Subject: [PATCH 2/9] network namespaces: IPv4 routing
Posted by Andrey Savochkin on Tue, 15 Aug 2006 14:48:42 GMT
View Forum Message <> Reply to Message

Structures related to IPv4 rounting (FIB and routing cache)
are made per-namespace.

Signed-off-by: Andrey Savochkin <saw@swsoft.com>

```
 include/linux/net_ns.h   |   10 +++
 include/net/flow.h       |    3 +
 include/net/ip_fib.h     |   46 ++++++++++++----
 net/core/dev.c           |    8 ++
 net/core/fib_rules.c     |   43 +++++++++++---
 net/ipv4/Kconfig         |    4 -
 net/ipv4/fib_frontend.c  |  132 ++++++++++++++++++++++++++++++++++++++++++++++----------
 net/ipv4/fib_hash.c      |   13 +++-
 net/ipv4/fib_rules.c     |   86 +++++++++++++++++++++++++++++-----
 net/ipv4/fib_semantics.c |   99 ++++++++++++++++++++++++++++++++++++-----
 net/ipv4/route.c         |   26 ++++++++-
 11 files changed, 375 insertions(+), 95 deletions(-)

--- ./include/linux/net_ns.h.vensroute Mon Aug 14 17:18:59 2006
+++ ./include/linux/net_ns.h Mon Aug 14 19:19:14 2006
@@ -14,7 +14,17 @@ struct net_namespace {
  atomic_t  active_ref, use_ref;
  struct net_device *dev_base_p, **dev_tail_p;
  struct net_device *loopback;
+#ifndef CONFIG_IP_MULTIPLE_TABLES
+ struct fib_table *fib4_local_table, *fib4_main_table;
+#else
+ struct list_head fib_rules_ops_list;
+ struct fib_rules_ops *fib4_rules_ops;
+ struct hlist_head *fib4_tables;
+#endif
+ struct hlist_head *fib4_hash, *fib4_laddrhash;
+ unsigned  fib4_hash_size, fib4_info_cnt;
  unsigned int  hash;
+ char   destroying;
  struct work_struct destroy_work;
 };

--- ./include/net/flow.h.vensroute Mon Aug 14 17:04:04 2006
+++ ./include/net/flow.h Mon Aug 14 17:18:59 2006
@@ -79,6 +79,9 @@ struct flowi {
 #define fl_icmp_code uli_u.icmpt.code
 #define fl_ipsec_spi uli_u.spi
  __u32          secid; /* used by xfrm; see secid.txt */
```

```
+#ifdef CONFIG_NET_NS
+ struct net_namespace *net_ns;
+#endif
 } __attribute__((__aligned__(BITS_PER_LONG/8)));

 #define FLOW_DIR_IN 0
--- ./include/net/ip_fib.h.vensroute Mon Aug 14 17:04:04 2006
+++ ./include/net/ip_fib.h Tue Aug 15 11:53:22 2006
@@ -18,6 +18,7 @@

 #include <net/flow.h>
 #include <linux/seq_file.h>
+#include <linux/net_ns.h>
 #include <net/fib_rules.h>

 /* WARNING: The ordering of these elements must match ordering
@@ -171,14 +172,21 @@ struct fib_table {

 #ifndef CONFIG_IP_MULTIPLE_TABLES

-extern struct fib_table *ip_fib_local_table;
-extern struct fib_table *ip_fib_main_table;
+#ifndef CONFIG_NET_NS
+extern struct fib_table *ip_fib_local_table_static;
+extern struct fib_table *ip_fib_main_table_static;
+#define ip_fib_local_table_ns()  ip_fib_local_table_static
+#define ip_fib_main_table_ns()  ip_fib_main_table_static
+#else
+#define ip_fib_local_table_ns()  (current_net_ns->fib4_local_table)
+#define ip_fib_main_table_ns()  (current_net_ns->fib4_main_table)
+#endif

 static inline struct fib_table *fib_get_table(u32 id)
 {
  if (id != RT_TABLE_LOCAL)
-  return ip_fib_main_table;
- return ip_fib_local_table;
+  return ip_fib_main_table_ns();
+ return ip_fib_local_table_ns();
 }

 static inline struct fib_table *fib_new_table(u32 id)
@@ -188,21 +196,29 @@ static inline struct fib_table *fib_new_

 static inline int fib_lookup(const struct flowi *flp, struct fib_result *res)
 {
- if (ip_fib_local_table->tb_lookup(ip_fib_local_table, flp, res) &&
-     ip_fib_main_table->tb_lookup(ip_fib_main_table, flp, res))
```

```
+ struct fib_table *tb;
+
+ tb = ip_fib_local_table_ns();
+ if (!tb->tb_lookup(tb, flp, res))
+  return 0;
+ tb = ip_fib_main_table_ns();
+ if (tb->tb_lookup(tb, flp, res))
   return -ENETUNREACH;
  return 0;
 }

 static inline void fib_select_default(const struct flowi *flp, struct fib_result *res)
 {
+ struct fib_table *tb;
+
+ tb = ip_fib_main_table_ns();
  if (FIB_RES_GW(*res) && FIB_RES_NH(*res).nh_scope == RT_SCOPE_LINK)
-  ip_fib_main_table->tb_select_default(ip_fib_main_table, flp, res);
+  tb->tb_select_default(main_table, flp, res);
 }

 #else /* CONFIG_IP_MULTIPLE_TABLES */
-#define ip_fib_local_table fib_get_table(RT_TABLE_LOCAL)
-#define ip_fib_main_table fib_get_table(RT_TABLE_MAIN)
+#define ip_fib_local_table_ns() fib_get_table(RT_TABLE_LOCAL)
+#define ip_fib_main_table_ns() fib_get_table(RT_TABLE_MAIN)

 extern int fib_lookup(struct flowi *flp, struct fib_result *res);

@@ -214,6 +230,10 @@ extern void fib_select_default(const str

 /* Exported by fib_frontend.c */
 extern void  ip_fib_init(void);
+#ifdef CONFIG_NET_NS
+extern int ip_fib_struct_init(void);
+extern void ip_fib_struct_cleanup(void);
+#endif
 extern int inet_rtm_delroute(struct sk_buff *skb, struct nlmsghdr* nlh, void *arg);
 extern int inet_rtm_newroute(struct sk_buff *skb, struct nlmsghdr* nlh, void *arg);
 extern int inet_rtm_getroute(struct sk_buff *skb, struct nlmsghdr* nlh, void *arg);
@@ -231,6 +251,9 @@ extern int fib_sync_up(struct net_device
 extern int fib_convert_rtentry(int cmd, struct nlmsghdr *nl, struct rtmsg *rtm,
          struct kern_rta *rta, struct rtentry *r);
 extern u32  __fib_res_prefsrc(struct fib_result *res);
+#ifdef CONFIG_NET_NS
+extern void fib_hashtable_destroy(void);
+#endif
```

```
 /* Exported by fib_hash.c */
 extern struct fib_table *fib_hash_init(u32 id);
@@ -238,7 +261,10 @@ extern struct fib_table *fib_hash_init(u
 #ifdef CONFIG_IP_MULTIPLE_TABLES
 extern int fib4_rules_dump(struct sk_buff *skb, struct netlink_callback *cb);

-extern void __init fib4_rules_init(void);
+extern int fib4_rules_init(void);
+#ifdef CONFIG_NET_NS
+extern void fib4_rules_cleanup(void);
+#endif

 #ifdef CONFIG_NET_CLS_ROUTE
 extern u32 fib_rules_tclass(struct fib_result *res);
--- ./net/core/dev.c.vensroute Mon Aug 14 17:18:59 2006
+++ ./net/core/dev.c Mon Aug 14 17:18:59 2006
@@ -3548,6 +3548,8 @@ static int __init netdev_dma_register(vo
 #endif /* CONFIG_NET_DMA */

 #ifdef CONFIG_NET_NS
+#include <net/ip_fib.h>
+
 struct net_namespace init_net_ns = {
  .active_ref = ATOMIC_INIT(2),
     /* one for init_task->net_context,
@@ -3588,6 +3590,9 @@ int net_ns_start(void)
  task = current;
  orig_ns = task->net_context;
  task->net_context = ns;
+ INIT_LIST_HEAD(&ns->fib_rules_ops_list);
+ if (ip_fib_struct_init())
+  goto out_fib4;
  err = register_netdev(dev);
  if (err)
   goto out_register;
@@ -3595,6 +3600,8 @@ int net_ns_start(void)
  return 0;

 out_register:
+ ip_fib_struct_cleanup();
+out_fib4:
  dev->destructor(dev);
  task->net_context = orig_ns;
  BUG_ON(atomic_read(&ns->active_ref) != 1);
@@ -3619,6 +3626,7 @@ static void net_ns_destroy(void *data)
   pop_net_ns(orig_ns);
   return;
  }
```

```
+ ip_fib_struct_cleanup();
  pop_net_ns(orig_ns);
  kfree(ns);
 }
--- ./net/core/fib_rules.c.vensroute Mon Aug 14 17:04:05 2006
+++ ./net/core/fib_rules.c Mon Aug 14 17:18:59 2006
@@ -11,9 +11,15 @@
 #include <linux/types.h>
 #include <linux/kernel.h>
 #include <linux/list.h>
+#include <linux/net_ns.h>
 #include <net/fib_rules.h>

-static LIST_HEAD(rules_ops);
+#ifndef CONFIG_NET_NS
+static struct list_head rules_ops_static;
+#define rules_ops_ns()  rules_ops_static
+#else
+#define rules_ops_ns()  (current_net_ns->fib_rules_ops_list)
+#endif
 static DEFINE_SPINLOCK(rules_mod_lock);

 static void notify_rule_change(int event, struct fib_rule *rule,
@@ -21,10 +27,12 @@ static void notify_rule_change(int event

 static struct fib_rules_ops *lookup_rules_ops(int family)
 {
+ struct list_head *ops_list;
  struct fib_rules_ops *ops;

+ ops_list = &rules_ops_ns();
  rcu_read_lock();
- list_for_each_entry_rcu(ops, &rules_ops, list) {
+ list_for_each_entry_rcu(ops, ops_list, list) {
   if (ops->family == family) {
    if (!try_module_get(ops->owner))
     ops = NULL;
@@ -46,6 +54,7 @@ static void rules_ops_put(struct fib_rul
 int fib_rules_register(struct fib_rules_ops *ops)
 {
  int err = -EEXIST;
+ struct list_head *ops_list;
  struct fib_rules_ops *o;

  if (ops->rule_size < sizeof(struct fib_rule))
@@ -56,12 +65,13 @@ int fib_rules_register(struct fib_rules_
      ops->action == NULL)
   return -EINVAL;
```

```
+ ops_list = &rules_ops_ns();
  spin_lock(&rules_mod_lock);
- list_for_each_entry(o, &rules_ops, list)
+ list_for_each_entry(o, ops_list, list)
   if (ops->family == o->family)
    goto errout;

- list_add_tail_rcu(&ops->list, &rules_ops);
+ list_add_tail_rcu(&ops->list, ops_list);
  err = 0;
 errout:
  spin_unlock(&rules_mod_lock);
@@ -84,10 +94,12 @@ static void cleanup_ops(struct fib_rules
 int fib_rules_unregister(struct fib_rules_ops *ops)
 {
  int err = 0;
+ struct list_head *ops_list;
  struct fib_rules_ops *o;

+ ops_list = &rules_ops_ns();
  spin_lock(&rules_mod_lock);
- list_for_each_entry(o, &rules_ops, list) {
+ list_for_each_entry(o, ops_list, list) {
   if (o == ops) {
    list_del_rcu(&o->list);
    cleanup_ops(ops);
@@ -114,6 +126,14 @@ int fib_rules_lookup(struct fib_rules_op

  rcu_read_lock();

+ err = -EINVAL;
+ if (ops->rules_list->next == NULL) {
+  if (net_ratelimit())
+   printk(" *** NULL head, ops %p, list %p\n",
+     ops, ops->rules_list);
+  goto out;
+ }
+
  list_for_each_entry_rcu(rule, ops->rules_list, list) {
   if (rule->ifindex && (rule->ifindex != fl->iif))
    continue;
@@ -127,6 +147,12 @@ int fib_rules_lookup(struct fib_rules_op
    arg->rule = rule;
    goto out;
   }
+  if (rule->list.next == NULL) {
+   if (net_ratelimit())
```

```
+    printk(" *** NULL, ops %p, list %p, item %p\n",
+      ops, ops->rules_list, rule);
+   goto out;
+  }
  }

  err = -ENETUNREACH;
@@ -382,19 +408,21 @@ static int fib_rules_event(struct notifi
        void *ptr)
 {
  struct net_device *dev = ptr;
+ struct list_head *ops_list;
  struct fib_rules_ops *ops;

  ASSERT_RTNL();
  rcu_read_lock();

+ ops_list = &rules_ops_ns();
  switch (event) {
  case NETDEV_REGISTER:
-  list_for_each_entry(ops, &rules_ops, list)
+  list_for_each_entry(ops, ops_list, list)
    attach_rules(ops->rules_list, dev);
   break;

  case NETDEV_UNREGISTER:
-  list_for_each_entry(ops, &rules_ops, list)
+  list_for_each_entry(ops, ops_list, list)
    detach_rules(ops->rules_list, dev);
   break;
  }
@@ -410,6 +438,7 @@ static struct notifier_block fib_rules_n

 static int __init fib_rules_init(void)
 {
+ INIT_LIST_HEAD(&rules_ops_ns());
  return register_netdevice_notifier(&fib_rules_notifier);
 }

--- ./net/ipv4/Kconfig.vensroute Mon Aug 14 17:04:07 2006
+++ ./net/ipv4/Kconfig Mon Aug 14 17:18:59 2006
@@ -53,7 +53,7 @@ config IP_ADVANCED_ROUTER

 choice
  prompt "Choose IP: FIB lookup algorithm (choose FIB_HASH if unsure)"
- depends on IP_ADVANCED_ROUTER
+ depends on IP_ADVANCED_ROUTER && !NET_NS
  default ASK_IP_FIB_HASH
```

```
 config ASK_IP_FIB_HASH
@@ -83,7 +83,7 @@ config IP_FIB_TRIE
 endchoice

 config IP_FIB_HASH
- def_bool ASK_IP_FIB_HASH || !IP_ADVANCED_ROUTER
+ def_bool ASK_IP_FIB_HASH || !IP_ADVANCED_ROUTER || NET_NS

 config IP_MULTIPLE_TABLES
  bool "IP: policy routing"
--- ./net/ipv4/fib_frontend.c.vensroute Mon Aug 14 17:04:07 2006
+++ ./net/ipv4/fib_frontend.c Tue Aug 15 11:53:22 2006
@@ -51,18 +51,23 @@
 #define FFprint(a...) printk(KERN_DEBUG a)

 #ifndef CONFIG_IP_MULTIPLE_TABLES
-
-struct fib_table *ip_fib_local_table;
-struct fib_table *ip_fib_main_table;
-
+#ifndef CONFIG_NET_NS
+struct fib_table *ip_fib_local_table_static;
+struct fib_table *ip_fib_main_table_static;
+#endif
 #define FIB_TABLE_HASHSZ 1
-static struct hlist_head fib_table_hash[FIB_TABLE_HASHSZ];
-
 #else
-
 #define FIB_TABLE_HASHSZ 256
-static struct hlist_head fib_table_hash[FIB_TABLE_HASHSZ];
+#endif

+#ifndef CONFIG_NET_NS
+static struct hlist_head fib_table_hash_static[FIB_TABLE_HASHSZ];
+#define fib_table_hash_ns()  fib_table_hash_static
+#else
+#define fib_table_hash_ns()  (current_net_ns->fib4_tables)
+#endif
+
+#ifdef CONFIG_IP_MULTIPLE_TABLES
 struct fib_table *fib_new_table(u32 id)
 {
  struct fib_table *tb;
@@ -77,21 +82,23 @@ struct fib_table *fib_new_table(u32 id)
  if (!tb)
   return NULL;
```

```
  h = id & (FIB_TABLE_HASHSZ - 1);
- hlist_add_head_rcu(&tb->tb_hlist, &fib_table_hash[h]);
+ hlist_add_head_rcu(&tb->tb_hlist, &fib_table_hash_ns()[h]);
  return tb;
 }

 struct fib_table *fib_get_table(u32 id)
 {
  struct fib_table *tb;
+ struct hlist_head *list;
  struct hlist_node *node;
  unsigned int h;

  if (id == 0)
   id = RT_TABLE_MAIN;
  h = id & (FIB_TABLE_HASHSZ - 1);
+ list = &fib_table_hash_ns()[h];
  rcu_read_lock();
- hlist_for_each_entry_rcu(tb, node, &fib_table_hash[h], tb_hlist) {
+ hlist_for_each_entry_rcu(tb, node, list, tb_hlist) {
   if (tb->tb_id == id) {
    rcu_read_unlock();
    return tb;
@@ -106,11 +113,13 @@ static void fib_flush(void)
 {
  int flushed = 0;
  struct fib_table *tb;
+ struct hlist_head *list;
  struct hlist_node *node;
  unsigned int h;

  for (h = 0; h < FIB_TABLE_HASHSZ; h++) {
-  hlist_for_each_entry(tb, node, &fib_table_hash[h], tb_hlist)
+  list = &fib_table_hash_ns()[h];
+  hlist_for_each_entry(tb, node, list, tb_hlist)
    flushed += tb->tb_flush(tb);
  }

@@ -126,14 +135,15 @@ struct net_device * ip_dev_find(u32 addr
 {
  struct flowi fl = { .nl_u = { .ip4_u = { .daddr = addr } } };
  struct fib_result res;
+ struct fib_table *tb;
  struct net_device *dev = NULL;

 #ifdef CONFIG_IP_MULTIPLE_TABLES
  res.r = NULL;
 #endif
```

```
- if (!ip_fib_local_table ||
-     ip_fib_local_table->tb_lookup(ip_fib_local_table, &fl, &res))
+ tb = ip_fib_local_table_ns();
+ if (!tb || tb->tb_lookup(tb, &fl, &res))
   return NULL;
  if (res.type != RTN_LOCAL)
   goto out;
@@ -150,6 +160,7 @@ unsigned inet_addr_type(u32 addr)
 {
  struct flowi  fl = { .nl_u = { .ip4_u = { .daddr = addr } } };
  struct fib_result res;
+ struct fib_table *tb;
  unsigned ret = RTN_BROADCAST;

  if (ZERONET(addr) || BADCLASS(addr))
@@ -161,10 +172,10 @@ unsigned inet_addr_type(u32 addr)
  res.r = NULL;
 #endif

- if (ip_fib_local_table) {
+ tb = ip_fib_local_table_ns();
+ if (tb) {
   ret = RTN_UNICAST;
-  if (!ip_fib_local_table->tb_lookup(ip_fib_local_table,
-         &fl, &res)) {
+  if (!tb->tb_lookup(tb, &fl, &res)) {
    ret = res.type;
    fib_res_put(&res);
   }
@@ -357,6 +368,7 @@ int inet_dump_fib(struct sk_buff *skb, s
  unsigned int h, s_h;
  unsigned int e = 0, s_e;
  struct fib_table *tb;
+ struct hlist_head *list;
  struct hlist_node *node;
  int dumped = 0;

@@ -369,7 +381,8 @@ int inet_dump_fib(struct sk_buff *skb, s

  for (h = s_h; h < FIB_TABLE_HASHSZ; h++, s_e = 0) {
   e = 0;
-  hlist_for_each_entry(tb, node, &fib_table_hash[h], tb_hlist) {
+  list = &fib_table_hash_ns()[h];
+  hlist_for_each_entry(tb, node, list, tb_hlist) {
    if (e < s_e)
     goto next;
    if (dumped)
```

```
@@ -680,25 +693,92 @@ static struct notifier_block fib_netdev_
  .notifier_call =fib_netdev_event,
 };

-void __init ip_fib_init(void)
+#if !defined(CONFIG_IP_MULTIPLE_TABLES)
+
+int inline ip_fib_struct_init(void)
 {
  unsigned int i;

  for (i = 0; i < FIB_TABLE_HASHSZ; i++)
-  INIT_HLIST_HEAD(&fib_table_hash[i]);
-#ifndef CONFIG_IP_MULTIPLE_TABLES
- ip_fib_local_table = fib_hash_init(RT_TABLE_LOCAL);
- hlist_add_head_rcu(&ip_fib_local_table->tb_hlist, &fib_table_hash[0]);
- ip_fib_main_table  = fib_hash_init(RT_TABLE_MAIN);
- hlist_add_head_rcu(&ip_fib_main_table->tb_hlist, &fib_table_hash[0]);
+  INIT_HLIST_HEAD(&fib_table_hash_ns()[i]);
+ ip_fib_local_table_ns() = fib_hash_init(RT_TABLE_LOCAL);
+ hlist_add_head_rcu(&ip_fib_local_table_ns()->tb_hlist,
+    &fib_table_hash_ns()[0]);
+ ip_fib_main_table_ns()  = fib_hash_init(RT_TABLE_MAIN);
+ hlist_add_head_rcu(&ip_fib_main_table_ns()->tb_hlist,
+    &fib_table_hash_ns()[0]);
+ return 0;
+}
+
+#elsif !defined(CONFIG_NET_NS)
+
+int inline ip_fib_struct_init(void)
+{
+ unsigned int i;
+
+ for (i = 0; i < FIB_TABLE_HASHSZ; i++)
+  INIT_HLIST_HEAD(&fib_table_hash_static[i]);
+ return fib4_rules_init();
+}
+
 #else
- fib4_rules_init();
-#endif
+
+int ip_fib_struct_init(void)
+{
+ struct net_namespace *ns = current_net_ns;
+ struct hlist_head *tables;
+ unsigned int i;
```

```
+
+ tables = kmalloc(FIB_TABLE_HASHSZ * sizeof(*tables), GFP_KERNEL);
+ if (tables == NULL)
+  return -ENOMEM;
+ for (i = 0; i < FIB_TABLE_HASHSZ; i++)
+  INIT_HLIST_HEAD(&tables[i]);
+ ns->fib4_tables = tables;
+
+ if (fib4_rules_init())
+  goto out_fib4_rules;
+ return 0;
+
+out_fib4_rules:
+ kfree(tables);
+ ns->fib4_tables = NULL;
+ return -ENOMEM;
+}
+
+#endif /* CONFIG_NET_NS */
+
+void __init ip_fib_init(void)
+{
+ ip_fib_struct_init();

  register_netdevice_notifier(&fib_netdev_notifier);
  register_inetaddr_notifier(&fib_inetaddr_notifier);
  nl_fib_lookup_init();
 }

+#ifdef CONFIG_NET_NS
+void ip_fib_struct_cleanup(void)
+{
+ current_net_ns->destroying = 1;
+ rtnl_lock();
+#ifdef CONFIG_IP_MULTIPLE_TABLES
+ fib4_rules_cleanup();
+#endif
+ /*
+  * FIB should already be empty since there is no netdevice,
+  * but clear it anyway
+  */
+ fib_flush();
+ rt_cache_flush(0);
+#ifdef CONFIG_IP_MULTIPLE_TABLES
+ kfree(fib_table_hash_ns());
+ fib_table_hash_ns() = NULL;
+#endif
+ fib_hashtable_destroy();
```

```
+ rtnl_unlock();
+}
+#endif /* CONFIG_NET_NS */
+
 EXPORT_SYMBOL(inet_addr_type);
 EXPORT_SYMBOL(ip_dev_find);
--- ./net/ipv4/fib_hash.c.vensroute Mon Aug 14 17:04:07 2006
+++ ./net/ipv4/fib_hash.c Mon Aug 14 17:18:59 2006
@@ -627,6 +627,11 @@ static int fn_flush_list(struct fn_zone
  struct hlist_node *node, *n;
  struct fib_node *f;
  int found = 0;
+#ifndef CONFIG_NET_NS
+ const int destroy = 0;
+#else
+ const int destroy = current_net_ns->destroying;
+#endif

  hlist_for_each_entry_safe(f, node, n, head, fn_hash) {
   struct fib_alias *fa, *fa_node;
@@ -636,7 +641,9 @@ static int fn_flush_list(struct fn_zone
   list_for_each_entry_safe(fa, fa_node, &f->fn_alias, fa_list) {
    struct fib_info *fi = fa->fa_info;

-   if (fi && (fi->fib_flags&RTNH_F_DEAD)) {
+   if (fi == NULL)
+    continue;
+   if (destroy || (fi->fib_flags&RTNH_F_DEAD)) {
     write_lock_bh(&fib_hash_lock);
     list_del(&fa->fa_list);
     if (list_empty(&f->fn_alias)) {
@@ -817,7 +824,7 @@ struct fib_iter_state {
 static struct fib_alias *fib_get_first(struct seq_file *seq)
 {
  struct fib_iter_state *iter = seq->private;
- struct fn_hash *table = (struct fn_hash *) ip_fib_main_table->tb_data;
+ struct fn_hash *table = (struct fn_hash *) ip_fib_main_table_ns()->tb_data;

  iter->bucket    = 0;
  iter->hash_head = NULL;
@@ -956,7 +963,7 @@ static void *fib_seq_start(struct seq_fi
  void *v = NULL;

  read_lock(&fib_hash_lock);
- if (ip_fib_main_table)
+ if (ip_fib_main_table_ns())
   v = *pos ? fib_get_idx(seq, *pos - 1) : SEQ_START_TOKEN;
  return v;
```

```
 }
--- ./net/ipv4/fib_rules.c.vensroute Mon Aug 14 17:04:08 2006
+++ ./net/ipv4/fib_rules.c Mon Aug 14 17:51:02 2006
@@ -32,7 +32,7 @@
 #include <net/ip_fib.h>
 #include <net/fib_rules.h>

-static struct fib_rules_ops fib4_rules_ops;
+static struct fib_rules_ops fib4_rules_ops_static;

 struct fib4_rule
 {
@@ -79,7 +79,12 @@ static struct fib4_rule local_rule = {
  },
 };

-static LIST_HEAD(fib4_rules);
+#ifndef CONFIG_NET_NS
+static LIST_HEAD(fib4_rules_static);
+#define fib4_rules_ops_ns() fib4_rules_ops_static
+#else
+#define fib4_rules_ops_ns() (*current_net_ns->fib4_rules_ops)
+#endif

 #ifdef CONFIG_NET_CLS_ROUTE
 u32 fib_rules_tclass(struct fib_result *res)
@@ -95,7 +100,7 @@ int fib_lookup(struct flowi *flp, struct
  };
  int err;

- err = fib_rules_lookup(&fib4_rules_ops, flp, 0, &arg);
+ err = fib_rules_lookup(&fib4_rules_ops_ns(), flp, 0, &arg);
  res->r = arg.rule;

  return err;
@@ -311,11 +316,13 @@ int fib4_rules_dump(struct sk_buff *skb,
 static u32 fib4_rule_default_pref(void)
 {
  struct list_head *pos;
+ struct list_head *fib4_rules;
  struct fib_rule *rule;

- if (!list_empty(&fib4_rules)) {
-  pos = fib4_rules.next;
-  if (pos->next != &fib4_rules) {
+ fib4_rules = fib4_rules_ops_ns().rules_list;
+ if (!list_empty(fib4_rules)) {
+  pos = fib4_rules->next;
```

```
+  if (pos->next != fib4_rules) {
    rule = list_entry(pos->next, struct fib_rule, list);
    if (rule->pref)
     return rule->pref - 1;
@@ -325,7 +332,7 @@ static u32 fib4_rule_default_pref(void)
  return 0;
 }

-static struct fib_rules_ops fib4_rules_ops = {
+static struct fib_rules_ops fib4_rules_ops_static = {
  .family  = AF_INET,
  .rule_size = sizeof(struct fib4_rule),
  .action  = fib4_rule_action,
@@ -336,15 +343,68 @@ static struct fib_rules_ops fib4_rules_o
  .default_pref = fib4_rule_default_pref,
  .nlgroup = RTNLGRP_IPV4_RULE,
  .policy  = fib4_rule_policy,
- .rules_list = &fib4_rules,
  .owner  = THIS_MODULE,
 };

-void __init fib4_rules_init(void)
+#ifndef CONFIG_NET_NS
+
+int fib4_rules_init(void)
+{
+ fib4_rules_ops_static.rules_list = &fib4_rules_static,
+ list_add_tail(&local_rule.common.list, &fib4_rules_static);
+ list_add_tail(&main_rule.common.list, &fib4_rules_static);
+ list_add_tail(&default_rule.common.list, &fib4_rules_static);
+ fib_rules_register(&fib4_rules_ops_static);
+ return 0;
+}
+
+#else
+
+static int fib4_rule_create(struct fib4_rule *orig, struct list_head *head)
+{
+ struct fib4_rule *p;
+
+ p = kmalloc(sizeof(*p), GFP_KERNEL);
+ if (p == NULL)
+  return -1;
+ memcpy(p, orig, sizeof(*p));
+ list_add_tail_rcu(&p->common.list, head);
+ return 0;
+}
+
```

```
+int fib4_rules_init(void)
 {
- list_add_tail(&local_rule.common.list, &fib4_rules);
- list_add_tail(&main_rule.common.list, &fib4_rules);
- list_add_tail(&default_rule.common.list, &fib4_rules);
+ struct fib_rules_ops *ops;
+ struct list_head *rules;
+
+ ops = kmalloc(sizeof(*ops) + sizeof(*rules), GFP_KERNEL);
+ if (ops == NULL)
+  goto out;
+ memcpy(ops, &fib4_rules_ops_static, sizeof(*ops));
+ rules = (struct list_head *)(ops + 1);
+ INIT_LIST_HEAD(rules);
+ ops->rules_list = rules;
+ current_net_ns->fib4_rules_ops = ops;
+
+ fib_rules_register(ops);
+
+ if (fib4_rule_create(&local_rule, rules) ||
+     fib4_rule_create(&main_rule, rules) ||
+     fib4_rule_create(&default_rule, rules))
+  goto out_rule;
+
+ return 0;

- fib_rules_register(&fib4_rules_ops);
+out_rule:
+ fib_rules_unregister(ops); /* list cleanup is inside */
+ kfree(ops);
+out:
+ return -ENOMEM;
 }
+
+void fib4_rules_cleanup(void)
+{
+ fib_rules_unregister(&fib4_rules_ops_ns());
+}
+
+#endif
--- ./net/ipv4/fib_semantics.c.vensroute Mon Aug 14 17:04:08 2006
+++ ./net/ipv4/fib_semantics.c Mon Aug 14 18:04:32 2006
@@ -50,10 +50,21 @@
 #define FSprintk(a...)

 static DEFINE_SPINLOCK(fib_info_lock);
-static struct hlist_head *fib_info_hash;
-static struct hlist_head *fib_info_laddrhash;
```

```
-static unsigned int fib_hash_size;
-static unsigned int fib_info_cnt;
+#ifndef CONFIG_NET_NS
+static struct hlist_head *fib_info_hash_static;
+static struct hlist_head *fib_info_laddrhash_static;
+static unsigned int fib_hash_size_static;
+static unsigned int fib_info_cnt_static;
+#define fib_info_hash(ns) fib_info_hash_static
+#define fib_info_laddrhash(ns) fib_info_laddrhash_static
+#define fib_hash_size(ns) fib_hash_size_static
+#define fib_info_cnt(ns) fib_info_cnt_static
+#else
+#define fib_info_hash(ns) ((ns)->fib4_hash)
+#define fib_info_laddrhash(ns) ((ns)->fib4_laddrhash)
+#define fib_hash_size(ns) ((ns)->fib4_hash_size)
+#define fib_info_cnt(ns) ((ns)->fib4_info_cnt)
+#endif

 #define DEVINDEX_HASHBITS 8
 #define DEVINDEX_HASHSIZE (1U << DEVINDEX_HASHBITS)
@@ -153,7 +164,7 @@ void free_fib_info(struct fib_info *fi)
    dev_put(nh->nh_dev);
   nh->nh_dev = NULL;
  } endfor_nexthops(fi);
- fib_info_cnt--;
+ fib_info_cnt(current_net_ns)--;
  kfree(fi);
 }

@@ -196,9 +207,10 @@ static __inline__ int nh_comp(const stru
  return 0;
 }

-static inline unsigned int fib_info_hashfn(const struct fib_info *fi)
+static inline unsigned int fib_info_hashfn(const struct fib_info *fi,
+   struct net_namespace *ns)
 {
- unsigned int mask = (fib_hash_size - 1);
+ unsigned int mask = (fib_hash_size(ns) - 1);
  unsigned int val = fi->fib_nhs;

  val ^= fi->fib_protocol;
@@ -210,13 +222,14 @@ static inline unsigned int fib_info_hash

 static struct fib_info *fib_find_info(const struct fib_info *nfi)
 {
+ struct net_namespace *ns = current_net_ns;
  struct hlist_head *head;
```

```
  struct hlist_node *node;
  struct fib_info *fi;
  unsigned int hash;

- hash = fib_info_hashfn(nfi);
- head = &fib_info_hash[hash];
+ hash = fib_info_hashfn(nfi, ns);
+ head = &fib_info_hash(ns)[hash];

  hlist_for_each_entry(fi, node, head, fib_hash) {
   if (fi->fib_nhs != nfi->fib_nhs)
@@ -236,11 +249,13 @@ static struct fib_info *fib_find_info(co

 static inline unsigned int fib_devindex_hashfn(unsigned int val)
 {
- unsigned int mask = DEVINDEX_HASHSIZE - 1;
+ unsigned int r, mask = DEVINDEX_HASHSIZE - 1;

- return (val ^
+ r = val ^
    (val >> DEVINDEX_HASHBITS) ^
-   (val >> (DEVINDEX_HASHBITS * 2))) & mask;
+   (val >> (DEVINDEX_HASHBITS * 2));
+ r ^= current_net_hash;
+ return r & mask;
 }

 /* Check, that the gateway is already configured.
@@ -563,9 +578,9 @@ out:
  return 0;
 }

-static inline unsigned int fib_laddr_hashfn(u32 val)
+static inline unsigned int fib_laddr_hashfn(u32 val, struct net_namespace *ns)
 {
- unsigned int mask = (fib_hash_size - 1);
+ unsigned int mask = (fib_hash_size(ns) - 1);

  return (val ^ (val >> 7) ^ (val >> 14)) & mask;
 }
@@ -594,17 +609,18 @@ static void fib_hash_move(struct hlist_h
      struct hlist_head *new_laddrhash,
      unsigned int new_size)
 {
+ struct net_namespace *ns = current_net_ns;
  struct hlist_head *old_info_hash, *old_laddrhash;
- unsigned int old_size = fib_hash_size;
+ unsigned int old_size = fib_hash_size(ns);
```

```
  unsigned int i, bytes;

  spin_lock(&fib_info_lock);
- old_info_hash = fib_info_hash;
- old_laddrhash = fib_info_laddrhash;
- fib_hash_size = new_size;
+ old_info_hash = fib_info_hash(ns);
+ old_laddrhash = fib_info_laddrhash(ns);
+ fib_hash_size(ns) = new_size;

  for (i = 0; i < old_size; i++) {
-  struct hlist_head *head = &fib_info_hash[i];
+  struct hlist_head *head = &old_info_hash[i];
   struct hlist_node *node, *n;
   struct fib_info *fi;

@@ -614,15 +630,15 @@ static void fib_hash_move(struct hlist_h

    hlist_del(&fi->fib_hash);

-   new_hash = fib_info_hashfn(fi);
+   new_hash = fib_info_hashfn(fi, ns);
    dest = &new_info_hash[new_hash];
    hlist_add_head(&fi->fib_hash, dest);
   }
  }
- fib_info_hash = new_info_hash;
+ fib_info_hash(ns) = new_info_hash;

  for (i = 0; i < old_size; i++) {
-  struct hlist_head *lhead = &fib_info_laddrhash[i];
+  struct hlist_head *lhead = &old_laddrhash[i];
   struct hlist_node *node, *n;
   struct fib_info *fi;

@@ -632,12 +648,12 @@ static void fib_hash_move(struct hlist_h

    hlist_del(&fi->fib_lhash);

-   new_hash = fib_laddr_hashfn(fi->fib_prefsrc);
+   new_hash = fib_laddr_hashfn(fi->fib_prefsrc, ns);
    ldest = &new_laddrhash[new_hash];
    hlist_add_head(&fi->fib_lhash, ldest);
   }
  }
- fib_info_laddrhash = new_laddrhash;
+ fib_info_laddrhash(ns) = new_laddrhash;
```

```
  spin_unlock(&fib_info_lock);

@@ -646,11 +662,27 @@ static void fib_hash_move(struct hlist_h
  fib_hash_free(old_laddrhash, bytes);
 }

+#ifdef CONFIG_NET_NS
+void fib_hashtable_destroy(void)
+{
+ struct net_namespace *ns;
+ unsigned int bytes;
+
+ ns = current_net_ns;
+ bytes = ns->fib4_hash_size * sizeof(struct hlist_head *);
+ fib_hash_free(ns->fib4_hash, bytes);
+ ns->fib4_hash = NULL;
+ fib_hash_free(ns->fib4_laddrhash, bytes);
+ ns->fib4_laddrhash = NULL;
+}
+#endif
+
 struct fib_info *
 fib_create_info(const struct rtmsg *r, struct kern_rta *rta,
    const struct nlmsghdr *nlh, int *errp)
 {
  int err;
+ struct net_namespace *ns = current_net_ns;
  struct fib_info *fi = NULL;
  struct fib_info *ofi;
 #ifdef CONFIG_IP_ROUTE_MULTIPATH
@@ -684,8 +716,8 @@ fib_create_info(const struct rtmsg *r, s
 #endif

  err = -ENOBUFS;
- if (fib_info_cnt >= fib_hash_size) {
-  unsigned int new_size = fib_hash_size << 1;
+ if (fib_info_cnt(ns) >= fib_hash_size(ns)) {
+  unsigned int new_size = fib_hash_size(ns) << 1;
    struct hlist_head *new_info_hash;
    struct hlist_head *new_laddrhash;
    unsigned int bytes;
@@ -705,14 +737,14 @@ fib_create_info(const struct rtmsg *r, s
     fib_hash_move(new_info_hash, new_laddrhash, new_size);
    }

-  if (!fib_hash_size)
+  if (!fib_hash_size(ns))
     goto failure;
```

```
  }

  fi = kzalloc(sizeof(*fi)+nhs*sizeof(struct fib_nh), GFP_KERNEL);
  if (fi == NULL)
   goto failure;
- fib_info_cnt++;
+ fib_info_cnt(ns)++;

  fi->fib_protocol = r->rtm_protocol;

@@ -822,11 +854,11 @@ link_it:
  atomic_inc(&fi->fib_clntref);
  spin_lock(&fib_info_lock);
  hlist_add_head(&fi->fib_hash,
-         &fib_info_hash[fib_info_hashfn(fi)]);
+         &fib_info_hash(ns)[fib_info_hashfn(fi, ns)]);
  if (fi->fib_prefsrc) {
   struct hlist_head *head;

-  head = &fib_info_laddrhash[fib_laddr_hashfn(fi->fib_prefsrc)];
+  head = &fib_info_laddrhash(ns)[fib_laddr_hashfn(fi->fib_prefsrc, ns)];
   hlist_add_head(&fi->fib_lhash, head);
  }
  change_nexthops(fi) {
@@ -1165,15 +1197,16 @@ fib_convert_rtentry(int cmd, struct nlms

 int fib_sync_down(u32 local, struct net_device *dev, int force)
 {
+ struct net_namespace *ns = current_net_ns;
  int ret = 0;
  int scope = RT_SCOPE_NOWHERE;

  if (force)
   scope = -1;

- if (local && fib_info_laddrhash) {
-  unsigned int hash = fib_laddr_hashfn(local);
-  struct hlist_head *head = &fib_info_laddrhash[hash];
+ if (local && fib_info_laddrhash(ns)) {
+  unsigned int hash = fib_laddr_hashfn(local, ns);
+  struct hlist_head *head = &fib_info_laddrhash(ns)[hash];
   struct hlist_node *node;
   struct fib_info *fi;

--- ./net/ipv4/route.c.vensroute Mon Aug 14 17:04:10 2006
+++ ./net/ipv4/route.c Mon Aug 14 17:18:59 2006
@@ -266,6 +266,7 @@ struct rt_cache_iter_state {
  int bucket;
```

```
 };

+static struct rtable *rt_cache_get_next(struct seq_file *seq, struct rtable *r);
 static struct rtable *rt_cache_get_first(struct seq_file *seq)
 {
  struct rtable *r = NULL;
@@ -278,21 +279,28 @@ static struct rtable *rt_cache_get_first
     break;
   rcu_read_unlock_bh();
  }
+ if (r && !net_ns_match(r->fl.net_ns, current_net_ns))
+  r = rt_cache_get_next(seq, r);
  return r;
 }

 static struct rtable *rt_cache_get_next(struct seq_file *seq, struct rtable *r)
 {
  struct rt_cache_iter_state *st = rcu_dereference(seq->private);
+ struct net_namespace *ns = current_net_ns;

+next:
  r = r->u.rt_next;
  while (!r) {
   rcu_read_unlock_bh();
   if (--st->bucket < 0)
-   break;
+   goto out;
   rcu_read_lock_bh();
   r = rt_hash_table[st->bucket].chain;
  }
+ if (!net_ns_match(r->fl.net_ns, ns))
+  goto next;
+out:
  return r;
 }

@@ -563,6 +571,7 @@ static inline u32 rt_score(struct rtable
 static inline int compare_keys(struct flowi *fl1, struct flowi *fl2)
 {
  return memcmp(&fl1->nl_u.ip4_u, &fl2->nl_u.ip4_u, sizeof(fl1->nl_u.ip4_u)) == 0 &&
+        net_ns_same(fl1->net_ns, fl2->net_ns) &&
         fl1->oif     == fl2->oif &&
         fl1->iif     == fl2->iif;
 }
@@ -1126,6 +1135,7 @@ void ip_rt_redirect(u32 old_gw, u32 dadd
  struct rtable *rth, **rthp;
  u32  skeys[2] = { saddr, 0 };
  int  ikeys[2] = { dev->ifindex, 0 };
```

```
+ struct net_namespace *ns = current_net_ns;
  struct netevent_redirect netevent;

  if (!in_dev)
@@ -1158,6 +1168,7 @@ void ip_rt_redirect(u32 old_gw, u32 dadd

     if (rth->fl.fl4_dst != daddr ||
         rth->fl.fl4_src != skeys[i] ||
+        !net_ns_match(rth->fl.net_ns, ns) ||
         rth->fl.oif != ikeys[k] ||
         rth->fl.iif != 0) {
       rthp = &rth->u.rt_next;
@@ -1649,6 +1660,9 @@ static int ip_route_input_mc(struct sk_b
  dev_hold(rth->u.dst.dev);
  rth->idev = in_dev_get(rth->u.dst.dev);
  rth->fl.oif = 0;
+#ifdef CONFIG_NET_NS
+ rth->fl.net_ns = current_net_ns;
+#endif
  rth->rt_gateway = daddr;
  rth->rt_spec_dst= spec_dst;
  rth->rt_type = RTN_MULTICAST;
@@ -1792,6 +1806,9 @@ static inline int __mkroute_input(struct
  dev_hold(rth->u.dst.dev);
  rth->idev = in_dev_get(rth->u.dst.dev);
  rth->fl.oif  = 0;
+#ifdef CONFIG_NET_NS
+ rth->fl.net_ns = current_net_ns;
+#endif
  rth->rt_spec_dst= spec_dst;

  rth->u.dst.input = ip_forward;
@@ -2093,6 +2110,7 @@ int ip_route_input(struct sk_buff *skb,
  struct rtable * rth;
  unsigned hash;
  int iif = dev->ifindex;
+ struct net_namespace *ns = current_net_ns;

  tos &= IPTOS_RT_MASK;
  hash = rt_hash_code(daddr, saddr ^ (iif << 5));
@@ -2102,6 +2120,7 @@ int ip_route_input(struct sk_buff *skb,
       rth = rcu_dereference(rth->u.rt_next)) {
   if (rth->fl.fl4_dst == daddr &&
       rth->fl.fl4_src == saddr &&
+      net_ns_match(rth->fl.net_ns, ns) &&
       rth->fl.iif == iif &&
       rth->fl.oif == 0 &&
 #ifdef CONFIG_IP_ROUTE_FWMARK
```

```
@@ -2241,6 +2260,9 @@ static inline int __mkroute_output(struc
  rth->u.dst.dev = dev_out;
  dev_hold(dev_out);
  rth->idev = in_dev_get(dev_out);
+#ifdef CONFIG_NET_NS
+ rth->fl.net_ns = current_net_ns;
+#endif
  rth->rt_gateway = fl->fl4_dst;
  rth->rt_spec_dst= fl->fl4_src;

@@ -2566,6 +2588,7 @@ int __ip_route_output_key(struct rtable
 {
  unsigned hash;
  struct rtable *rth;
+ struct net_namespace *ns = current_net_ns;

  hash = rt_hash_code(flp->fl4_dst, flp->fl4_src ^ (flp->oif << 5));

@@ -2574,6 +2597,7 @@ int __ip_route_output_key(struct rtable
   rth = rcu_dereference(rth->u.rt_next)) {
   if (rth->fl.fl4_dst == flp->fl4_dst &&
       rth->fl.fl4_src == flp->fl4_src &&
+      net_ns_match(rth->fl.net_ns, ns) &&
       rth->fl.iif == 0 &&
       rth->fl.oif == flp->oif &&
 #ifdef CONFIG_IP_ROUTE_FWMARK
```