **`rtnl_net_lock()`**

How to replace RTNL ?

Before conversion

```
rtnl_lock();

ASSERT_RTNL();

ptr = rtnl_dereference(...);

rtnl_unlock();
```

During conversion

```
rtnl_net_lock(net);

ASSERT_RTNL_NET(net);

ptr = rtnl_net_dereference(net, ...);

rtnl_net_unlock(net);
```

No functional change without `CONFIG_DEBUG_NET_SMALL_RTNL`

New flags for per-netns `doit()` conversion

- Indicate that **`doit()`** is converted or under conversion
- Call **`doit()`** without RTNL

```
enum rtnl_link_flags {
        RTNL_FLAG_DOIT_UNLOCKED         = BIT(0),
#define RTNL_FLAG_DOIT_PERNET           RTNL_FLAG_DOIT_UNLOCKED
#define RTNL_FLAG_DOIT_PERNET_WIP       RTNL_FLAG_DOIT_UNLOCKED
        RTNL_FLAG_BULK_DEL_SUPPORTED    = BIT(1),
        RTNL_FLAG_DUMP_UNLOCKED         = BIT(2),
        RTNL_FLAG_DUMP_SPLIT_NLM_DONE   = BIT(3),
};
```

Convert `doit()` : Single netns

Most **`doit()`**s operate on a single netns

1. Add a lock for global data
   or
   Convert it to per-netns
2. (Optional) Move message validation forward
3. Add **`RTNL_FLAG_DOIT_PERNET`**
   and
   Push RTNL down as **`rtnl_net_lock()`**
4. Replace RTNL helpers

Convert RTM_NEWADDR

1. Namespacify the IPv4 addr hash table (IPv6 one is already namespacified)

```
static __net_init int devinet_init_net(struct net *net)
{
        net->ipv4.inet_addr_lst = kmalloc_array(IN4_ADDR_HSIZE,
                                                sizeof(struct hlist_head),
                                                GFP_KERNEL);
```

```
static void inet_hash_insert(struct net *net, struct in_ifaddr *ifa)
{
        u32 hash = inet_addr_hash(net, ifa->ifa_local);

        ASSERT_RTNL_NET(net);
        hlist_add_head_rcu(&ifa->hash_node, &net->ipv4.inet_addr_lst[hash]);
}
```

Convert RTM_NEWADDR

1. Namespacify the IPv4 addr hash table (IPv6 one is already namespacified)
2. Namespacify the IPv4 GC

```
static __net_init int devinet_init_net(struct net *net)
{
...
        INIT_DEFERRABLE_WORK(&net->ipv4.addr_chk_work, check_lifetime);
...
```

Convert RTM_NEWADDR

1. Namespacify the IPv4 addr hash table (IPv6 one is already namespacified)
2. Namespacify the IPv4 GC
3. Place rtnl_net_lock() and push RTNL down

```
static int inet_rtm_newaddr(struct sk_buff *skb, struct nlmsghdr *nlh, ...)
{
...
        rtnl_net_lock(net);
        in_dev = __in_dev_get_rtnl_net(dev);
...
        rtnl_net_unlock(net);
        return 0;
}
```

Convert `doit()` : Multiple netns

Some **`doit()`**s operate on multiple netns

- **`RTM_NEWLINK`**
- **`RTM_DELLINK`**
- **`RTM_SETLINK`**

Need clear locking order to avoid AB-BA deadlock

Multiple netns : Locking order

```
static __net_init void preinit_net(struct net *net, struct user_namespace *user_ns)
{
...
        lock_set_cmp_fn(&net->rtnl_mutex, rtnl_net_lock_cmp_fn, NULL);
}
```

```
int rtnl_net_lock_cmp_fn(const struct lockdep_map *a, const struct lockdep_map *b)
{
        const struct net *net_a, *net_b;

        net_a = container_of(a, struct net, rtnl_mutex.dep_map);
        net_b = container_of(b, struct net, rtnl_mutex.dep_map);

        return rtnl_net_cmp_locks(net_a, net_b);
}
```

Multiple netns : Locking order

1. init_net is always first
2. Address ascending order

```
static int rtnl_net_cmp_locks(const struct net *net_a, const struct net *net_b)
{
        if (net_eq(net_a, net_b))
                return 0;

        if (net_eq(net_a, &init_net))
                return -1;
        if (net_eq(net_b, &init_net))
                return 1;

        return net_a < net_b ? -1 : 1;
}
```

Multiple netns : Locking order

1. init_net is always first
2. Address ascending order

```
static void __net_exit default_device_exit_batch(struct list_head *net_list)
{
...
        rtnl_net_lock(&init_net);
        list_for_each_entry(net, net_list, exit_list) {
                __rtnl_net_lock(net);
                default_device_exit_net(net); /* Move network interfaces to init_net */
                __rtnl_net_unlock(net);
        }
        rtnl_net_unlock(&init_net);
...
```

Multiple netns : Helper functions

```
#define RTNL_NETS_MAX  ???
struct rtnl_nets {
        struct net *net[RTNL_NETS_MAX];
        unsigned char len;
};
```

```
static void rtnl_nets_lock(struct rtnl_nets *rtnl_nets)
{
        int i;

        for (i = 0; i < rtnl_nets->len; i++)
                rtnl_net_lock(rtnl_nets->net[i]);
}
```

Multiple netns : Helper functions

```
#define RTNL_NETS_MAX  ???
struct rtnl_nets {
        struct net *net[RTNL_NETS_MAX];
        unsigned char len;
};
```

```
        rtnl_nets_init(&rtnl_nets);                 /* memset() */

        tgt_net = rtnl_link_get_net_capable(...);   /* call get_net() */
        rtnl_nets_add(&rtnl_nets, tgt_net);         /* Add and sort net with cmp_fn */

        rtnl_nets_lock(&rtnl_nets);                 /* lock from net[0] to len - 1 */
        rtnl_nets_unlock(&rtnl_nets);

        rtnl_nets_destroy(&rtnl_nets);              /* call put_net() */
```

Multiple netns : Helper functions

```
int rtnl_nets_add(struct rtnl_nets *rtnl_nets, struct net *net)
{
        if (WARN_ON_ONCE(rtnl_nets->len == ARRAY_SIZE(rtnl_nets->net)))
                return -ENOBUF;

        for (int i = 0; i < rtnl_nets->len; i++) {
                switch (rtnl_net_cmp_locks(net, rtnl_nets->net[i])) {
                case 0:
                        return -EEXIST;
                case -1:
                        swap(net, rtnl_nets->net[i]); /* Bubble sort */
                }
        }

        rtnl_nets->net[rtnl_nets->len++] = net;
        return 0;
}
```

RTM_NEWLINK

4 netns attributes

- IFLA_NET_NS_PID
- IFLA_NET_NS_FD
- IFLA_TARGET_NETNSID
- IFLA_LINK_NETNSID

RTM_NEWLINK : w/o IFLA_LINK_NETNSID

dest_net

- sock_net(skb->sk)
- IFLA_NET_NS_PID
- IFLA_NET_NS_FD
- IFLA_TARGET_NETNSID

net

- sock_net(skb->sk)

```
dev =
 rtnl_create_link(dest_net)

                  ops->newlink(net, dev)
```

RTM_NEWLINK : w/ IFLA_LINK_NETNSID

dest_net

- sock_net(skb->sk)
- IFLA_NET_NS_PID
- IFLA_NET_NS_FD
- IFLA_TARGET_NETNSID

link_net

- IFLA_LINK_NETNSID

```
dev =
 rtnl_create_link(link_net)

ops->newlink(link_net, dev)

dev_change_net_namespace(dev, dest_net)
```

`RTM_NEWLINK` : veth, vxcan, and netkit

`rtnl_link_ops.newlink()` creates a peer device

- veth (`VETH_INFO_PEER`)
- vxcan (`VXCAN_INFO_PEER`)
- netkit (`IFLA_NETKIT_PEER_INFO`)

whose netns could be changed by

- `IFLA_NET_NS_PID`
- `IFLA_NET_NS_FD`
- `IFLA_TARGET_NETNSID`

RTM_NEWLINK : veth, vxcan, and netkit

dest_net

- sock_net(skb->sk)
- IFLA_NET_NS_PID
- IFLA_NET_NS_FD
- IFLA_TARGET_NETNSID

link_net

- IFLA_LINK_NETNSID

```
dev =
 rtnl_create_link(link_net)

ops->newlink(link_net, dev)

dev_change_net_namespace(dev, dest_net)
```

peer_net

- 1st arg of newlink()
- VETH_INFO_PEER
- VXCAN_INFO_PEER
- IFLA_NETKIT_PEER_INFO
  - IFLA_NET_NS_PID
  - IFLA_NET_NS_FD
  - IFLA_TARGET_NETNSID

RTM_NEWLINK

Freeze 3 netns at most

- `IFLA_NET_NS_PID`, `IFLA_NET_NS_FD`, `IFLA_TARGET_NETNSID`
- `IFLA_LINK_NETNSID`
- Peer netns for veth, vxcan, netkit

Convert `RTM_NEWLINK`

For the first two netns

- Need to prefetch netns before **`rtnl_nets_lock()`**
- Move message validation forward

```
        tgt_net = rtnl_link_get_net_capable(skb, sock_net(skb->sk), tb, CAP_NET_ADMIN);
        rtnl_nets_add(&rtnl_nets, tgt_net);
...
        if (tb[IFLA_LINK_NETNSID]) {
                link_net = get_net_ns_by_id(tgt_net, nla_get_s32(tb[IFLA_LINK_NETNSID]));
                rtnl_nets_add(&rtnl_nets, link_net);
        }
...
        rtnl_nets_lock(&rtnl_nets);
```

Convert RTM_NEWLINK

For the peer netns (veth, vxcan, netkit)

- Need a new ops method to fetch the peer netns
- Need to guarantee ops alive without RTNL (module-unload-tolerant)

```
        ops = rtnl_link_ops_get(kind);        /* Acquire SRCU */
        if (ops && ops->get_peer_net)
                /* Call rtnl_nets_add() internally */
                ret = ops->get_peer_net(&rtnl_nets, tb, data, extack);

        rtnl_nets_lock(&rtnl_nets);
...
        rtnl_nets_unlock(&rtnl_nets);

        rtnl_link_ops_put(ops);               /* Release SRCU */
```

Convert RTM_NEWLINK

```
void __rtnl_link_unregister(struct rtnl_link_ops *ops)
{
        struct net *net;

        spin_lock(&link_ops_lock);
        list_del_rcu(&ops->list);
        spin_unlock(&link_ops_lock);                    /* No one can see ops now. */

        synchronize_srcu(&ops->srcu);                   /* Wait RTM_NEWLINK to complete. */

        for_each_net(net)
                __rtnl_kill_links(net, ops);
}
```

Convert `RTM_NEWLINK`

1. Move message validation forward
   - Prefetch netns attributes
   - Add **`ops->get_peer_net()`** for veth, vxcan, and netkit
2. Protect link_ops with SRCU
3. Add **`RTNL_FLAG_DOIT_PERNET`**
   and
   Push RTNL down as **`rtnl_net_lock()`**

RTM_DELLINK

Remove a network interface in either of

- `sock_net(skb->sk)`
- `IFLA_TARGET_NETNSID`

What if the dev has peer/child devices ?

- veth, vxcan, netkit
- macvlan, ipvlan, etc

Does prefetching netns in `rtnl_delink()` work ?

RTM_DELLINK : macvlan

```
static int macvlan_device_event(struct notifier_block *unused,
                                unsigned long event, void *ptr)
{
...
        switch (event) {
...
        case NETDEV_UNREGISTER:
...
                /* How many netns touched here ?
                 * Can’t add rtnl_net_lock() in netdev notifier !
                 */
                list_for_each_entry_safe(vlan, next, &port->vlans, list)
                        vlan->dev->rtnl_link_ops->dellink(vlan->dev);

                unregister_netdevice_flush();
                break;
...
```

Convert RTM_DELLINK

Unregister network interfaces per netns

- Queue network interfaces to a per netns list
- Trigger per-net unregistration work

```
void unregister_netdevice_queue(struct net_device *dev)
{
        llist_add(&dev->unreg_list, &net->dev_unreg_head);

        /* queue per-netns unreg work, need rework of rtnl_unlock() */
}
```

Convert RTM_DELLINK

Snapshot dying network interfaces list and unregister them

```
static void unregister_netdevice_net_work_fn(struct work_struct *work)
{
        struct net *net = container_of(work, struct net, dev_unreg_work);
        struct llist_node *unreg_list;

        rtnl_net_lock(net);

        unreg_list = llist_del_all(&net->dev_unreg_head,);

        /* Do dev unregistraion here. */

        rtnl_net_unlock(net);
}
```

Convert `RTM_DELLINK`

Until **`rtnl_unlock()`** can be removed

- Manage netns list pending for device unregistration
- Splice the list in **`unregister_netdevice_many_notify()`**

```
void unregister_netdevice_queue(struct net_device *dev)
{
        llist_add(&dev->unreg_list, &net->dev_unreg_head);

        /* queue per-netns unreg work, need rework of rtnl_unlock() */

        if (list_empty(&net->unreg_list))
                list_add(&net->unreg_list, &net_unreg_list);
}
```

Convert `RTM_DELLINK`

1. Queue dying network interfaces to a per-netns list
2. Splice the list in **`unregister_netdevice_many_notify()`**
3. Remove **`kill_list`** from **`ops->devlink()`** and **`pernet->exit_batch_rtnl()`**
4. Push RTNL down as **`rtnl_net_lock()`**

But netns dismantle will slow down by **un-batching**

- **`flush_backlog()`**
- **`synchronize_net()`**

in **`unregister_netdevice_many_notify()`** ... This would be No-Go

RTM_SETLINK

Basically operate on the current netns

Move a network interface to another netns specified by

- `IFLA_NET_NS_PID`
- `IFLA_NET_NS_FD`
- `IFLA_TARGET_NETNSID`

Same question, what if the dev has peer/child devices ?

Convert `RTM_SETLINK`

1. Look up a network interface under RCU
2. Fetch all netns (yet another ops method needed)
3. Release RCU
4. Hold **`rtnl_nets_lock()`**
5. Look up a network interface again

What if peer/child in a new netns is added just before 5 ?

How can we avoid **`rtnl_nets_lock()`** for peer/child devices ?

Challenges

- `RTM_DELLINK` and netns dismantle
  - How to remove dev in different netns ?
  - How to batch per-netns device unregistration ?
- `RTM_SETLINK`
  - How to avoid prefetching all netns in the same group ?

Thank you!

Kuniyuki Iwashima

kuniyu@amazon.com