

---

Subject: [PATCH] Virtual ethernet tunnel

Posted by [Pavel Emelianov](#) on Wed, 06 Jun 2007 15:11:38 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.

Eric recently sent a similar driver called etun. This implementation uses another interface - the RTM\_NRELINK message introduced by Patric. The patch fits today netdev tree with Patrick's patches.

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.

The patch for an ip utility is also provided.

Eric, since ethtool interface was from your patch, I add your Signed-off-by line.

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

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

---

```
diff --git a/drivers/net/Kconfig b/drivers/net/Kconfig
index 7d57f4a..7e144be 100644
--- a/drivers/net/Kconfig
+++ b/drivers/net/Kconfig
@@ -119,6 +119,12 @@ @@ 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 a77affa..4764119 100644
--- a/drivers/net/Makefile
+++ b/drivers/net/Makefile
@@ -185,6 +185,7 @@ 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..6746c91
--- /dev/null
+++ b/drivers/net/veth.c
@@ -0,0 +1,391 @@
+/*
+ * drivers/net/veth.c
+ *
+ * Copyright (C) 2007 OpenVZ http://openvz.org, SWsoft Inc
+ *
+ */
+
+#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_priv {
+ struct net_device *peer;
+ struct net_device *dev;
+ struct list_head list;
+ struct net_device_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;
+ int length;
+
+ skb_orphan(skb);
+
+ priv = netdev_priv(dev);
+ rcv = priv->peer;
+ rcv_priv = netdev_priv(rcv);
+
+ if (!(rcv->flags & IFF_UP))
+ goto outf;
+
+ skb->dev = rcv;
+ 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;
+
+ secpath_reset(skb);
+ nf_reset(skb);
+
+ length = skb->len;
+
+ priv->stats.tx_bytes += length;
+ priv->stats.tx_packets++;
+
+ rcv_priv->stats.rx_bytes += length;
+ rcv_priv->stats.rx_packets++;
+
+ netif_rx(skb);
+ return 0;
+
+outf:
+ kfree_skb(skb);

```

```

+ priv->stats.tx_dropped++;
+ return 0;
+}
+
+/*
+ * general routines
+ */
+
+static struct net_device_stats *veth_get_stats(struct net_device *dev)
+{
+ struct veth_priv *priv;
+
+ priv = netdev_priv(dev);
+ return &priv->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;
+}
+
+/*
+ * netlink interface
+ */
+
+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->destructor = free_netdev;
+ dev->ethtool_ops = &veth_ethtool_ops;
+ dev->features |= NETIF_F_LLTX;
+ netif_carrier_off(dev);
+}
+
+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];
+
+ /*
+  * setup the first device
+  */
+
+ if (data != NULL && data[VETH_INFO_MAC] != NULL)
+ memcpy(dev->dev_addr,
+ nla_data(data[VETH_INFO_MAC]), ETH_ALEN);
+ else
+ random_ether_addr(dev->dev_addr);
+
+ err = register_netdevice(dev);
+ if (err < 0)
+ goto err_register_dev;
+
+ /*
+  * alloc and setup the second one
+  *
+  * this should be done in another namespace, but we
+  * do not have them yet
+  */
+
+ if (data != NULL && data[VETH_INFO_PEER] != NULL)
+ nla_strncpy(ifname, data[VETH_INFO_PEER], IFNAMSIZ);
+ else
+ snprintf(ifname, IFNAMSIZ, "veth%%d");
+
+ err = -ENOMEM;

```

```

+ peer = alloc_netdev(sizeof(struct veth_priv), ifname, veth_setup);
+ if (peer == NULL)
+ goto err_alloc;
+
+ if (strchr(peer->name, '%')) {
+ err = dev_alloc_name(peer, peer->name);
+ if (err < 0)
+ goto err_name;
+ }
+
+ if (data != NULL && data[VETH_INFO_PEER_MAC] != NULL)
+ memcpy(peer->dev_addr,
+ nla_data(data[VETH_INFO_PEER_MAC]), ETH_ALEN);
+ else
+ random_ether_addr(peer->dev_addr);
+
+ /* this should be in sync with rtnl_newlink */
+ peer->mtu = dev->mtu;
+ peer->tx_queue_len = dev->tx_queue_len;
+ peer->weight = dev->weight;
+ peer->link_mode = dev->link_mode;
+ peer->rtnl_link_ops = dev->rtnl_link_ops;
+
+ if (peer->operstate != dev->operstate) {
+ write_lock_bh(&dev_base_lock);
+ peer->operstate = dev->operstate;
+ write_unlock_bh(&dev_base_lock);
+ netdev_state_change(peer);
+ }
+
+ err = register_netdevice(peer);
+ if (err < 0)
+ goto err_register_peer;
+
+ /*
+  * tie the devices 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_peer:
+ /* nothing special to do */
+err_name:
+ free_netdev(peer);
+err_alloc:
+ unregister_netdevice(dev);
+err_register_dev:
+ 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] = {
+ [VETH_INFO_MAC] = { .type = NLA_BINARY, .len = ETH_ALEN },
+ [VETH_INFO_PEER] = { .type = NLA_STRING },
+ [VETH_INFO_PEER_MAC] = { .type = NLA_BINARY, .len = ETH_ALEN },
+};
+
+static struct rtnl_link_ops veth_link_ops = {
+ .name = "veth",
+ .priv_size = sizeof(struct veth_priv),
+ .setup = veth_setup,
+ .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();
+ __rtnl_link_unregister(&veth_link_ops);
+
+ list_for_each_entry_safe(priv, next, &veth_list, list)
+ veth_dellink(priv->dev);
+ rtnl_unlock();
+}
+
+module_init(veth_init);
+module_exit(veth_exit);
+
+MODULE_DESCRIPTION("Virtual Ethernet Tunnel");
+MODULE_LICENSE("GPL v2");
diff --git a/include/net/veth.h b/include/net/veth.h
new file mode 100644
index 00000000..74c8e1e
--- /dev/null
+++ b/include/net/veth.h
@@ -0,0 +1,13 @@
+#ifndef __NET_VETH_H__
+#define __NET_VETH_H__
+
+enum {
+ VETH_INFO_UNSPEC,
+ VETH_INFO_MAC,
+ VETH_INFO_PEER,
+ VETH_INFO_PEER_MAC,
+
+ VETH_INFO_MAX
+};
+
+#endif

```

---

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