Subject: [PATCH] Virtual ethernet device (tunnel)
Posted by xemul on Wed, 02 May 2007 10:51:28 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 is closer to the OpenVZ one and it lacks
some unimportant features of etun driver (like ethtool_ops)
for simplicity.

The general difference from etun is that a netlink interface
is used to create and destroy the pairs. The patch for an
ip utility is also provided.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Acked-By: Kirill Korotaev <dev@sw.ru>
Acked-By: Dmitry Mishin <dim@openvz.org>
Acked-By: Alexey Kuznetsov <kuznet@ms2.inr.ac.ru>

---

```
diff --git a/drivers/net/Kconfig b/drivers/net/Kconfig
index c3f9f59..445dbc7 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 33af833..2730b80 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..6105f99
--- /dev/null
+++ b/drivers/net/veth.c
@@ -0,0 +1,387 @@
+/*
+ *  drivers/net/veth.c
+ *
+ *  Copyright (C) 2007 OpenVZ http://openvz.org, SWsoft Inc
+ *
+ */
+
+#include <linux/init.h>
+#include <linux/if.h>
+#include <linux/netdevice.h>
+#include <linux/etherdevice.h>
+#include <net/dst.h>
+#include <net/xfrm.h>
+#include <net/genetlink.h>
+#include <net/veth.h>
+
+struct veth_struct {
+ struct net_device *peer;
+ struct net_device *dev;
+
+ struct list_head list;
+ struct net_device_stats *real_stats;
+
+ struct net_device_stats stats;
+};
+
+static LIST_HEAD(veth_list);
+
+static inline struct net_device_stats *veth_stats(struct veth_struct *veth,
+  int cpuid)
+{
+ return per_cpu_ptr(veth->real_stats, cpuid);
+}
+
```

```
+/*
+ * Device functions
+ */
+
+static int veth_open(struct net_device *dev)
+{
+ return 0;
+}
+
+static int veth_close(struct net_device *dev)
+{
+ return 0;
+}
+
+static void veth_destructor(struct net_device *dev)
+{
+ struct veth_struct *veth;
+
+ veth = dev->priv;
+ free_percpu(veth->real_stats);
+ free_netdev(dev);
+}
+
+static struct net_device_stats *veth_get_stats(struct net_device *dev)
+{
+ int i;
+ struct veth_struct *veth;
+ struct net_device_stats *stats;
+ struct net_device_stats *dev_stats;
+
+ veth = dev->priv;
+ stats = &veth->stats;
+ memset(stats, 0, sizeof(struct net_device_stats));
+
+ for_each_possible_cpu (i) {
+  dev_stats = veth_stats(veth, i);
+  stats->rx_bytes   += dev_stats->rx_bytes;
+  stats->tx_bytes   += dev_stats->tx_bytes;
+  stats->rx_packets += dev_stats->rx_packets;
+  stats->tx_packets += dev_stats->tx_packets;
+ }
+
+ return stats;
+}
+
+static int veth_xmit(struct sk_buff *skb, struct net_device *dev)
+{
+ struct net_device_stats *stats;
```

```
+ struct net_device *rcv = NULL;
+ struct veth_struct *veth;
+ int length, cpu;
+
+ skb_orphan(skb);
+
+ veth = dev->priv;
+ rcv = veth->peer;
+
+ cpu = smp_processor_id();
+ stats = veth_stats(veth, cpu);
+
+ if (!(rcv->flags & IFF_UP))
+  goto outf;
+
+ skb->dev = rcv;
+ skb->pkt_type = PACKET_HOST;
+ skb->protocol = eth_type_trans(skb, rcv);
+
+ dst_release(skb->dst);
+ skb->dst = NULL;
+
+ secpath_reset(skb);
+ nf_reset(skb);
+
+ length = skb->len;
+
+ stats->tx_bytes += length;
+ stats->tx_packets++;
+
+ stats = veth_stats(rcv->priv, cpu);
+ stats->rx_bytes += length;
+ stats->rx_packets++;
+
+ netif_rx(skb);
+ return 0;
+
+outf:
+ kfree_skb(skb);
+ stats->tx_dropped++;
+ return 0;
+}
+
+/*
+ * Setup / remove routines
+ */
+
+static int veth_init_dev(struct net_device *dev)
```

```
+{
+ struct veth_struct *veth;
+
+ dev->hard_start_xmit = veth_xmit;
+ dev->get_stats = veth_get_stats;
+ dev->open = veth_open;
+ dev->stop = veth_close;
+ dev->destructor = veth_destructor;
+
+ veth = dev->priv;
+ veth->real_stats = alloc_percpu(struct net_device_stats);
+ if (veth->real_stats == NULL)
+  return -ENOMEM;
+
+ return 0;
+}
+
+static void veth_setup(struct net_device *dev)
+{
+ ether_setup(dev);
+
+ dev->init = veth_init_dev;
+ dev->addr_len = ETH_ALEN;
+ dev->features |= NETIF_F_LLTX;
+ dev->tx_queue_len = 0;
+ random_ether_addr(dev->dev_addr);
+}
+
+struct net_device *veth_dev_start(char *name)
+{
+ struct net_device *dev;
+ int err;
+
+ err = -ENOMEM;
+ dev = alloc_netdev(sizeof(struct veth_struct), name, veth_setup);
+ if (!dev)
+  goto err_alloc;
+
+ err = register_netdev(dev);
+ if (err != 0)
+  goto err;
+
+ return dev;
+
+err:
+ free_netdev(dev);
+err_alloc:
+ return ERR_PTR(err);
```

```
+}
+
+static int veth_create_pair(char *name, char *peer_name)
+{
+ struct net_device *dev;
+ struct net_device *peer;
+ struct veth_struct *dev_veth;
+ struct veth_struct *peer_veth;
+ int err;
+
+ dev = veth_dev_start(name);
+ if (IS_ERR(dev)) {
+  err = PTR_ERR(dev);
+  goto err;
+ }
+
+ peer = veth_dev_start(peer_name);
+ if (IS_ERR(peer)) {
+  err = PTR_ERR(peer);
+  goto err_peer;
+ }
+
+ dev_veth = dev->priv;
+ peer_veth = peer->priv;
+
+ dev_veth->peer = peer;
+ dev_veth->dev = dev;
+ peer_veth->peer = dev;
+ peer_veth->dev = peer;
+
+ rtnl_lock();
+ list_add(&dev_veth->list, &veth_list);
+ INIT_LIST_HEAD(&peer_veth->list);
+ rtnl_unlock();
+ return 0;
+
+err_peer:
+ unregister_netdev(dev);
+err:
+ return err;
+}
+
+static void veth_dev_stop(struct net_device *dev)
+{
+ struct net_device *peer;
+ struct veth_struct *dev_veth;
+ struct veth_struct *peer_veth;
+
```

```
+ dev_veth = dev->priv;
+ peer = dev_veth->peer;
+ peer_veth = peer->priv;
+
+ /*
+  * Since we obtain the device by name, we can have dev point to the
+  * device that was 'peer' during creation. So check for list_empty
+  * before removing.
+  */
+ if (!list_empty(&dev_veth->list))
+  list_del(&dev_veth->list);
+ if (!list_empty(&peer_veth->list))
+  list_del(&peer_veth->list);
+
+ dev_close(dev);
+ dev_close(peer);
+
+ unregister_netdevice(peer);
+ unregister_netdevice(dev);
+}
+
+static int veth_destroy_pair(char *name)
+{
+ struct net_device *dev;
+ int err;
+
+ err = -ENODEV;
+ rtnl_lock();
+ dev = __dev_get_by_name(name);
+ if (dev == NULL)
+  goto out;
+
+ err = 0;
+ veth_dev_stop(dev);
+out:
+ rtnl_unlock();
+ return err;
+}
+
+/*
+ * Netlink interface
+ */
+
+static int veth_get_name(struct nlattr *na, char *name)
+{
+ int len;
+
+ if (na == NULL)
```

```
+  return -ENOENT;
+
+ len = nla_len(na);
+ if (len > IFNAMSIZ)
+  return -E2BIG;
+ if (len < 1)
+  return -EINVAL;
+
+ nla_strlcpy(name, na, len);
+ return 0;
+}
+
+static int veth_add(struct sk_buff *skb, struct genl_info *info)
+{
+ int err;
+ char name[IFNAMSIZ], peer[IFNAMSIZ];
+
+ err = veth_get_name(info->attrs[VETH_ATTR_DEVNAME], name);
+ if (err < 0)
+  goto out;
+
+ err = veth_get_name(info->attrs[VETH_ATTR_PEERNAME], peer);
+ if (err < 0)
+  goto out;
+
+ err = veth_create_pair(name, peer);
+out:
+ return err;
+}
+
+static int veth_del(struct sk_buff *skb, struct genl_info *info)
+{
+ int err;
+ char name[IFNAMSIZ];
+
+ err = veth_get_name(info->attrs[VETH_ATTR_DEVNAME], name);
+ if (err < 0)
+  goto out;
+
+ err = veth_destroy_pair(name);
+out:
+ return err;
+}
+
+static struct nla_policy veth_policy[VETH_ATTR_MAX] = {
+ [VETH_ATTR_DEVNAME] = { .type = NLA_STRING },
+ [VETH_ATTR_PEERNAME] = { .type = NLA_STRING },
+};
```

```
+
+static struct genl_ops veth_add_ops = {
+ .cmd = VETH_CMD_ADD,
+ .doit = veth_add,
+ .policy = veth_policy,
+};
+
+static struct genl_ops veth_del_ops = {
+ .cmd = VETH_CMD_DEL,
+ .doit = veth_del,
+ .policy = veth_policy,
+};
+
+static struct genl_family veth_family = {
+ .id  = GENL_ID_GENERATE,
+ .name  = "veth",
+ .version = 0x1,
+ .maxattr = 2,
+};
+
+static __init int veth_init(void)
+{
+ int err;
+
+ err = genl_register_family(&veth_family);
+ if (err < 0)
+  goto out;
+
+ err = genl_register_ops(&veth_family, &veth_add_ops);
+ if (err < 0)
+  goto out_unregister_fam;
+
+ err = genl_register_ops(&veth_family, &veth_del_ops);
+ if (err < 0)
+  goto out_unregister_add;
+
+ return 0;
+
+out_unregister_add:
+ genl_unregister_ops(&veth_family, &veth_add_ops);
+out_unregister_fam:
+ genl_unregister_family(&veth_family);
+out:
+ return err;
+}
+
+static __exit void veth_exit(void)
+{
```

```
+ struct veth_struct *veth, *tmp;
+
+ genl_unregister_ops(&veth_family, &veth_del_ops);
+ genl_unregister_ops(&veth_family, &veth_add_ops);
+ genl_unregister_family(&veth_family);
+
+ rtnl_lock();
+ list_for_each_entry_safe (veth, tmp, &veth_list, list)
+  veth_dev_stop(veth->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 0000000..ef21713
--- /dev/null
+++ b/include/net/veth.h
@@ -0,0 +1,20 @@
+#ifndef __NET_VETH_H__
+#define __NET_VETH_H__
+
+enum {
+ VETH_CMD_UNSPEC,
+ VETH_CMD_ADD,
+ VETH_CMD_DEL,
+
+ VETH_CMD_MAX
+};
+
+enum {
+ VETH_ATTR_UNSPEC,
+ VETH_ATTR_DEVNAME,
+ VETH_ATTR_PEERNAME,
+
+ VETH_ATTR_MAX
+};
+
+#endif
```