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
```

Subject: [PATCH] Make ip utility veth driver aware
Posted by xemul on Wed, 02 May 2007 10:55:10 GMT

The new command is called "veth" with the following syntax:
* ip veth add <dev1> <dev2>
  creates interconnected pair of veth devices.
* ip veth del <dev>
  destroys the pair of veth devices, where <dev> is either
  <dev1> or <dev2> used to create the pair.

One question that is to be solved is whether or not to create
a hard-coded netlink family for veth driver. Without it the
family resolution code has to be moved to general place in ip
utility (by now it is copy-paste-ed from one file to another
till final decision).

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

```
diff --git a/ip/Makefile b/ip/Makefile
index a749993..864e04e 100644
--- a/ip/Makefile
+++ b/ip/Makefile
@@ -1,7 +1,7 @@
 IPOBJ=ip.o ipaddress.o iproute.o iprule.o \
     rtm_map.o iptunnel.o ip6tunnel.o tunnel.o ipneigh.o ipntable.o iplink.o \
     ipmaddr.o ipmonitor.o ipmroute.o ipprefix.o \
-    ipxfrm.o xfrm_state.o xfrm_policy.o xfrm_monitor.o
+    ipxfrm.o xfrm_state.o xfrm_policy.o xfrm_monitor.o veth.o

 RTMONOBJ=rtmon.o

diff --git a/ip/ip.c b/ip/ip.c
index c084292..d3d3a8a 100644
--- a/ip/ip.c
+++ b/ip/ip.c
@@ -27,6 +27,7 @@
 #include "SNAPSHOT.h"
 #include "utils.h"
 #include "ip_common.h"
+#include "veth.h"

 int preferred_family = AF_UNSPEC;
 int show_stats = 0;
@@ -46,7 +47,7 @@ static void usage(void)
 "Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }\n"
 "       ip [ -force ] [-batch filename\n"
 "where  OBJECT := { link | addr | route | rule | neigh | ntable | tunnel |\n"
-"                   maddr | mroute | monitor | xfrm }\n"
```

```
+"                  maddr | mroute | monitor | xfrm | veth }\n"
 "       OPTIONS := { -V[ersion] | -s[tatistics] | -r[esolve] |\n"
 "                  -f[amily] { inet | inet6 | ipx | dnet | link } |\n"
 "                  -o[neline] | -t[imestamp] }\n");
@@ -76,6 +77,7 @@ static const struct cmd {
  { "monitor", do_ipmonitor },
  { "xfrm", do_xfrm },
  { "mroute", do_multiroute },
+ { "veth", do_veth },
  { "help", do_help },
  { 0 }
 };
diff --git a/ip/veth.c b/ip/veth.c
new file mode 100644
index 0000000..d4eecc8
--- /dev/null
+++ b/ip/veth.c
@@ -0,0 +1,196 @@
+/*
+ * veth.c        "ethernet tunnel"
+ *
+ *  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 <stdio.h>
+#include <string.h>
+#include <unistd.h>
+#include <sys/types.h>
+#include <sys/socket.h>
+#include <linux/genetlink.h>
+
+#include "utils.h"
+#include "veth.h"
+
+#define GENLMSG_DATA(glh)       ((void *)(NLMSG_DATA(glh) + GENL_HDRLEN))
+#define NLA_DATA(na)            ((void *)((char*)(na) + NLA_HDRLEN))
+
+static int do_veth_help(void)
+{
+ fprintf(stderr, "Usage: ip veth add DEVICE PEER_NAME\n");
+ fprintf(stderr, "               del DEVICE\n");
+ exit(-1);
```

```
+}
+
+static int genl_ctrl_resolve_family(const char *family)
+{
+ struct rtnl_handle rth;
+ struct nlmsghdr *nlh;
+ struct genlmsghdr *ghdr;
+ int ret = 0;
+ struct {
+  struct nlmsghdr         n;
+  char                    buf[4096];
+ } req;
+
+ memset(&req, 0, sizeof(req));
+
+ nlh = &req.n;
+ nlh->nlmsg_len = NLMSG_LENGTH(GENL_HDRLEN);
+ nlh->nlmsg_flags = NLM_F_REQUEST | NLM_F_ACK;
+ nlh->nlmsg_type = GENL_ID_CTRL;
+
+ ghdr = NLMSG_DATA(&req.n);
+ ghdr->cmd = CTRL_CMD_GETFAMILY;
+
+ if (rtnl_open_byproto(&rth, 0, NETLINK_GENERIC) < 0) {
+  fprintf(stderr, "Cannot open generic netlink socket\n");
+  exit(1);
+ }
+
+ addattr_l(nlh, 128, CTRL_ATTR_FAMILY_NAME, family, strlen(family) + 1);
+
+ if (rtnl_talk(&rth, nlh, 0, 0, nlh, NULL, NULL) < 0) {
+  fprintf(stderr, "Error talking to the kernel\n");
+  goto errout;
+ }
+
+ {
+  struct rtattr *tb[CTRL_ATTR_MAX + 1];
+  struct genlmsghdr *ghdr = NLMSG_DATA(nlh);
+  int len = nlh->nlmsg_len;
+  struct rtattr *attrs;
+
+  if (nlh->nlmsg_type !=  GENL_ID_CTRL) {
+   fprintf(stderr, "Not a controller message, nlmsg_len=%d "
+    "nlmsg_type=0x%x\n", nlh->nlmsg_len, nlh->nlmsg_type);
+   goto errout;
+  }
+
+  if (ghdr->cmd != CTRL_CMD_NEWFAMILY) {
```

```
+   fprintf(stderr, "Unkown controller command %d\n", ghdr->cmd);
+   goto errout;
+  }
+
+  len -= NLMSG_LENGTH(GENL_HDRLEN);
+
+  if (len < 0) {
+   fprintf(stderr, "wrong controller message len %d\n", len);
+   return -1;
+  }
+
+  attrs = (struct rtattr *) ((char *) ghdr + GENL_HDRLEN);
+  parse_rtattr(tb, CTRL_ATTR_MAX, attrs, len);
+
+  if (tb[CTRL_ATTR_FAMILY_ID] == NULL) {
+   fprintf(stderr, "Missing family id TLV\n");
+   goto errout;
+  }
+
+  ret = *(__u16 *) RTA_DATA(tb[CTRL_ATTR_FAMILY_ID]);
+ }
+
+errout:
+ rtnl_close(&rth);
+ return ret;
+}
+
+static int do_veth_operate(char *dev, char *peer, int cmd)
+{
+ struct rtnl_handle rth;
+ struct nlmsghdr *nlh;
+ struct genlmsghdr *ghdr;
+ struct nlattr *attr;
+ struct  {
+   struct nlmsghdr n;
+   struct genlmsghdr h;
+   char bug[1024];
+ } req;
+ int family, len;
+ int err = 0;
+
+ family = genl_ctrl_resolve_family("veth");
+ if (family == 0) {
+   fprintf(stderr, "veth: Can't resolve family\n");
+   exit(1);
+ }
+
+ if (rtnl_open_byproto(&rth, 0, NETLINK_GENERIC) < 0)
```

```
+  exit(1);
+
+ nlh = &req.n;
+ nlh->nlmsg_len = NLMSG_LENGTH(GENL_HDRLEN);
+ nlh->nlmsg_flags = NLM_F_REQUEST;
+ nlh->nlmsg_type = family;
+ nlh->nlmsg_seq = 0;
+
+ ghdr = &req.h;
+ ghdr->cmd = cmd;
+
+ attr = (struct nlattr *) GENLMSG_DATA(&req);
+ len = strlen(dev);
+ attr->nla_type = VETH_ATTR_DEVNAME;
+ attr->nla_len = len + 1 + NLA_HDRLEN;
+ memcpy(NLA_DATA(attr), dev, len);
+ nlh->nlmsg_len += NLMSG_ALIGN(attr->nla_len);
+
+ if (peer) {
+  attr = (struct nlattr *)((char *)attr +
+    NLMSG_ALIGN(attr->nla_len));
+  len = strlen(peer);
+  attr->nla_type = VETH_ATTR_PEERNAME;
+  attr->nla_len = len + 1 + NLA_HDRLEN;
+  memcpy(NLA_DATA(attr), peer, len);
+  nlh->nlmsg_len += NLMSG_ALIGN(attr->nla_len);
+ }
+
+ if (rtnl_send(&rth, (char *) &req, nlh->nlmsg_len) < 0) {
+  err = -1;
+  fprintf(stderr, "Error talking to the kernel (add)\n");
+ }
+
+ rtnl_close(&rth);
+ return err;
+}
+
+static int do_veth_add(int argc, char **argv)
+{
+ if (argc < 2)
+  return do_veth_help();
+
+ return do_veth_operate(argv[0], argv[1], VETH_CMD_ADD);
+}
+
+static int do_veth_del(int argc, char **argv)
+{
+ char *name;
```

```
+
+ if (argc < 1)
+  return do_veth_help();
+
+ return do_veth_operate(argv[0], NULL, VETH_CMD_DEL);
+}
+
+int do_veth(int argc, char **argv)
+{
+ if (argc == 0)
+  return do_veth_help();
+
+ if (strcmp(*argv, "add") == 0 || strcmp(*argv, "a") == 0)
+  return do_veth_add(argc - 1, argv + 1);
+ if (strcmp(*argv, "del") == 0 || strcmp(*argv, "d") == 0)
+  return do_veth_del(argc - 1, argv + 1);
+ if (strcmp(*argv, "help") == 0)
+  return do_veth_help();
+
+ fprintf(stderr, "Command \"%s\" is unknown, try \"ip veth help\".\n", *argv);
+ exit(-1);
+}
diff --git a/ip/veth.h b/ip/veth.h
new file mode 100644
index 0000000..4d7b357
--- /dev/null
+++ b/ip/veth.h
@@ -0,0 +1,17 @@
+int do_veth(int argc, char **argv);
+
+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
+};
```

Subject: Re: [PATCH] Virtual ethernet device (tunnel)
Posted by Daniel Lezcano on Wed, 02 May 2007 12:07:26 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> 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.
if etun and veth are similar, why didn't you put the netlink interface
to the etun driver instead of sending a new driver ?

-- Daniel

Subject: Re: [PATCH] Virtual ethernet device (tunnel)
Posted by xemul on Wed, 02 May 2007 12:23:27 GMT
View Forum Message <> Reply to Message

Daniel Lezcano wrote:
> Pavel Emelianov wrote:
>> 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.
> if etun and veth are similar, why didn't you put the netlink interface

> to the etun driver instead of sending a new driver ?

Alexey said, that he preferred the name "veth" to "etun".
So an incremental patch would look too bad.

> -- Daniel
> -
> To unsubscribe from this list: send the line "unsubscribe netdev" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
>

Subject: Re: [PATCH] Virtual ethernet device (tunnel)
Posted by Patrick McHardy on Wed, 02 May 2007 12:34:42 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> 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.

Thats a lot better than using sysfs, but I think it would be
preferrable to use rtnetlink instead of genetlink for network
configuration.

Subject: Re: [PATCH] Virtual ethernet device (tunnel)
Posted by jamal on Wed, 02 May 2007 12:49:52 GMT
View Forum Message <> Reply to Message

On Wed, 2007-02-05 at 14:34 +0200, Patrick McHardy wrote:

> Thats a lot better than using sysfs, but I think it would be

> preferrable to use rtnetlink instead of genetlink for network
> configuration.

or you can just hold rtnl while using genl.
I do agree it would be easier to just use rtnetlink ...

cheers,
jamal

Subject: Re: [PATCH] Virtual ethernet device (tunnel)
Posted by Patrick McHardy on Wed, 02 May 2007 12:59:05 GMT
View Forum Message <> Reply to Message

jamal wrote:
> On Wed, 2007-02-05 at 14:34 +0200, Patrick McHardy wrote:
>
>
>>Thats a lot better than using sysfs, but I think it would be
>>preferrable to use rtnetlink instead of genetlink for network
>>configuration.
>
>
> or you can just hold rtnl while using genl.
> I do agree it would be easier to just use rtnetlink ...

The rtnl needs to be held in either case, but using a different
netlink family introduces races in message processing. For example
a simple:

ip link add dev veth0
ip route add 10.0.0.0/8 dev veth0

might fail because we have two different input queues and the routing
message might get processed before the link message.

Subject: Re: [PATCH] Virtual ethernet device (tunnel)
Posted by ebiederm on Wed, 02 May 2007 13:40:31 GMT
View Forum Message <> Reply to Message

Patrick McHardy <kaber@trash.net> writes:

> jamal wrote:
>> On Wed, 2007-02-05 at 14:34 +0200, Patrick McHardy wrote:
>>

>>
>>>Thats a lot better than using sysfs, but I think it would be
>>>preferrable to use rtnetlink instead of genetlink for network
>>>configuration.
>>
>>
>> or you can just hold rtnl while using genl.
>> I do agree it would be easier to just use rtnetlink ...
>
>
> The rtnl needs to be held in either case, but using a different
> netlink family introduces races in message processing. For example
> a simple:
>
> ip link add dev veth0
> ip route add 10.0.0.0/8 dev veth0
>
> might fail because we have two different input queues and the routing
> message might get processed before the link message.

The consensus from the last thread was pretty much that we need
to implement RTM_NEWLINK and RTM_DELLINK, if it is at all possible.

So that we can get code reuse between different virtual devices.
Although I suspect we will need some per type attribute parsing.

Eric

Subject: Re: [PATCH] Virtual ethernet device (tunnel)
Posted by Patrick McHardy on Wed, 02 May 2007 13:57:10 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> The consensus from the last thread was pretty much that we need
> to implement RTM_NEWLINK and RTM_DELLINK, if it is at all possible.

Yes, as I said, I can take care of this for 2.6.23.

> So that we can get code reuse between different virtual devices.
> Although I suspect we will need some per type attribute parsing.

Yes, we could use IFLA_PROTINFO, but since some devices (for example
bridging) already use a non-nested attribute for this when sending
notifications, its probably better to introduce a new attribute for
device specific stuff to keep things symetrical.

Subject: Re: [PATCH] Virtual ethernet device (tunnel)
Posted by Stephen Hemminger on Wed, 02 May 2007 16:37:02 GMT
View Forum Message <> Reply to Message

On Wed, 02 May 2007 14:54:51 +0400
Pavel Emelianov <xemul@sw.ru> wrote:

> 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 ethtool stuff lets bonding and bridging work better.

Subject: Re: [PATCH] Virtual ethernet device (tunnel)
Posted by Ben Greear on Wed, 02 May 2007 16:50:48 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> 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>
>

In your veth_xmit method, do you perhaps also need to initialize
skb->mark to zero?

> +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;
>
It looks like you add it to netdev list before you set up the peer, so
it could race
and crash on stale/null peer pointers?

> +
> + rtnl_lock();
> + list_add(&dev_veth->list, &veth_list);
> + INIT_LIST_HEAD(&peer_veth->list);
> + rtnl_unlock();
> + return 0;
>
Seems to me you might want get all of your state and internal list setup
completed before you register it with the
netdev list, though I'm not sure it matters in this case.

--

Ben Greear <greearb@candelatech.com>
Candela Technologies Inc http://www.candelatech.com