Subject: Re: [PATCH] Module for ip utility to support veth device
Posted by Patrick McHardy on Wed, 06 Jun 2007 15:18:59 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> diff --git a/ip/iplink.c b/ip/iplink.c
> index 5170419..6975990 100644
> --- a/ip/iplink.c
> +++ b/ip/iplink.c
> @@ -287,7 +287,7 @@ static int iplink_modify(int cmd, unsign
> strlen(type));
>
> lu = get_link_type(type);
> - if (lu) {
> + if (lu && argc) {
> struct rtattr * data = NLMSG_TAIL(&req.n);
> addattr_l(&req.n, sizeof(req), IFLA_INFO_DATA, NULL, 0);

I've folded this part into my iproute patch, thanks.

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

Subject: [PATCH] Virtual ethernet device (v.4)
Posted by Pavel Emelianov on Thu, 19 Jul 2007 09:24:55 GMT
View Forum Message <> Reply to Message

Veth stands for Virtual ETHernet. It is a simple tunnel driver
that works at the link layer and looks like a pair of ethernet
devices interconnected with each other.

Mainly it allows to communicate between network namespaces but
it can be used as is as well. E.g. one may join to bridged
networking segments together.

Eric recently sent a similar driver called etun with the sysfs
interface. This implementation uses another interface - the
RTM_NRELINK message introduced by Patric.

The newlink callback is organized that way to make it easy to
create the peer device in the separate namespace when we have
them in kernel.

Many thanks to Patrick for reviewing the patches and his advises
on how to make driver cleaner.

Changes from v.3:
 * Reserved place for struct ifinfomsg in IFLA_INFO_DATA part
   of the packet. This is not used yet, but may be in the
   future.

Changes from v.2.1:
 * Made the generic routine for link creation to be used
   by veth driver, any other tunnel driver that needs to
   create several devices at once and rtnl_newlink() code.

Changes from v.2:
 * Rebase over latest netdev tree. No actual changes;
 * Small code rework.

Changes from v.1:
 * Per-cpu statistics;
 * Standard convention for nla policy names;
 * Module alias added;
 * Xmit function fixes noticed by Patric;
 * Code cleanup.

The patch for an ip utility is also provided.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Acked-by: Patrick McHardy <kaber@trash.net>

Subject: [PATCH 1/2] Introduce the generic rtnl_create_link()
Posted by Pavel Emelianov on Thu, 19 Jul 2007 09:26:52 GMT
View Forum Message <> Reply to Message

This routine gets the parsed rtnl attributes and creates a new
link with generic info (IFLA_LINKINFO policy). Its intention
is to help the drivers, that need to create several links at
once (like VETH).

This is nothing but a copy-paste-ed part of rtnl_newlink() function
that is responsible for creation of new device.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Acked-by: Patrick McHardy <kaber@trash.net>

---

 include/net/rtnetlink.h | 4 ++
 net/core/rtnetlink.c | 83 +++++++++++++++++++++++++++++++----------------
 2 files changed, 57 insertions(+), 30 deletions(-)

```
diff --git a/include/net/rtnetlink.h b/include/net/rtnetlink.h
index 3861c05..8218288 100644
--- a/include/net/rtnetlink.h
+++ b/include/net/rtnetlink.h
@@ -78,6 +78,10 @@ extern void __rtnl_link_unregister(struc
 extern int rtnl_link_register(struct rtnl_link_ops *ops);
 extern void rtnl_link_unregister(struct rtnl_link_ops *ops);

+extern struct net_device *rtnl_create_link(char *ifname,
+  const struct rtnl_link_ops *ops, struct nlattr *tb[]);
+extern const struct nla_policy ifla_policy[IFLA_MAX+1];
+
 #define MODULE_ALIAS_RTNL_LINK(kind) MODULE_ALIAS("rtnl-link-" kind)

 #endif
diff --git a/net/core/rtnetlink.c b/net/core/rtnetlink.c
index 864cbdf..02e00e8 100644
--- a/net/core/rtnetlink.c
+++ b/net/core/rtnetlink.c
@@ -714,7 +714,7 @@ cont:
  return skb->len;
 }

-static const struct nla_policy ifla_policy[IFLA_MAX+1] = {
+const struct nla_policy ifla_policy[IFLA_MAX+1] = {
  [IFLA_IFNAME]  = { .type = NLA_STRING, .len = IFNAMSIZ-1 },
  [IFLA_ADDRESS]  = { .type = NLA_BINARY, .len = MAX_ADDR_LEN },
  [IFLA_BROADCAST] = { .type = NLA_BINARY, .len = MAX_ADDR_LEN },
@@ -941,6 +941,50 @@ static int rtnl_dellink(struct sk_buff *
  return 0;
 }

+struct net_device *rtnl_create_link(char *ifname,
+  const struct rtnl_link_ops *ops, struct nlattr *tb[])
+{
+ int err;
+ struct net_device *dev;
+
+ err = -ENOMEM;
+ dev = alloc_netdev(ops->priv_size, ifname, ops->setup);
+ if (!dev)
+  goto err;
+
+ if (strchr(dev->name, '%')) {
+  err = dev_alloc_name(dev, dev->name);
+  if (err < 0)
+   goto err_free;
```

```
+ }
+
+ dev->rtnl_link_ops = ops;
+
+ if (tb[IFLA_MTU])
+  dev->mtu = nla_get_u32(tb[IFLA_MTU]);
+ if (tb[IFLA_ADDRESS])
+  memcpy(dev->dev_addr, nla_data(tb[IFLA_ADDRESS]),
+    nla_len(tb[IFLA_ADDRESS]));
+ if (tb[IFLA_BROADCAST])
+  memcpy(dev->broadcast, nla_data(tb[IFLA_BROADCAST]),
+    nla_len(tb[IFLA_BROADCAST]));
+ if (tb[IFLA_TXQLEN])
+  dev->tx_queue_len = nla_get_u32(tb[IFLA_TXQLEN]);
+ if (tb[IFLA_WEIGHT])
+  dev->weight = nla_get_u32(tb[IFLA_WEIGHT]);
+ if (tb[IFLA_OPERSTATE])
+  set_operstate(dev, nla_get_u8(tb[IFLA_OPERSTATE]));
+ if (tb[IFLA_LINKMODE])
+  dev->link_mode = nla_get_u8(tb[IFLA_LINKMODE]);
+
+ return dev;
+
+err_free:
+ free_netdev(dev);
+err:
+ return ERR_PTR(err);
+}
+
 static int rtnl_newlink(struct sk_buff *skb, struct nlmsghdr *nlh, void *arg)
 {
  const struct rtnl_link_ops *ops;
@@ -1051,40 +1095,17 @@ replay:

   if (!ifname[0])
    snprintf(ifname, IFNAMSIZ, "%s%%d", ops->kind);
-  dev = alloc_netdev(ops->priv_size, ifname, ops->setup);
-  if (!dev)
-   return -ENOMEM;
-
-  if (strchr(dev->name, '%')) {
-   err = dev_alloc_name(dev, dev->name);
-   if (err < 0)
-    goto err_free;
-  }
-  dev->rtnl_link_ops = ops;

-  if (tb[IFLA_MTU])
```

```
-   dev->mtu = nla_get_u32(tb[IFLA_MTU]);
-  if (tb[IFLA_ADDRESS])
-   memcpy(dev->dev_addr, nla_data(tb[IFLA_ADDRESS]),
-          nla_len(tb[IFLA_ADDRESS]));
-  if (tb[IFLA_BROADCAST])
-   memcpy(dev->broadcast, nla_data(tb[IFLA_BROADCAST]),
-          nla_len(tb[IFLA_BROADCAST]));
-  if (tb[IFLA_TXQLEN])
-   dev->tx_queue_len = nla_get_u32(tb[IFLA_TXQLEN]);
-  if (tb[IFLA_WEIGHT])
-   dev->weight = nla_get_u32(tb[IFLA_WEIGHT]);
-  if (tb[IFLA_OPERSTATE])
-   set_operstate(dev, nla_get_u8(tb[IFLA_OPERSTATE]));
-  if (tb[IFLA_LINKMODE])
-   dev->link_mode = nla_get_u8(tb[IFLA_LINKMODE]);
+  dev = rtnl_create_link(ifname, ops, tb);

-  if (ops->newlink)
+  if (IS_ERR(dev))
+   err = PTR_ERR(dev);
+  else if (ops->newlink)
    err = ops->newlink(dev, tb, data);
   else
    err = register_netdevice(dev);
-err_free:
-  if (err < 0)
+
+  if (err < 0 && !IS_ERR(dev))
    free_netdev(dev);
   return err;
  }
@@ -1333,3 +1354,5 @@ EXPORT_SYMBOL(rtnl_unlock);
 EXPORT_SYMBOL(rtnl_unicast);
 EXPORT_SYMBOL(rtnl_notify);
 EXPORT_SYMBOL(rtnl_set_sk_err);
+EXPORT_SYMBOL(rtnl_create_link);
+EXPORT_SYMBOL(ifla_policy);
```

Subject: [PATCH 2/2] Virtual ethernet device driver
Posted by Pavel Emelianov on Thu, 19 Jul 2007 09:28:39 GMT
View Forum Message <> Reply to Message

Veth stands for Virtual ETHernet. It is a simple tunnel driver
that works at the link layer and looks like a pair of ethernet
devices interconnected with each other.

Mainly it allows to communicate between network namespaces but

it can be used as is as well.

The newlink callback is organized that way to make it easy to
create the peer device in the separate namespace when we have
them in kernel.

This implementation uses another interface - the RTM_NRELINK
message introduced by Patric.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
Acked-by: Patrick McHardy <kaber@trash.net>

```
 drivers/net/Kconfig  |    6
 drivers/net/Makefile |    1
 drivers/net/veth.c   |  475 +++++++++++++++++++++++++++++++++++++++++++++++++++++
 include/net/veth.h   |   12 +
 4 files changed, 494 insertions(+)

diff --git a/drivers/net/Kconfig b/drivers/net/Kconfig
index 43d0317..ccfc002 100644
--- a/drivers/net/Kconfig
+++ b/drivers/net/Kconfig
@@ -134,6 +134,12 @@ config TUN

    If you don't know what to use this for, you don't need it.

+config VETH
+ tristate "Virtual ethernet device"
+ ---help---
+   The device is an ethernet tunnel. Devices are created in pairs. When
+   one end receives the packet it appears on its pair and vice versa.
+
 config NET_SB1000
  tristate "General Instruments Surfboard 1000"
  depends on PNP
diff --git a/drivers/net/Makefile b/drivers/net/Makefile
index eb41676..32db559 100644
--- a/drivers/net/Makefile
+++ b/drivers/net/Makefile
@@ -189,6 +189,7 @@ obj-$(CONFIG_MACSONIC) += macsonic.o
 obj-$(CONFIG_MACMACE) += macmace.o
 obj-$(CONFIG_MAC89x0) += mac89x0.o
 obj-$(CONFIG_TUN) += tun.o
+obj-$(CONFIG_VETH) += veth.o
 obj-$(CONFIG_NET_NETX) += netx-eth.o
 obj-$(CONFIG_DL2K) += dl2k.o
```

```
 obj-$(CONFIG_R8169) += r8169.o
diff --git a/drivers/net/veth.c b/drivers/net/veth.c
new file mode 100644
index 0000000..b8f33f0
--- /dev/null
+++ b/drivers/net/veth.c
@@ -0,0 +1,475 @@
+/*
+ *  drivers/net/veth.c
+ *
+ *  Copyright (C) 2007 OpenVZ http://openvz.org, SWsoft Inc
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ * Ethtool interface from: Eric W. Biederman <ebiederm@xmission.com>
+ *
+ */
+
+#include <linux/list.h>
+#include <linux/netdevice.h>
+#include <linux/ethtool.h>
+#include <linux/etherdevice.h>
+
+#include <net/dst.h>
+#include <net/xfrm.h>
+#include <net/veth.h>
+
+#define DRV_NAME "veth"
+#define DRV_VERSION "1.0"
+
+struct veth_net_stats {
+ unsigned long rx_packets;
+ unsigned long tx_packets;
+ unsigned long rx_bytes;
+ unsigned long tx_bytes;
+ unsigned long tx_dropped;
+};
+
+struct veth_priv {
+ struct net_device *peer;
+ struct net_device *dev;
+ struct list_head list;
+ struct veth_net_stats *stats;
+ unsigned ip_summed;
+};
+
+static LIST_HEAD(veth_list);
+
+/*
```

```
+ * ethtool interface
+ */
+
+static struct {
+ const char string[ETH_GSTRING_LEN];
+} ethtool_stats_keys[] = {
+ { "peer_ifindex" },
+};
+
+static int veth_get_settings(struct net_device *dev, struct ethtool_cmd *cmd)
+{
+ cmd->supported  = 0;
+ cmd->advertising = 0;
+ cmd->speed  = SPEED_10000;
+ cmd->duplex  = DUPLEX_FULL;
+ cmd->port  = PORT_TP;
+ cmd->phy_address = 0;
+ cmd->transceiver = XCVR_INTERNAL;
+ cmd->autoneg  = AUTONEG_DISABLE;
+ cmd->maxtxpkt  = 0;
+ cmd->maxrxpkt  = 0;
+ return 0;
+}
+
+static void veth_get_drvinfo(struct net_device *dev, struct ethtool_drvinfo *info)
+{
+ strcpy(info->driver, DRV_NAME);
+ strcpy(info->version, DRV_VERSION);
+ strcpy(info->fw_version, "N/A");
+}
+
+static void veth_get_strings(struct net_device *dev, u32 stringset, u8 *buf)
+{
+ switch(stringset) {
+ case ETH_SS_STATS:
+  memcpy(buf, &ethtool_stats_keys, sizeof(ethtool_stats_keys));
+  break;
+ }
+}
+
+static int veth_get_stats_count(struct net_device *dev)
+{
+ return ARRAY_SIZE(ethtool_stats_keys);
+}
+
+static void veth_get_ethtool_stats(struct net_device *dev,
+  struct ethtool_stats *stats, u64 *data)
+{
```

```
+ struct veth_priv *priv;
+
+ priv = netdev_priv(dev);
+ data[0] = priv->peer->ifindex;
+}
+
+static u32 veth_get_rx_csum(struct net_device *dev)
+{
+ struct veth_priv *priv;
+
+ priv = netdev_priv(dev);
+ return priv->ip_summed == CHECKSUM_UNNECESSARY;
+}
+
+static int veth_set_rx_csum(struct net_device *dev, u32 data)
+{
+ struct veth_priv *priv;
+
+ priv = netdev_priv(dev);
+ priv->ip_summed = data ? CHECKSUM_UNNECESSARY : CHECKSUM_NONE;
+ return 0;
+}
+
+static u32 veth_get_tx_csum(struct net_device *dev)
+{
+ return (dev->features & NETIF_F_NO_CSUM) != 0;
+}
+
+static int veth_set_tx_csum(struct net_device *dev, u32 data)
+{
+ if (data)
+  dev->features |= NETIF_F_NO_CSUM;
+ else
+  dev->features &= ~NETIF_F_NO_CSUM;
+ return 0;
+}
+
+static struct ethtool_ops veth_ethtool_ops = {
+ .get_settings  = veth_get_settings,
+ .get_drvinfo  = veth_get_drvinfo,
+ .get_link  = ethtool_op_get_link,
+ .get_rx_csum  = veth_get_rx_csum,
+ .set_rx_csum  = veth_set_rx_csum,
+ .get_tx_csum  = veth_get_tx_csum,
+ .set_tx_csum  = veth_set_tx_csum,
+ .get_sg   = ethtool_op_get_sg,
+ .set_sg   = ethtool_op_set_sg,
+ .get_strings  = veth_get_strings,
```

```
+ .get_stats_count = veth_get_stats_count,
+ .get_ethtool_stats = veth_get_ethtool_stats,
+ .get_perm_addr  = ethtool_op_get_perm_addr,
+};
+
+/*
+ * xmit
+ */
+
+static int veth_xmit(struct sk_buff *skb, struct net_device *dev)
+{
+ struct net_device *rcv = NULL;
+ struct veth_priv *priv, *rcv_priv;
+ struct veth_net_stats *stats;
+ int length, cpu;
+
+ skb_orphan(skb);
+
+ priv = netdev_priv(dev);
+ rcv = priv->peer;
+ rcv_priv = netdev_priv(rcv);
+
+ cpu = smp_processor_id();
+ stats = per_cpu_ptr(priv->stats, cpu);
+
+ if (!(rcv->flags & IFF_UP))
+  goto outf;
+
+ skb->pkt_type = PACKET_HOST;
+ skb->protocol = eth_type_trans(skb, rcv);
+ if (dev->features & NETIF_F_NO_CSUM)
+  skb->ip_summed = rcv_priv->ip_summed;
+
+ dst_release(skb->dst);
+ skb->dst = NULL;
+ skb->mark = 0;
+ secpath_reset(skb);
+ nf_reset(skb);
+
+ length = skb->len;
+
+ stats->tx_bytes += length;
+ stats->tx_packets++;
+
+ stats = per_cpu_ptr(rcv_priv->stats, cpu);
+ stats->rx_bytes += length;
+ stats->rx_packets++;
+
```

```
+ netif_rx(skb);
+ return 0;
+
+outf:
+ kfree_skb(skb);
+ stats->tx_dropped++;
+ return 0;
+}
+
+/*
+ * general routines
+ */
+
+static struct net_device_stats *veth_get_stats(struct net_device *dev)
+{
+ struct veth_priv *priv;
+ struct net_device_stats *dev_stats;
+ int cpu;
+ struct veth_net_stats *stats;
+
+ priv = netdev_priv(dev);
+ dev_stats = &dev->stats;
+
+ dev_stats->rx_packets = 0;
+ dev_stats->tx_packets = 0;
+ dev_stats->rx_bytes = 0;
+ dev_stats->tx_bytes = 0;
+ dev_stats->tx_dropped = 0;
+
+ for_each_online_cpu(cpu) {
+  stats = per_cpu_ptr(priv->stats, cpu);
+
+  dev_stats->rx_packets += stats->rx_packets;
+  dev_stats->tx_packets += stats->tx_packets;
+  dev_stats->rx_bytes += stats->rx_bytes;
+  dev_stats->tx_bytes += stats->tx_bytes;
+  dev_stats->tx_dropped += stats->tx_dropped;
+ }
+
+ return dev_stats;
+}
+
+static int veth_open(struct net_device *dev)
+{
+ struct veth_priv *priv;
+
+ priv = netdev_priv(dev);
+ if (priv->peer == NULL)
```

```
+  return -ENOTCONN;
+
+ if (priv->peer->flags & IFF_UP) {
+  netif_carrier_on(dev);
+  netif_carrier_on(priv->peer);
+ }
+ return 0;
+}
+
+static int veth_close(struct net_device *dev)
+{
+ struct veth_priv *priv;
+
+ if (netif_carrier_ok(dev)) {
+  priv = netdev_priv(dev);
+  netif_carrier_off(dev);
+  netif_carrier_off(priv->peer);
+ }
+ return 0;
+}
+
+static int veth_dev_init(struct net_device *dev)
+{
+ struct veth_net_stats *stats;
+ struct veth_priv *priv;
+
+ stats = alloc_percpu(struct veth_net_stats);
+ if (stats == NULL)
+  return -ENOMEM;
+
+ priv = netdev_priv(dev);
+ priv->stats = stats;
+ return 0;
+}
+
+static void veth_dev_free(struct net_device *dev)
+{
+ struct veth_priv *priv;
+
+ priv = netdev_priv(dev);
+ free_percpu(priv->stats);
+ free_netdev(dev);
+}
+
+static void veth_setup(struct net_device *dev)
+{
+ ether_setup(dev);
+
```

```
+ dev->hard_start_xmit = veth_xmit;
+ dev->get_stats = veth_get_stats;
+ dev->open = veth_open;
+ dev->stop = veth_close;
+ dev->ethtool_ops = &veth_ethtool_ops;
+ dev->features |= NETIF_F_LLTX;
+ dev->init = veth_dev_init;
+ dev->destructor = veth_dev_free;
+ netif_carrier_off(dev);
+}
+
+/*
+ * netlink interface
+ */
+
+static int veth_validate(struct nlattr *tb[], struct nlattr *data[])
+{
+ if (tb[IFLA_ADDRESS]) {
+  if (nla_len(tb[IFLA_ADDRESS]) != ETH_ALEN)
+   return -EINVAL;
+  if (!is_valid_ether_addr(nla_data(tb[IFLA_ADDRESS])))
+   return -EADDRNOTAVAIL;
+ }
+ return 0;
+}
+
+static struct rtnl_link_ops veth_link_ops;
+
+static int veth_newlink(struct net_device *dev,
+    struct nlattr *tb[], struct nlattr *data[])
+{
+ int err;
+ struct net_device *peer;
+ struct veth_priv *priv;
+ char ifname[IFNAMSIZ];
+ struct nlattr *peer_tb[IFLA_MAX + 1], **tbp;
+
+ /*
+  * create and register peer first
+  *
+  * struct ifinfomsg is at the head of VETH_INFO_PEER, but we
+  * skip it since no info from it is useful yet
+  */
+
+ if (data != NULL && data[VETH_INFO_PEER] != NULL) {
+  struct nlattr *nla_peer;
+
+  nla_peer = data[VETH_INFO_PEER];
```

```
+  err = nla_parse(peer_tb, IFLA_MAX,
+    nla_data(nla_peer) + sizeof(struct ifinfomsg),
+    nla_len(nla_peer) - sizeof(struct ifinfomsg),
+    ifla_policy);
+  if (err < 0)
+   return err;
+
+  err = veth_validate(peer_tb, NULL);
+  if (err < 0)
+   return err;
+
+  tbp = peer_tb;
+ } else
+  tbp = tb;
+
+ if (tbp[IFLA_IFNAME])
+  nla_strlcpy(ifname, tbp[IFLA_IFNAME], IFNAMSIZ);
+ else
+  snprintf(ifname, IFNAMSIZ, DRV_NAME "%%d");
+
+ peer = rtnl_create_link(ifname, &veth_link_ops, tbp);
+ if (IS_ERR(peer))
+  return PTR_ERR(peer);
+
+ if (tbp[IFLA_ADDRESS] == NULL)
+  random_ether_addr(peer->dev_addr);
+
+ err = register_netdevice(peer);
+ if (err < 0)
+  goto err_register_peer;
+
+ /*
+  * register dev last
+  *
+  * note, that since we've registered new device the dev's name
+  * should be re-allocated
+  */
+
+ if (tb[IFLA_ADDRESS] == NULL)
+  random_ether_addr(dev->dev_addr);
+
+ if (tb[IFLA_IFNAME])
+  nla_strlcpy(dev->name, tb[IFLA_IFNAME], IFNAMSIZ);
+ else
+  snprintf(dev->name, IFNAMSIZ, DRV_NAME "%%d");
+
+ if (strchr(dev->name, '%')) {
+  err = dev_alloc_name(dev, dev->name);
```

```
+  if (err < 0)
+   goto err_alloc_name;
+ }
+
+ err = register_netdevice(dev);
+ if (err < 0)
+  goto err_register_dev;
+
+ /*
+  * tie the deviced together
+  */
+
+ priv = netdev_priv(dev);
+ priv->dev = dev;
+ priv->peer = peer;
+ list_add(&priv->list, &veth_list);
+
+ priv = netdev_priv(peer);
+ priv->dev = peer;
+ priv->peer = dev;
+ INIT_LIST_HEAD(&priv->list);
+ return 0;
+
+err_register_dev:
+ /* nothing to do */
+err_alloc_name:
+ unregister_netdevice(peer);
+ return err;
+
+err_register_peer:
+ free_netdev(peer);
+ return err;
+}
+
+static void veth_dellink(struct net_device *dev)
+{
+ struct veth_priv *priv;
+ struct net_device *peer;
+
+ priv = netdev_priv(dev);
+ peer = priv->peer;
+
+ if (!list_empty(&priv->list))
+  list_del(&priv->list);
+
+ priv = netdev_priv(peer);
+ if (!list_empty(&priv->list))
+  list_del(&priv->list);
```

```
+
+ unregister_netdevice(dev);
+ unregister_netdevice(peer);
+}
+
+static const struct nla_policy veth_policy[VETH_INFO_MAX + 1];
+
+static struct rtnl_link_ops veth_link_ops = {
+ .kind  = DRV_NAME,
+ .priv_size = sizeof(struct veth_priv),
+ .setup  = veth_setup,
+ .validate = veth_validate,
+ .newlink = veth_newlink,
+ .dellink = veth_dellink,
+ .policy  = veth_policy,
+ .maxtype = VETH_INFO_MAX,
+};
+
+/*
+ * init/fini
+ */
+
+static __init int veth_init(void)
+{
+ return rtnl_link_register(&veth_link_ops);
+}
+
+static __exit void veth_exit(void)
+{
+ struct veth_priv *priv, *next;
+
+ rtnl_lock();
+ /*
+  * cannot trust __rtnl_link_unregister() to unregister all
+  * devices, as each ->dellink call will remove two devices
+  * from the list at once.
+  */
+ list_for_each_entry_safe(priv, next, &veth_list, list)
+  veth_dellink(priv->dev);
+
+ __rtnl_link_unregister(&veth_link_ops);
+ rtnl_unlock();
+}
+
+module_init(veth_init);
+module_exit(veth_exit);
+
+MODULE_DESCRIPTION("Virtual Ethernet Tunnel");
```

```
+MODULE_LICENSE("GPL v2");
+MODULE_ALIAS_RTNL_LINK(DRV_NAME);
diff --git a/include/net/veth.h b/include/net/veth.h
new file mode 100644
index 0000000..3354c1e
--- /dev/null
+++ b/include/net/veth.h
@@ -0,0 +1,12 @@
+#ifndef __NET_VETH_H_
+#define __NET_VETH_H_
+
+enum {
+ VETH_INFO_UNSPEC,
+ VETH_INFO_PEER,
+
+ __VETH_INFO_MAX
+#define VETH_INFO_MAX (__VETH_INFO_MAX - 1)
+};
+
+#endif
```

Subject: [PATCH] Module for ip utility to support veth device (v.3)
Posted by Pavel Emelianov on Thu, 19 Jul 2007 09:30:24 GMT
View Forum Message <> Reply to Message

The usage is
ip link add [name] type veth [device parameters] peer [name <name> <other device params>]

This consists of two parts:
1/2 makes some copy-paste changes in ip/iplink.c for generic CLI
attributes parsing,
2/2 the module itself.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
Acked-by: Patrick McHardy <kaber@trash.net>

Subject: [PATCH 1/2] Introduce iplink_parse() routine
Posted by Pavel Emelianov on Thu, 19 Jul 2007 09:32:31 GMT
View Forum Message <> Reply to Message

This routine parses CLI attributes, describing generic link
parameters such as name, address, etc.

This is mostly copy-pasted from iplink_modify().

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
Acked-by: Patrick McHardy <kaber@trash.net>

```
---

 include/utils.h |    3 +
 ip/iplink.c     |  127 ++++++++++++++++++++++++++++++++++++++++++------------------------------
 2 files changed, 76 insertions(+), 54 deletions(-)

diff --git a/include/utils.h b/include/utils.h
index a3fd335..3fd851d 100644
--- a/include/utils.h
+++ b/include/utils.h
@@ -146,4 +146,7 @@ extern int cmdlineno;
 extern size_t getcmdline(char **line, size_t *len, FILE *in);
 extern int makeargs(char *line, char *argv[], int maxargs);

+struct iplink_req;
+int iplink_parse(int argc, char **argv, struct iplink_req *req,
+  char **name, char **type, char **link, char **dev);
 #endif /* __UTILS_H__ */
diff --git a/ip/iplink.c b/ip/iplink.c
index cfacdab..e0e0d85 100644
--- a/ip/iplink.c
+++ b/ip/iplink.c
@@ -142,140 +142,159 @@ static int iplink_have_newlink(void)
 }
 #endif /* ! IPLINK_IOCTL_COMPAT */

-static int iplink_modify(int cmd, unsigned int flags, int argc, char **argv)
+struct iplink_req {
+ struct nlmsghdr  n;
+ struct ifinfomsg i;
+ char   buf[1024];
+};
+
+int iplink_parse(int argc, char **argv, struct iplink_req *req,
+  char **name, char **type, char **link, char **dev)
 {
+ int ret, len;
+ char abuf[32];
  int qlen = -1;
  int mtu = -1;
- int len;
- char abuf[32];
- char *dev = NULL;
- char *name = NULL;
- char *link = NULL;
```

```
- char *type = NULL;
- struct link_util *lu = NULL;
- struct {
-  struct nlmsghdr  n;
-  struct ifinfomsg i;
-  char   buf[1024];
- } req;

- memset(&req, 0, sizeof(req));
-
- req.n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
- req.n.nlmsg_flags = NLM_F_REQUEST|flags;
- req.n.nlmsg_type = cmd;
- req.i.ifi_family = preferred_family;
+ ret = argc;

  while (argc > 0) {
   if (strcmp(*argv, "up") == 0) {
-   req.i.ifi_change |= IFF_UP;
-   req.i.ifi_flags |= IFF_UP;
+   req->i.ifi_change |= IFF_UP;
+   req->i.ifi_flags |= IFF_UP;
   } else if (strcmp(*argv, "down") == 0) {
-   req.i.ifi_change |= IFF_UP;
-   req.i.ifi_flags &= ~IFF_UP;
+   req->i.ifi_change |= IFF_UP;
+   req->i.ifi_flags &= ~IFF_UP;
   } else if (strcmp(*argv, "name") == 0) {
    NEXT_ARG();
-   name = *argv;
+   *name = *argv;
   } else if (matches(*argv, "link") == 0) {
    NEXT_ARG();
-   link = *argv;
+   *link = *argv;
   } else if (matches(*argv, "address") == 0) {
    NEXT_ARG();
    len = ll_addr_a2n(abuf, sizeof(abuf), *argv);
-   addattr_l(&req.n, sizeof(req), IFLA_ADDRESS, abuf, len);
+   addattr_l(&req->n, sizeof(*req), IFLA_ADDRESS, abuf, len);
   } else if (matches(*argv, "broadcast") == 0 ||
-      strcmp(*argv, "brd") == 0) {
+    strcmp(*argv, "brd") == 0) {
    NEXT_ARG();
    len = ll_addr_a2n(abuf, sizeof(abuf), *argv);
-   addattr_l(&req.n, sizeof(req), IFLA_BROADCAST, abuf, len);
+   addattr_l(&req->n, sizeof(*req), IFLA_BROADCAST, abuf, len);
   } else if (matches(*argv, "txqueuelen") == 0 ||
```

```
-      strcmp(*argv, "qlen") == 0 ||
-      matches(*argv, "txqlen") == 0) {
+     strcmp(*argv, "qlen") == 0 ||
+     matches(*argv, "txqlen") == 0) {
     NEXT_ARG();
     if (qlen != -1)
      duparg("txqueuelen", *argv);
     if (get_integer(&qlen,  *argv, 0))
      invarg("Invalid \"txqueuelen\" value\n", *argv);
-    addattr_l(&req.n, sizeof(req), IFLA_TXQLEN, &qlen, 4);
+    addattr_l(&req->n, sizeof(*req), IFLA_TXQLEN, &qlen, 4);
    } else if (strcmp(*argv, "mtu") == 0) {
     NEXT_ARG();
     if (mtu != -1)
      duparg("mtu", *argv);
     if (get_integer(&mtu, *argv, 0))
      invarg("Invalid \"mtu\" value\n", *argv);
-    addattr_l(&req.n, sizeof(req), IFLA_MTU, &mtu, 4);
+    addattr_l(&req->n, sizeof(*req), IFLA_MTU, &mtu, 4);
    } else if (strcmp(*argv, "multicast") == 0) {
     NEXT_ARG();
-    req.i.ifi_change |= IFF_MULTICAST;
+    req->i.ifi_change |= IFF_MULTICAST;
     if (strcmp(*argv, "on") == 0) {
-     req.i.ifi_flags |= IFF_MULTICAST;
+     req->i.ifi_flags |= IFF_MULTICAST;
     } else if (strcmp(*argv, "off") == 0) {
-     req.i.ifi_flags &= ~IFF_MULTICAST;
+     req->i.ifi_flags &= ~IFF_MULTICAST;
     } else
      return on_off("multicast");
    } else if (strcmp(*argv, "allmulticast") == 0) {
     NEXT_ARG();
-    req.i.ifi_change |= IFF_ALLMULTI;
+    req->i.ifi_change |= IFF_ALLMULTI;
     if (strcmp(*argv, "on") == 0) {
-     req.i.ifi_flags |= IFF_ALLMULTI;
+     req->i.ifi_flags |= IFF_ALLMULTI;
     } else if (strcmp(*argv, "off") == 0) {
-     req.i.ifi_flags &= ~IFF_ALLMULTI;
+     req->i.ifi_flags &= ~IFF_ALLMULTI;
     } else
      return on_off("allmulticast");
    } else if (strcmp(*argv, "promisc") == 0) {
     NEXT_ARG();
-    req.i.ifi_change |= IFF_PROMISC;
+    req->i.ifi_change |= IFF_PROMISC;
     if (strcmp(*argv, "on") == 0) {
```

```
-    req.i.ifi_flags |= IFF_PROMISC;
+    req->i.ifi_flags |= IFF_PROMISC;
     } else if (strcmp(*argv, "off") == 0) {
-    req.i.ifi_flags &= ~IFF_PROMISC;
+    req->i.ifi_flags &= ~IFF_PROMISC;
     } else
      return on_off("promisc");
   } else if (strcmp(*argv, "trailers") == 0) {
    NEXT_ARG();
-   req.i.ifi_change |= IFF_NOTRAILERS;
+   req->i.ifi_change |= IFF_NOTRAILERS;
    if (strcmp(*argv, "off") == 0) {
-    req.i.ifi_flags |= IFF_NOTRAILERS;
+    req->i.ifi_flags |= IFF_NOTRAILERS;
    } else if (strcmp(*argv, "on") == 0) {
-    req.i.ifi_flags &= ~IFF_NOTRAILERS;
+    req->i.ifi_flags &= ~IFF_NOTRAILERS;
    } else
     return on_off("trailers");
   } else if (strcmp(*argv, "arp") == 0) {
    NEXT_ARG();
-   req.i.ifi_change |= IFF_NOARP;
+   req->i.ifi_change |= IFF_NOARP;
    if (strcmp(*argv, "on") == 0) {
-    req.i.ifi_flags &= ~IFF_NOARP;
+    req->i.ifi_flags &= ~IFF_NOARP;
    } else if (strcmp(*argv, "off") == 0) {
-    req.i.ifi_flags |= IFF_NOARP;
+    req->i.ifi_flags |= IFF_NOARP;
    } else
     return on_off("noarp");
 #ifdef IFF_DYNAMIC
   } else if (matches(*argv, "dynamic") == 0) {
    NEXT_ARG();
-   req.i.ifi_change |= IFF_DYNAMIC;
+   req->i.ifi_change |= IFF_DYNAMIC;
    if (strcmp(*argv, "on") == 0) {
-    req.i.ifi_flags |= IFF_DYNAMIC;
+    req->i.ifi_flags |= IFF_DYNAMIC;
    } else if (strcmp(*argv, "off") == 0) {
-    req.i.ifi_flags &= ~IFF_DYNAMIC;
+    req->i.ifi_flags &= ~IFF_DYNAMIC;
    } else
     return on_off("dynamic");
 #endif
   } else if (matches(*argv, "type") == 0) {
    NEXT_ARG();
-   type = *argv;
```

```
+   *type = *argv;
    argc--; argv++;
    break;
   } else {
-                       if (strcmp(*argv, "dev") == 0) {
+   if (strcmp(*argv, "dev") == 0) {
     NEXT_ARG();
    }
-   if (dev)
+   if (*dev)
     duparg2("dev", *argv);
-   dev = *argv;
+   *dev = *argv;
   }
   argc--; argv++;
  }

+ return ret - argc;
+}
+
+static int iplink_modify(int cmd, unsigned int flags, int argc, char **argv)
+{
+ int len;
+ char *dev = NULL;
+ char *name = NULL;
+ char *link = NULL;
+ char *type = NULL;
+ struct link_util *lu = NULL;
+ struct iplink_req req;
+ int ret;
+
+ memset(&req, 0, sizeof(req));
+
+ req.n.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
+ req.n.nlmsg_flags = NLM_F_REQUEST|flags;
+ req.n.nlmsg_type = cmd;
+ req.i.ifi_family = preferred_family;
+
+ ret = iplink_parse(argc, argv, &req, &name, &type, &link, &dev);
+ if (ret < 0)
+  return ret;
+
+ argc -= ret;
+ argv += ret;
  ll_init_map(&rth);

  if (type) {
```

Subject: [PATCH 2/2] Module for ip utility to support veth device
Posted by Pavel Emelianov on Thu, 19 Jul 2007 09:33:56 GMT
View Forum Message <> Reply to Message

The link_veth.so itself.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
Acked-by: Patrick McHardy <kaber@trash.net>

 Makefile | 6 ++++-
 link_veth.c | 63 ++
 veth.h | 12 ++++++++++
 3 files changed, 80 insertions(+), 1 deletion(-)

```
diff --git a/ip/Makefile b/ip/Makefile
index 9a5bfe3..b46bce3 100644
--- a/ip/Makefile
+++ b/ip/Makefile
@@ -8,8 +8,9 @@ RTMONOBJ=rtmon.o
 ALLOBJ=$(IPOBJ) $(RTMONOBJ)
 SCRIPTS=ifcfg rtpr routel routef
 TARGETS=ip rtmon
+LIBS=link_veth.so

-all: $(TARGETS) $(SCRIPTS)
+all: $(TARGETS) $(SCRIPTS) $(LIBS)

 ip: $(IPOBJ) $(LIBNETLINK) $(LIBUTIL)

@@ -24,3 +25,6 @@ clean:

 LDLIBS += -ldl
 LDFLAGS += -Wl,-export-dynamic
+
+%.so: %.c
+ $(CC) $(CFLAGS) -shared $< -o $@
diff --git a/ip/link_veth.c b/ip/link_veth.c
new file mode 100644
index 0000000..ded2cdd
--- /dev/null
+++ b/ip/link_veth.c
@@ -0,0 +1,63 @@
+/*
+ * link_veth.c veth driver module
+ *
+ *  This program is free software; you can redistribute it and/or
+ *  modify it under the terms of the GNU General Public License
```

```
+ *  as published by the Free Software Foundation; either version
+ *  2 of the License, or (at your option) any later version.
+ *
+ * Authors: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+#include <string.h>
+
+#include "utils.h"
+#include "ip_common.h"
+#include "veth.h"
+
+#define IFNAMSIZ 16
+
+static void usage(void)
+{
+ printf("Usage: ip link add ... type veth "
+   "[peer <peer-name>] [mac <mac>] [peer_mac <mac>]\n");
+}
+
+static int veth_parse_opt(struct link_util *lu, int argc, char **argv,
+  struct nlmsghdr *hdr)
+{
+ char *name, *type, *link, *dev;
+ int err, len;
+ struct rtattr * data;
+
+ if (strcmp(argv[0], "peer") != 0) {
+  usage();
+  return -1;
+ }
+
+ data = NLMSG_TAIL(hdr);
+ addattr_l(hdr, 1024, VETH_INFO_PEER, NULL, 0);
+
+ hdr->nlmsg_len += sizeof(struct ifinfomsg);
+
+ err = iplink_parse(argc - 1, argv + 1, (struct iplink_req *)hdr,
+   &name, &type, &link, &dev);
+ if (err < 0)
+  return err;
+
+ if (name) {
+  len = strlen(name) + 1;
+  if (len > IFNAMSIZ)
+   invarg("\"name\" too long\n", *argv);
+  addattr_l(hdr, 1024, IFLA_IFNAME, name, len);
```

```
+ }
+
+ data->rta_len = (void *)NLMSG_TAIL(hdr) - (void *)data;
+ return argc - 1 - err;
+}
+
+struct link_util veth_link_util = {
+ .id = "veth",
+ .parse_opt = veth_parse_opt,
+};
diff --git a/ip/veth.h b/ip/veth.h
new file mode 100644
index 0000000..aa2e6f9
--- /dev/null
+++ b/ip/veth.h
@@ -0,0 +1,12 @@
+#ifndef __NET_VETH_H__
+#define __NET_VETH_H__
+
+enum {
+ VETH_INFO_UNSPEC,
+ VETH_INFO_PEER,
+
+ __VETH_INFO_MAX
+#define VETH_INFO_MAX (__VETH_INFO_MAX - 1)
+};
+
+#endif
```

Subject: Re: [PATCH] Virtual ethernet device (v.4)
Posted by Pavel Emelianov on Wed, 08 Aug 2007 10:42:50 GMT
View Forum Message <> Reply to Message

Hi, David.

What are your plans about this driver?

Thanks,
Pavel

> Veth stands for Virtual ETHernet. It is a simple tunnel driver
> that works at the link layer and looks like a pair of ethernet
> devices interconnected with each other.
>
> Mainly it allows to communicate between network namespaces but
> it can be used as is as well. E.g. one may join to bridged
> networking segments together.

>
> Eric recently sent a similar driver called etun with the sysfs
> interface. This implementation uses another interface - the RTM_NRELINK
> message introduced by Patric.
>
> The newlink callback is organized that way to make it easy to create the
> peer device in the separate namespace when we have them in kernel.
>
> Many thanks to Patrick for reviewing the patches and his advises
> on how to make driver cleaner.
>
> Changes from v.3:
> * Reserved place for struct ifinfomsg in IFLA_INFO_DATA part
> of the packet. This is not used yet, but may be in the
> future.
>
> Changes from v.2.1:
> * Made the generic routine for link creation to be used
> by veth driver, any other tunnel driver that needs to
> create several devices at once and rtnl_newlink() code.
>
> Changes from v.2:
> * Rebase over latest netdev tree. No actual changes;
> * Small code rework.
>
> Changes from v.1:
> * Per-cpu statistics;
> * Standard convention for nla policy names;
> * Module alias added;
> * Xmit function fixes noticed by Patric;
> * Code cleanup.
>
> The patch for an ip utility is also provided.
>
> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
> Acked-by: Patrick McHardy <kaber@trash.net>
>

Subject: Re: [PATCH] Virtual ethernet device (v.4)
Posted by davem on Thu, 09 Aug 2007 05:18:27 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Wed, 08 Aug 2007 14:42:50 +0400

> What are your plans about this driver?

I've added it to my net-2.6.24 which I'm building to night.