
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 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_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

```
