Subject: [PATCH] Virtual ethernet tunnel (v.2)
Posted by Pavel Emelianov on Thu, 07 Jun 2007 11:13:02 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 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.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>

Since ethtool interface was taken from Eric's patch, I think
that he would like to see his Signed-off line as well (however
he didn't answer yesterday).

---

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..e7ad43d
--- /dev/null
+++ b/drivers/net/veth.c
@@ -0,0 +1,442 @@
+/*
+ *  drivers/net/veth.c
+ *
+ *  Copyright (C) 2007 OpenVZ http://openvz.org, SWsoft Inc
+ *
+ *  Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+#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_device_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_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_device_stats *stats;
+ struct veth_priv *priv, *rcv_priv;
+ int length, cpu;
+
+ skb_orphan(skb);
+
+ priv = netdev_priv(dev);
+ cpu = smp_processor_id();
+ stats = per_cpu_ptr(priv->stats, cpu);
+ rcv = priv->peer;
+
+ if (!(rcv->flags & IFF_UP))
+  goto outf;
+
+ rcv_priv = netdev_priv(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);
+ skb->mark = 0;
+
+ 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 *stats;
+ struct veth_device_stats *vstats;
+ int cpu;
+
+ priv = netdev_priv(dev);
+ stats = &dev->stats;
+ stats->rx_packets = 0;
+ stats->tx_packets = 0;
+ stats->rx_bytes = 0;
+ stats->tx_bytes = 0;
+ stats->tx_dropped = 0;
+
+ for_each_possible_cpu(cpu) {
+  vstats = per_cpu_ptr(priv->stats, cpu);
+
+  stats->rx_packets += vstats->rx_packets;
+  stats->tx_packets += vstats->tx_packets;
+  stats->rx_bytes += vstats->rx_bytes;
```

```
+  stats->tx_bytes += vstats->tx_bytes;
+  stats->tx_dropped += vstats->tx_dropped;
+ }
+
+ return 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_init(struct net_device *dev)
+{
+ struct veth_priv *priv;
+
+ priv = netdev_priv(dev);
+ priv->stats = alloc_percpu(struct veth_device_stats);
+ return priv->stats == NULL ? -ENOMEM : 0;
+}
+
+static void veth_destructor(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->init = veth_init;
+ dev->destructor = veth_destructor;
+ dev->ethtool_ops = &veth_ethtool_ops;
+ dev->features |= NETIF_F_LLTX;
+ 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
+  *
```

```
+  * TODO: this should be done in another namespace
+  */
+
+ 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);
+ if (!list_empty(&priv->list))
+  list_del(&priv->list);
+
+ peer = priv->peer;
+ 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 = {
+ .name  = 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_module(void)
+{
+ return rtnl_link_register(&veth_link_ops);
+}
+
+static __exit void veth_exit_module(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);
+module_exit(veth_exit_module);
+
+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..d52e0c5
--- /dev/null
+++ b/include/net/veth.h
@@ -0,0 +1,15 @@
+#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)
Posted by Pavel Emelianov on Thu, 07 Jun 2007 11:16:34 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>

```
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..f2e4079
--- /dev/null
+++ b/ip/link_veth.c
@@ -0,0 +1,86 @@
+/*
+ * ip/link_veth.c
+ *
+ * Virtual ETHernet tunnel supprt.
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ */
+
+#include <stdio.h>
+#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 ... "
+   "[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..d52e0c5
--- /dev/null
```

```
+++ b/ip/veth.h
@@ -0,0 +1,15 @@
+#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
```

_____

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

---

Subject: Re: [PATCH] Virtual ethernet tunnel (v.2)
Posted by Ben Greear on Fri, 08 Jun 2007 17:00:20 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Ben Greear wrote:
>
> [snip]
>
>
>>>> I would also like some way to identify veth from other device types,
>>>> preferably
>>>> something like a value in sysfs. However, that should not hold up
>>>>
>>>>
>>> We can do this with ethtool. It can get and print the driver name of
>>> the device.
>>>
>>>
>> I think I'd like something in sysfs that we could query for any
>> interface. Possible return
>> strings could be:
>> VLAN
>> VETH
>> ETH

>> PPP
>> BRIDGE
>> AP /* wifi access point interface */
>> STA /* wifi station */
>> ....
>>
>> I will cook up a patch for consideration after veth goes in.
>>
>>
>
> Ben, could you please tell what sysfs features do you
> plan to implement?
>
I think this is the only thing that has a chance of getting into the kernel. Basically, I have a user-space app and I want to be able to definitively know the type for
all interfaces. Currently, I have a hodge-podge of logic to query various ioctls and /proc
files and finally, guess by name if nothing else works. There must be a better way :P

I have another sysfs patch that allows setting a default skb->mark for an interface so that you can set the skb->mark
before it hits the connection tracking logic, but I'm been told this one has very little chance
of getting into the kernel. The skb->mark patch is only useful (as far as I can tell) if you
also include a patch Patrick McHardy did for me that allowed the conn-tracking logic to
use skb->mark as part of it's tuple. This allows me to do NAT between virtual routers
(routing tables) on the same machine using veth-equivalent drivers to connect the
routers. He thinks this will probably not ever get into the kernel either.

I have another sysctl related send-to-self patch that also has little chance of getting into the kernel, but
it might be quite useful with veth (it's useful to me..but my needs aren't exactly mainstream :))
I'll post this separately for consideration....

Thanks,
Ben

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

Subject: Re: [PATCH] Virtual ethernet tunnel (v.2)
Posted by Carl-Daniel Hailfinge on Fri, 08 Jun 2007 19:49:17 GMT
View Forum Message <> Reply to Message

On 08.06.2007 19:00, Ben Greear wrote:
> I have another sysfs patch that allows setting a default skb->mark for
> an interface so that you can set the skb->mark
> before it hits the connection tracking logic, but I'm been told this one
> has very little chance
> of getting into the kernel. The skb->mark patch is only useful (as far
> as I can tell) if you
> also include a patch Patrick McHardy did for me that allowed the
> conn-tracking logic to
> use skb->mark as part of it's tuple. This allows me to do NAT between
> virtual routers
> (routing tables) on the same machine using veth-equivalent drivers to
> connect the
> routers. He thinks this will probably not ever get into the kernel either.

Are these patches available somewhere? I'm currently doing NAT between
virtual routers by some advanced iproute2/iptables trickery, but I have
no way to handle the occasional tuple conflict.

Regards,
Carl-Daniel

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

Subject: Re: [PATCH] Virtual ethernet tunnel (v.2)
Posted by Ben Greear on Fri, 08 Jun 2007 23:46:32 GMT
View Forum Message <> Reply to Message

Carl-Daniel Hailfinger wrote:
> On 08.06.2007 19:00, Ben Greear wrote:
>> I have another sysfs patch that allows setting a default skb->mark for
>> an interface so that you can set the skb->mark

>> before it hits the connection tracking logic, but I'm been told this one
>> has very little chance
>> of getting into the kernel. The skb->mark patch is only useful (as far
>> as I can tell) if you
>> also include a patch Patrick McHardy did for me that allowed the
>> conn-tracking logic to
>> use skb->mark as part of it's tuple. This allows me to do NAT between
>> virtual routers
>> (routing tables) on the same machine using veth-equivalent drivers to
>> connect the
>> routers. He thinks this will probably not ever get into the kernel either.
>
> Are these patches available somewhere? I'm currently doing NAT between
> virtual routers by some advanced iproute2/iptables trickery, but I have
> no way to handle the occasional tuple conflict.

A consolidated patch against 2.6.20.12 is here. It has a lot more than
just the patches mentioned above, but it shouldn't hurt anything to have
the whole patch applied:

http://www.candelatech.com/oss/candela_2.6.20.patch

The original patch for using skb->mark as a tuple was
written by Patrick McHardy, and is here:

http://www.candelatech.com/oss/skb_mark_conntrack.patch

His patch merged with my patch to sysfs to set skb->mark on ingress is here:
http://www.candelatech.com/oss/conntrack_mark_with_ssyctl.patch

Thanks,
Ben

>
> Regards,
> Carl-Daniel

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