Forwarding CAN Bus traffic to a Docker container using vxcan on Raspberry Pi

vxcan is a Linux kernel driver/module that can be used to set up a virtual CAN tunnel across network namespaces. For example it allows you to generate virtual CAN frames on your host and send them to a container; or forward real CAN traffic between a USB-CAN adapter and a container, without exposing the entire host network to the container.

The following instructions are for a Raspberry Pi 4 model B running Raspberry Pi OS, on kernel `5.4.72-v7l+` (use `uname -r` to verify). It should work with fairly minor modifications for other OSes; some paths and package names may be different.

First, install some dependencies and download the vxcan module source code:

```
sudo apt-get update
sudo apt-get install raspberrypi-kernel-headers can-utils
mkdir vxcan
cd vxcan
wget "https://raw.githubusercontent.com/torvalds/linux/master/drivers/net/can/vxcan.c"
```

We'll also need a Makefile (make sure it's using tabs and not spaces!):

```
obj-m += vxcan.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

At this point you should have a directory with two files, `vxcan.c` and `Makefile`, so let's build the kernel module and load it:

```
make
sudo chown root:root vxcan.ko
sudo chmod 0644 vxcan.ko
sudo mv vxcan.ko /lib/modules/5.4.72-v7l+/kernel/net/can/
sudo depmod -A
sudo modprobe vxcan
sudo modprobe can-gw
```

Let's also add a file to `/etc/modules-load.d` so that the modules will load on startup. Create `/etc/modules-load.d/can.conf` and add the following:

```
vxcan
can-gw
```

Next, in a separate terminal let's start a container and install canutils within the container:

```
docker run --rm -it --name cantest ubuntu:20.04
apt-get update && apt-get install -y can-utils
```

Then in our original terminal let's set up the vxcan network and move one end of it into the container's network namespace:

```
DOCKERPID=$(docker inspect -f '{{ .State.Pid }}' cantest)
sudo ip link add vxcan0 type vxcan peer name vxcan1
sudo ip link set vxcan1 netns $DOCKERPID
sudo ip link set vxcan0 up
sudo nsenter -t $DOCKERPID -n ip link set vxcan1 up
```

We moved `vxcan1` into the container's namespace, so now back in the container we can run: `candump vxcan1`

Finally, in our host we can send data to `vxcan0` and have it show up in the container: `cansend vxcan0 123#1122`

Bonus point: if you have a real CAN adapter, you can also forward traffic from that adapter into the container using cangw:

```
sudo cangw -A -s can0 -d vxcan0 -e
sudo cangw -A -s vxcan0 -d can0 -e
```

Questions? Feedback? Want to learn more about how Lager can help debug your CAN device? Contact us at blog@lagerdata.com