Subject: [PATCH] Virtual ethernet device (v2.1)
Posted by Pavel Emelianov on Wed, 11 Jul 2007 09:31:15 GMT
View Forum Message <> Reply to Message

 == =
Since the netlink NEWLINK interface is now in the netdev tree
I resend the veth driver patch as "submittion for inclusion".
 == =

LOG:

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 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.

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

Changes from v.1:
 * percpu 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>

---

 drivers/net/Kconfig | 6
 drivers/net/Makefile | 1
 drivers/net/veth.c | 452 ++
 include/net/veth.h | 14 +
 4 files changed, 473 insertions(+)

```
diff --git a/drivers/net/Kconfig b/drivers/net/Kconfig
index d4e39ff..7bdc6e3 100644
--- a/drivers/net/Kconfig
+++ b/drivers/net/Kconfig
@@ -124,6 +124,12 @@ 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 a2241e6..d7b1103 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..bf56e0e
--- /dev/null
+++ b/drivers/net/veth.c
@@ -0,0 +1,452 @@
+/*
+ *  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_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_strlcpy(ifname, data[VETH_INFO_PEER], IFNAMSIZ);
+ else
+  snprintf(ifname, IFNAMSIZ, DRV_NAME "%%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 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_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 + 1] = {
+ [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 = {
+ .kind  = DRV_NAME,
+ .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");
+MODULE_ALIAS_RTNL_LINK(DRV_NAME);
diff --git a/include/net/veth.h b/include/net/veth.h
new file mode 100644
index 0000000..b84a530
--- /dev/null
+++ b/include/net/veth.h
@@ -0,0 +1,14 @@
+#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
+#define VETH_INFO_MAX (__VETH_INFO_MAX - 1)
+};
+
+#endif
```

Subject: [PATCH] Module for ip utility to support veth device (v.2.1)
Posted by Pavel Emelianov on Wed, 11 Jul 2007 09:33:55 GMT
View Forum Message <> Reply to Message

The usage is
ip link add [name] type veth [peer <name>] [mac <mac>] [peer_mac <mac>]

This version doesn't include the fix for ip/iplink.c as Patrick
said that he had included it into his patches already.

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

```
 Makefile    |   6 +++-
 link_veth.c |  89 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 veth.h      |  14 +++++++++
 3 files changed, 108 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..7760684
--- /dev/null
+++ b/ip/link_veth.c
@@ -0,0 +1,89 @@
+/*
+ * 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 ETH_ALEN 6
```

```
+
+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)
+{
+ __u8 mac[ETH_ALEN];
+
+ for (; argc != 0; argv++, argc--) {
+  if (strcmp(*argv, "peer") == 0) {
+   argv++;
+   argc--;
+   if (argc == 0) {
+    usage();
+    return -1;
+   }
+
+   addattr_l(hdr, 1024, VETH_INFO_PEER,
+     *argv, strlen(*argv));
+
+   continue;
+  }
+
+  if (strcmp(*argv, "mac") == 0) {
+   argv++;
+   argc--;
+   if (argc == 0) {
+    usage();
+    return -1;
+   }
+
+   if (hexstring_a2n(*argv, mac, sizeof(mac)) == NULL)
+    return -1;
+
+   addattr_l(hdr, 1024, VETH_INFO_MAC,
+     mac, ETH_ALEN);
+   continue;
+  }
+
+  if (strcmp(*argv, "peer_mac") == 0) {
+   argv++;
+   argc--;
+   if (argc == 0) {
+    usage();
```

```
+    return -1;
+   }
+
+   if (hexstring_a2n(*argv, mac, sizeof(mac)) == NULL)
+    return -1;
+
+   addattr_l(hdr, 1024, VETH_INFO_PEER_MAC,
+     mac, ETH_ALEN);
+   continue;
+  }
+
+  usage();
+  return -1;
+ }
+
+ return 0;
+}
+
+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..b84a530
--- /dev/null
+++ b/ip/veth.h
@@ -0,0 +1,14 @@
+#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
+#define VETH_INFO_MAX (__VETH_INFO_MAX - 1)
+};
+
+#endif
```

Subject: Re: [PATCH] Virtual ethernet device (v2.1)
Posted by Evgeniy Polyakov on Wed, 11 Jul 2007 09:44:30 GMT
View Forum Message <> Reply to Message

Hi.

On Wed, Jul 11, 2007 at 01:31:15PM +0400, Pavel Emelianov (xemul@openvz.org) 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.

I have minor comment about flags used in veth. It looks like you only
set LLTX bit, but check for other flags in the code, so why not to
include another performance features?

```
> +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);
> +}
```

Don't you want to include NETIF_F_SG, NETIF_F_NO_CSUM, NETIF_F_FRAGLIST,
NETIF_F_HIGHDMA and probably TSO fancy flags also to reduce cpu usage more?

--
Evgeniy Polyakov

Subject: Re: [PATCH] Virtual ethernet device (v2.1)
Posted by Christian Borntraeger on Wed, 11 Jul 2007 10:54:24 GMT
View Forum Message <> Reply to Message

Am Mittwoch, 11. Juli 2007 schrieb Pavel Emelianov:
> drivers/net/veth.c | 452
> include/net/veth.h | 14 +

I know, I am late in the game, but wont the name collide somewhat with
drivers/net/ibmveth.h, drivers/net/iseries_veth.c, and drivers/net/ibmveth.c?

Christian

Subject: Re: [PATCH] Virtual ethernet device (v2.1)
Posted by Patrick McHardy on Wed, 11 Jul 2007 12:23:53 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> +static const struct nla_policy veth_policy[VETH_INFO_MAX + 1] = {
> + [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 },
> +};

Looks good, just one question. What happended to the IFLA_PARTNER
attribute idea? I have a patch to allow specifying the initial
MAC address for a device, IFLA_PARTNER would make the whole thing
symetrical.

Subject: Re: [PATCH] Virtual ethernet device (v2.1)
Posted by Pavel Emelianov on Wed, 11 Jul 2007 12:24:20 GMT
View Forum Message <> Reply to Message

Christian Borntraeger wrote:
> Am Mittwoch, 11. Juli 2007 schrieb Pavel Emelianov:
>> drivers/net/veth.c | 452
>> include/net/veth.h | 14 +
>
> I know, I am late in the game, but wont the name collide somewhat with
> drivers/net/ibmveth.h, drivers/net/iseries_veth.c, and drivers/net/ibmveth.c?

David asked the same thing after v.1. The answer is :
No it will not. These devices use eth%d strings to generate their names.

> Christian
>

Thanks,
Pavel

Subject: Re: [PATCH] Virtual ethernet device (v2.1)
Posted by Pavel Emelianov on Wed, 11 Jul 2007 12:44:53 GMT
View Forum Message <> Reply to Message

Patrick McHardy wrote:
> Pavel Emelianov wrote:
>> +static const struct nla_policy veth_policy[VETH_INFO_MAX + 1] = {
>> + [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 },
>> +};
>
>
> Looks good, just one question. What happended to the IFLA_PARTNER
> attribute idea? I have a patch to allow specifying the initial
> MAC address for a device, IFLA_PARTNER would make the whole thing
> symetrical.

Well, the notion of a partner is not applicable to any generic link
unlike the device name, physical (MAC) address or MTU value. So i
think that it's better to keep this as a specific driver information
not to confuse the generic code. I think it's worth making this as
some more generic code than it is now, but since this driver is the
only user of "partner" maybe it's better not to make it right now.
Later, when we do know what do we want "partner" to be.

Thanks,
Pavel

Subject: Re: [PATCH] Virtual ethernet device (v2.1)
Posted by Patrick McHardy on Wed, 11 Jul 2007 12:53:32 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Patrick McHardy wrote:
>
>>Pavel Emelianov wrote:
>>
>>>+static const struct nla_policy veth_policy[VETH_INFO_MAX + 1] = {
>>>+ [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 },
>>>+};
>>
>>
>>Looks good, just one question. What happended to the IFLA_PARTNER
>>attribute idea? I have a patch to allow specifying the initial
>>MAC address for a device, IFLA_PARTNER would make the whole thing
>>symetrical.
>
>
> Well, the notion of a partner is not applicable to any generic link
> unlike the device name, physical (MAC) address or MTU value. So i
> think that it's better to keep this as a specific driver information
> not to confuse the generic code. I think it's worth making this as

> some more generic code than it is now, but since this driver is the
> only user of "partner" maybe it's better not to make it right now.
> Later, when we do know what do we want "partner" to be.

Mhh doing it later means dealing with compatibility issues, which
is why I'm asking now. We currently support IFLA_NAME, IFLA_MTU,
IFLA_TXQLEN, IFLA_WEIGTH, IFLA_OPERSTATE and IFLA_LINKMODE, and
with my patch additionally IFLA_ADDRESS and IFLA_BROADCAST.
AFAICT they are all applicable for the partner link as well.

Subject: Re: [PATCH] Virtual ethernet device (v2.1)
Posted by Pavel Emelianov on Wed, 11 Jul 2007 13:00:35 GMT
View Forum Message <> Reply to Message

Patrick McHardy wrote:
> Pavel Emelianov wrote:
>> Patrick McHardy wrote:
>>
>>> Pavel Emelianov wrote:
>>>
>>>> +static const struct nla_policy veth_policy[VETH_INFO_MAX + 1] = {
>>>> + [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 },
>>>> +};
>>>
>>> Looks good, just one question. What happended to the IFLA_PARTNER
>>> attribute idea? I have a patch to allow specifying the initial
>>> MAC address for a device, IFLA_PARTNER would make the whole thing
>>> symetrical.
>>
>> Well, the notion of a partner is not applicable to any generic link
>> unlike the device name, physical (MAC) address or MTU value. So i
>> think that it's better to keep this as a specific driver information
>> not to confuse the generic code. I think it's worth making this as
>> some more generic code than it is now, but since this driver is the
>> only user of "partner" maybe it's better not to make it right now.
>> Later, when we do know what do we want "partner" to be.
>
>
> Mhh doing it later means dealing with compatibility issues, which
> is why I'm asking now. We currently support IFLA_NAME, IFLA_MTU,

Oh, I see.

> IFLA_TXQLEN, IFLA_WEIGTH, IFLA_OPERSTATE and IFLA_LINKMODE, and

> with my patch additionally IFLA_ADDRESS and IFLA_BROADCAST.
> AFAICT they are all applicable for the partner link as well.

Agree. Maybe it is better to make some generic routine to create the
device with the parameters specified in the netlink packet. Then the
generic code creates one end of a tunnel and calls ->new_link callback.
This callback extracts the PARTNER packet part and calls this generic
routine to create the second pair.

Thanks,
Pavel

Subject: Re: [PATCH] Virtual ethernet device (v2.1)
Posted by Patrick McHardy on Wed, 11 Jul 2007 13:12:33 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Patrick McHardy wrote:
>
>>Mhh doing it later means dealing with compatibility issues, which
>>is why I'm asking now. We currently support IFLA_NAME, IFLA_MTU,
>
>
> Oh, I see.
>
>
>>IFLA_TXQLEN, IFLA_WEIGTH, IFLA_OPERSTATE and IFLA_LINKMODE, and
>>with my patch additionally IFLA_ADDRESS and IFLA_BROADCAST.
>>AFAICT they are all applicable for the partner link as well.
>
>
> Agree. Maybe it is better to make some generic routine to create the
> device with the parameters specified in the netlink packet. Then the
> generic code creates one end of a tunnel and calls ->new_link callback.
> This callback extracts the PARTNER packet part and calls this generic
> routine to create the second pair.

Something like that. Moving the part between NLM_F_CREATE and the
ops->newlink call of rtnl_newlink to a new function should work.

For now you could even parse the IFLA_PARTNER attribute and nested
IFLA_NAME/IFLA_ADDRESS attributes yourself and ignore the rest,
this will at least leave us the option of handling it generically
later.

Subject: Re: [PATCH] Virtual ethernet device (v2.1)
Posted by Pavel Emelianov on Wed, 11 Jul 2007 13:32:22 GMT
View Forum Message <> Reply to Message

Patrick McHardy wrote:
> Pavel Emelianov wrote:
>> Patrick McHardy wrote:
>>
>>> Mhh doing it later means dealing with compatibility issues, which
>>> is why I'm asking now. We currently support IFLA_NAME, IFLA_MTU,
>>
>> Oh, I see.
>>
>>
>>> IFLA_TXQLEN, IFLA_WEIGTH, IFLA_OPERSTATE and IFLA_LINKMODE, and
>>> with my patch additionally IFLA_ADDRESS and IFLA_BROADCAST.
>>> AFAICT they are all applicable for the partner link as well.
>>
>> Agree. Maybe it is better to make some generic routine to create the
>> device with the parameters specified in the netlink packet. Then the
>> generic code creates one end of a tunnel and calls ->new_link callback.
>> This callback extracts the PARTNER packet part and calls this generic
>> routine to create the second pair.
>
>
> Something like that. Moving the part between NLM_F_CREATE and the
> ops->newlink call of rtnl_newlink to a new function should work.
>
> For now you could even parse the IFLA_PARTNER attribute and nested
> IFLA_NAME/IFLA_ADDRESS attributes yourself and ignore the rest,
> this will at least leave us the option of handling it generically
> later.

OK. I'll try to make the generic call. Could you please send me the
patches with IFLA_ADDRESS support for booth kernel and ip utility.

Thanks,
Pavel

Subject: Re: [PATCH] Virtual ethernet device (v2.1)
Posted by Patrick McHardy on Wed, 11 Jul 2007 17:32:52 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> OK. I'll try to make the generic call. Could you please send me the
> patches with IFLA_ADDRESS support for booth kernel and ip utility.

I'll send the kernel patches seperately in a few minutes. iproute
doesn't need any further patches.

Subject: Re: [PATCH] Virtual ethernet device (v2.1)
Posted by davem on Wed, 11 Jul 2007 20:44:52 GMT
View Forum Message <> Reply to Message

From: Christian Borntraeger <borntraeger@de.ibm.com>
Date: Wed, 11 Jul 2007 12:54:24 +0200

> Am Mittwoch, 11. Juli 2007 schrieb Pavel Emelianov:
> > drivers/net/veth.c | 452
> > include/net/veth.h | 14 +
>
> I know, I am late in the game, but wont the name collide somewhat with
> drivers/net/ibmveth.h, drivers/net/iseries_veth.c, and drivers/net/ibmveth.c?

We discussed that and decided that it's not an issue.
