Subject: Re: [PATCH] net: Add etun driver
Posted by ebiederm on Sat, 07 Apr 2007 02:51:58 GMT
View Forum Message <> Reply to Message

Stephen Hemminger <shemminger@linux-foundation.org> writes:

> Why not implement a true virtual network rather than simple
> tunnel pairs?

Mostly I don't even need etun. It is a cheap way to make up for
the lack of sufficient physical network adapters on the machine.
If you plug in enough network cards into the machine my network
namespace code works just fine without it.

etun is just a way to use the current bridging and routing code,
without a lot of complexity.

I think what I really want is something like your current ethernet
vlan code that worked at the mac address level. While handling
broadcasts and configuration similar to our current ethernet
bridging code.

However truly advanced low overhead things that require touching the
driver or are specific to a particular kind of networking are never
universally available so I need something that is.

With a little luck and a couple of tweaks as yet undetermined tweaks
to the etun driver and you won't be able to measure any overhead so it
will be enough. I doubt it but I can always hope.

Not doing anything extra and using a tool building approach puts
emphasis on improving our ethernet bridgine and our ip routing code.

```
> Details:
>
>
>> +static struct {
>> + const char string[ETH_GSTRING_LEN];
>> +} ethtool_stats_keys[ETUN_NUM_STATS] = {
>> + { "partner_ifindex" },
>> +};
>
>
> You should use sysfs attributes for this rather than
> ethtool.
```

I think it is six of one half a dozen of the other. It is easy enough to change if need be.

```
>
>> +static int etun_ioctl(struct net_device *dev, struct ifreq *rq, int cmd)
>> +{
>> + return -EOPNOTSUPP;
>> +}
>
> If not supported, then no stub needed just leave null.
```

It's a place holder in case I need to support some ioctl. Having it
hear makes it clear that I don't support anything right now by design.

```
>> +/* Only allow letters and numbers in an etun device name */
>> +static int is_valid_name(const char *name)
>> +{
>> + const char *ptr;
>> + for (ptr = name; *ptr; ptr++) {
>> +  if (!isalnum(*ptr))
>> +   return 0;
>> + }
>> + return 1;
>> +}
>> +
>> +static struct net_device *etun_alloc(const char *name)
>> +{
>> + struct net_device *dev;
>> + struct etun_info *info;
>> + int err;
>> +
>> + if (!name || !is_valid_name(name))
>> +  return ERR_PTR(-EINVAL);
>> +
>> + dev = alloc_netdev(sizeof(struct etun_info), name, ether_setup);
>> + if (!dev)
>> +  return ERR_PTR(-ENOMEM);
>
> Use alloc_etherdev() instead.
```

I could and it might be a little bit less code, but I would loose the
ability for users to request the name of their network device when
they allocate it.

Since I can't return any kind of a handle it seems very useful to
let the user pick the name a network device will initially be known
as.

```
>> +module_param_call(newif, etun_newif, etun_noget, NULL, S_IWUSR);
>> +module_param_call(delif, etun_delif, etun_noget, NULL, S_IWUSR);
>
>
> Doing create/delete via module parameter is wierd.
> Why not either use an ioctl, or sysfs.
```

I agree. However I could not find a create/delete idiom that was
at all common when I looked through the kernel virtual interfaces.
Which makes every way to create/delete an interface weird to
some degree.

In this case I am using sysfs.

The only sysfs attributes that were always available that I could find
were module parameters. A little odd because we can specify them on
the kernel command line, or when loading the module in addition to
being available at run time.

It gives me a general interface that is usable so long as the module
is loaded, and does not depend on the availability of any specific
network device. I will happily use any other interface that gives
me the same level of functionality for the roughly the same level
of effort.

```
>> +static int __init etun_init(void)
>> +{
>> + printk(KERN_INFO "etun: %s, %s\n", DRV_DESCRIPTION, DRV_VERSION);
>> + printk(KERN_INFO "etun: %s\n", DRV_COPYRIGHT);
>> +
>> + return 0;
>
> Why bother it is just advertising noise...
```

True, I really haven't given it a lot of though.

I doesn't really hurt to advertise and as well as social consequences
it technically allows detection of a kernel capability by reading boot
messages which can be very useful depending on the situation.

Eric