SocketCAN mit Docker unter Linux

Daniel Krüger, SYS TEC electronic AG, 13.03.2021

SocketCAN with Docker under Linux

Daniel Krüger, SYS TEC electronic AG, 13.03.2021

Stackoverflow answer

- https://stackoverflow.com/questions/42769011/access-non-ip-interface-in-docker-container

stackoverflow About Products For Teams Search...

Home

PUBLIC

Stack Overflow

Tags

Users

Access non-IP interface in docker container

Asked 3 years, 11 months ago Active 3 years, 11 months ago Viewed 805 times

1

is it possible to access a non-IP network interface inside a docker-container? All docker (bridge etc.) networks seem to run with IP-addressing, but I want to access a CAN (`PF_CAN` = SocketCAN) interface `can0`.

[...]

1 Answer

Active | Oldest | Votes

1

Thank you @Salem: I didn't know the `--net=host` parameter - It works! So in summary:

Run

```
$ docker run --net=host ...`
```

or use Docker-Compose:

```
$ cat docker-compose.yml
version: '3'
services:
  foo:
    network_mode: host
```

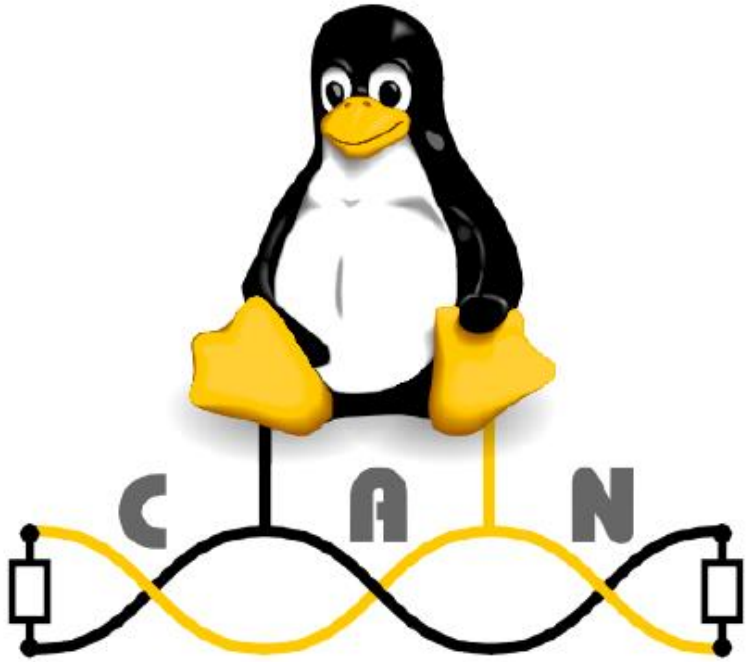
Docker introduction

- Popular (Linux) container runtime environment
- Packages application into portable container (OCI images)
- More lightweight than full machine virtualization (e.g. VirtualBox or KVM)
- Easy to use

SocketCAN

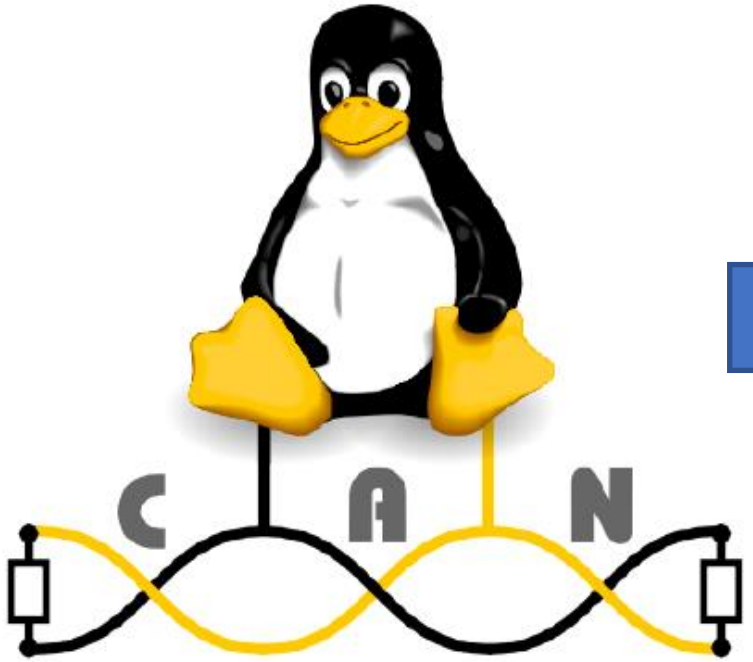
- SocketCAN is the framework for CAN under Linux
- CAN drivers are network drivers (delivered with Linux kernel)
- Applications receive and transmit CAN messages via BSD Socket API

- CAN message:
  - Identifier (11-bit often)
  - Payload (0..8 byte)
  - Bit rate: up to 1 MBit/s
  - Multi master network (layer 2)



Use cases for SocketCAN in Docker

- Testing of CAN applications in Docker container
- Have multiple CAN applications with different runtime environments



SocketCAN in Docker (simple solution 1)

- What does not work?
  - ✖ Forward USB interface to Container
    and run driver in Container
- Why?
  - ➔ Container is user-space only,
    SocketCAN drivers aren't based on libusb
    (because they use Linux kernel networking sub-system)

SocketCAN in Docker (simple solution 2)

- Host networking mode and/or Priviledged mode
  - Application in container accesses CAN interface on host
- Why isn‘t this an ideal solution?
  - ➔ No/less container isolation
  - ➔ Bad solution

2 Have you tried to run your container with `--net=host`? – Salem Mar 13 '17 at 20:55

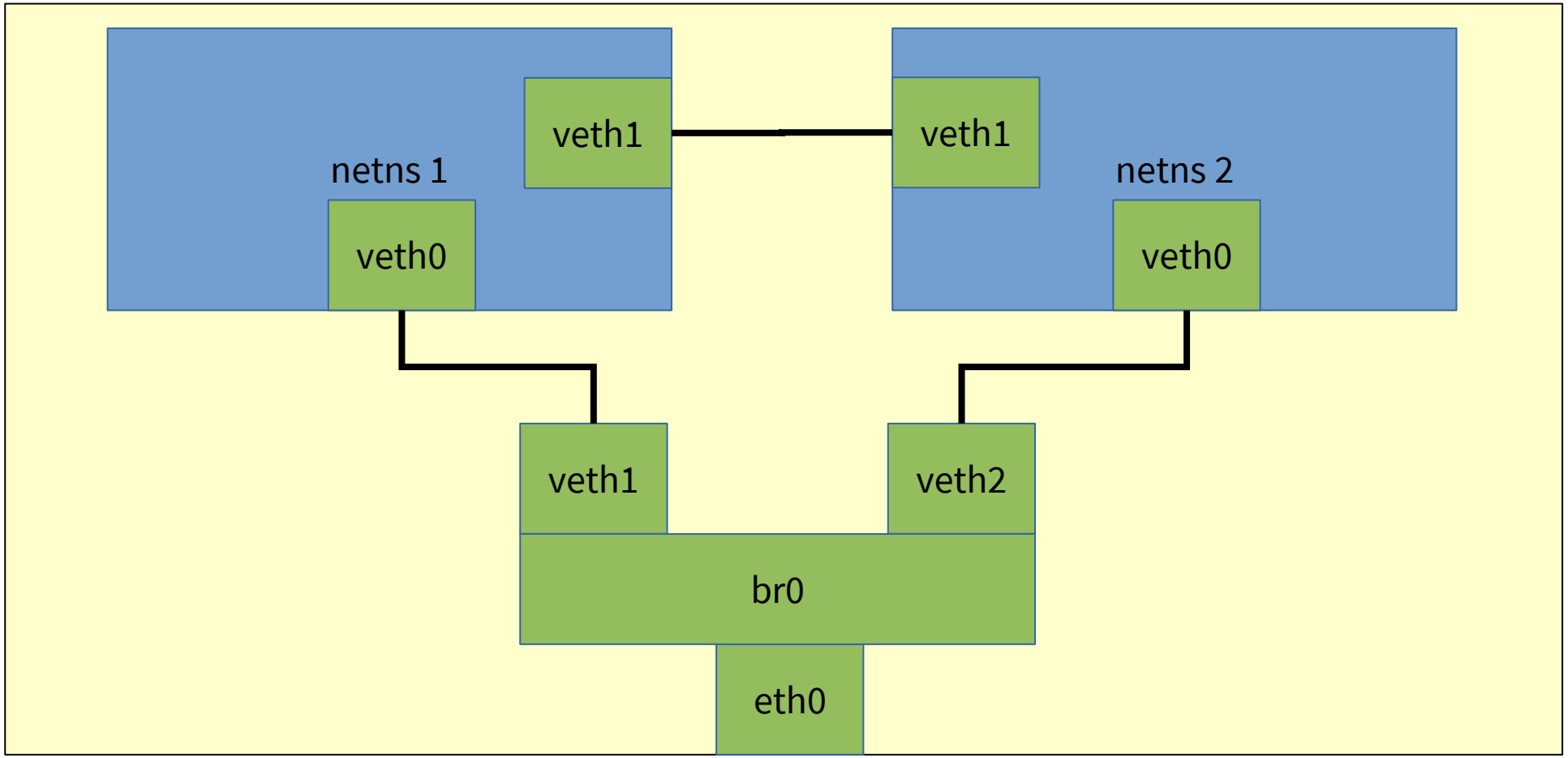
1 @Salem thanks, this works for me too. However, is it possible to share only that specific CAN interface? (if you set it `network mode` to `host`, then the container can't be connected to any other `networks`).
– Gautam Jun 27 '19 at 17:16

Add a comment

Source: https://stackoverflow.com/questions/42769011/access-non-ip-interface-in-docker-container

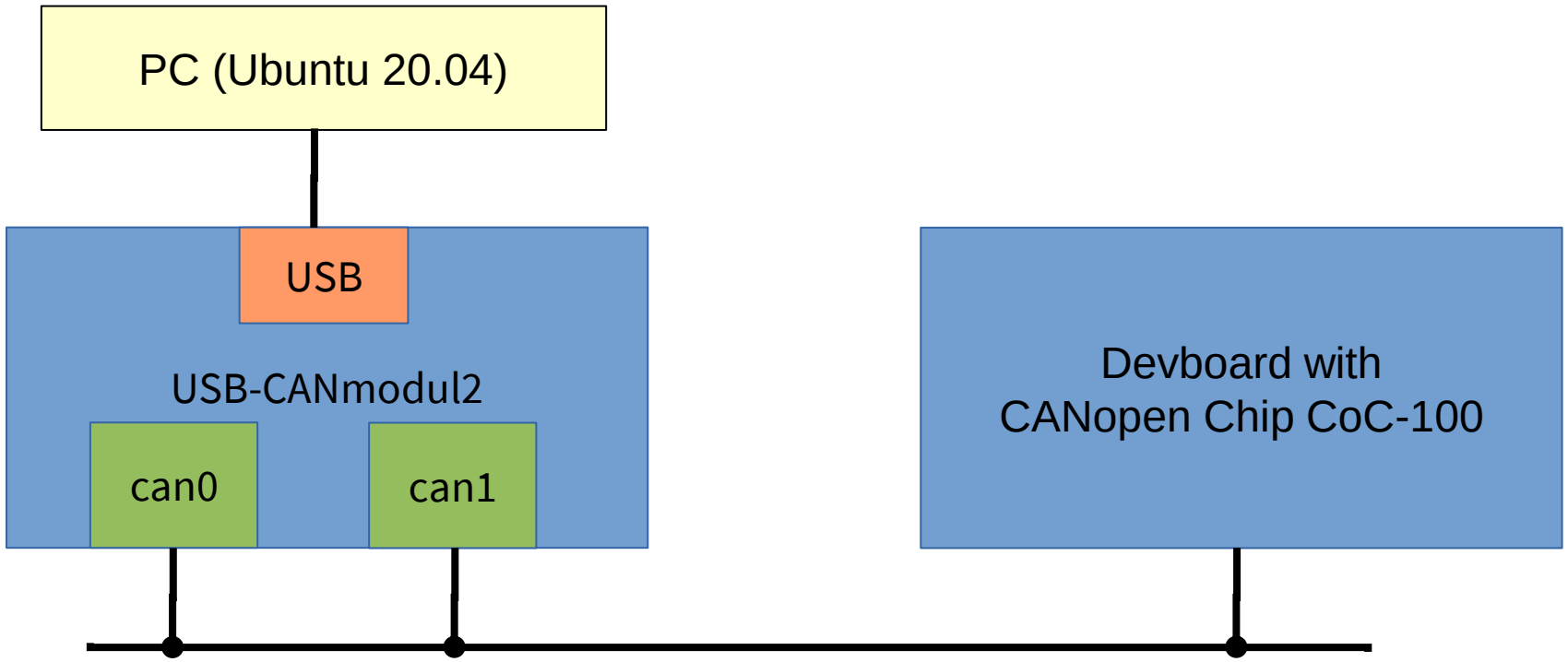
Ethernet (TCP/IP) in Docker

- veth (Virtual Ethernet): local Ethernet tunnel between network namespaces



Source: https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-for-virtual-networking/

Demo Setup


PC (Ubuntu 20.04)
USB
USB-CANmodul2
can0
can1
Devboard with
CANopen Chip CoC-100

Solution 1: Move can0 into namespace of Docker

- 1. Terminal:

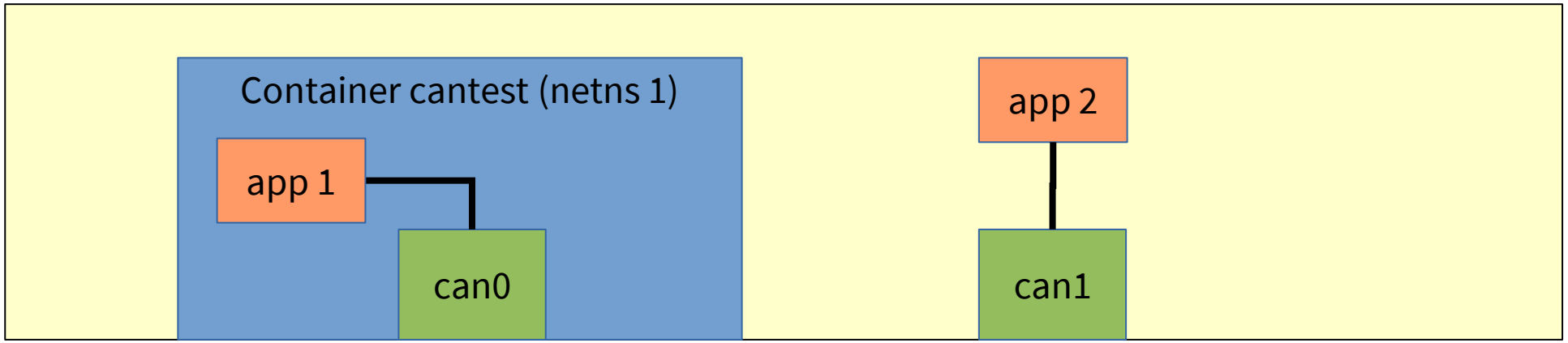
```
docker run --rm -it --name cantest ubuntu:20.04
apt-get update && apt-get install -y can-utils
```

- 2. Terminal:

```
DOCKERPID=$(docker inspect -f '{{ .State.Pid }}' cantest)
sudo ip link set can0 netns $DOCKERPID
sudo nsenter -t $DOCKERPID -n ip link set can0 type can bitrate 125000
sudo nsenter -t $DOCKERPID -n ip link set can0 up
```

- 1. Terminal:

```
candump can0
```



Solution 1: Move can0 back to root namespace

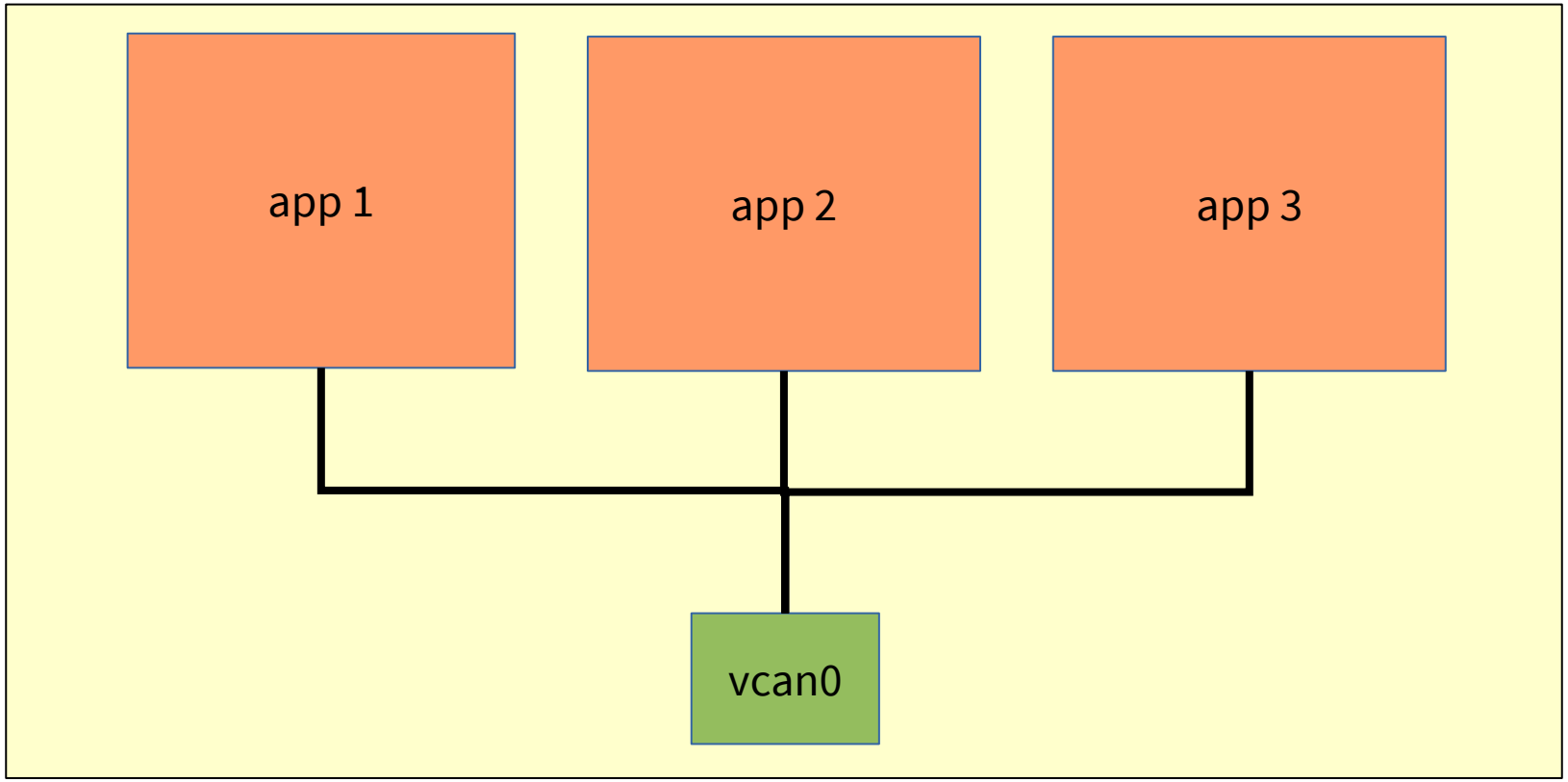
- 2. Terminal:

```
sudo nsenter -t $DOCKERPID -n ip link set can0 netns 1
```

Important: *Do it before Docker container is closed. Otherwise can0 gets lost in space.*

SocketCAN virtual interfaces (1)

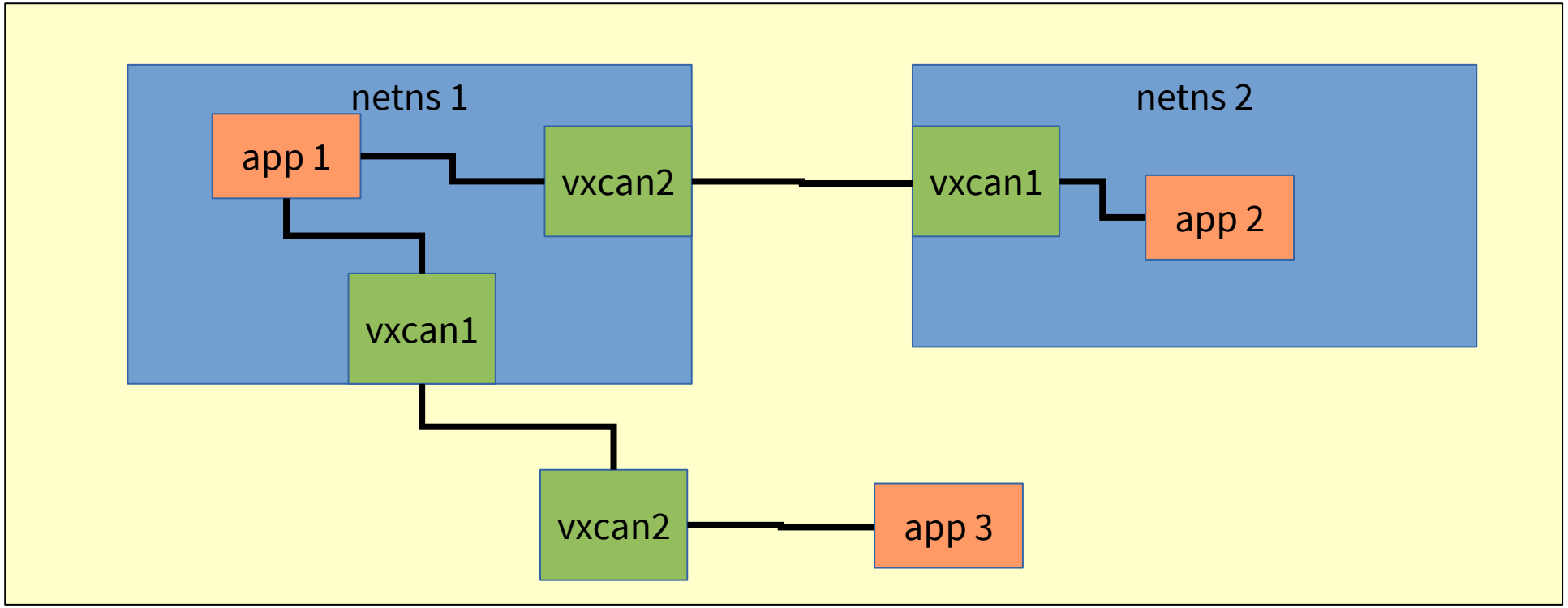
- vcan (virtual CAN): virtual local CAN bus (within single network namespace)
  `ip link add dev vcan0 type vcan`



SocketCAN virtual interfaces (2)

- vxcan (virtual CAN tunnel): pair of devices to create tunnel between network namespaces (no local echo, single app only)

```
ip netns add net1
ip netns add net2
ip link add vxcan1 netns net1 type vxcan peer name vxcan2 netns net2
```



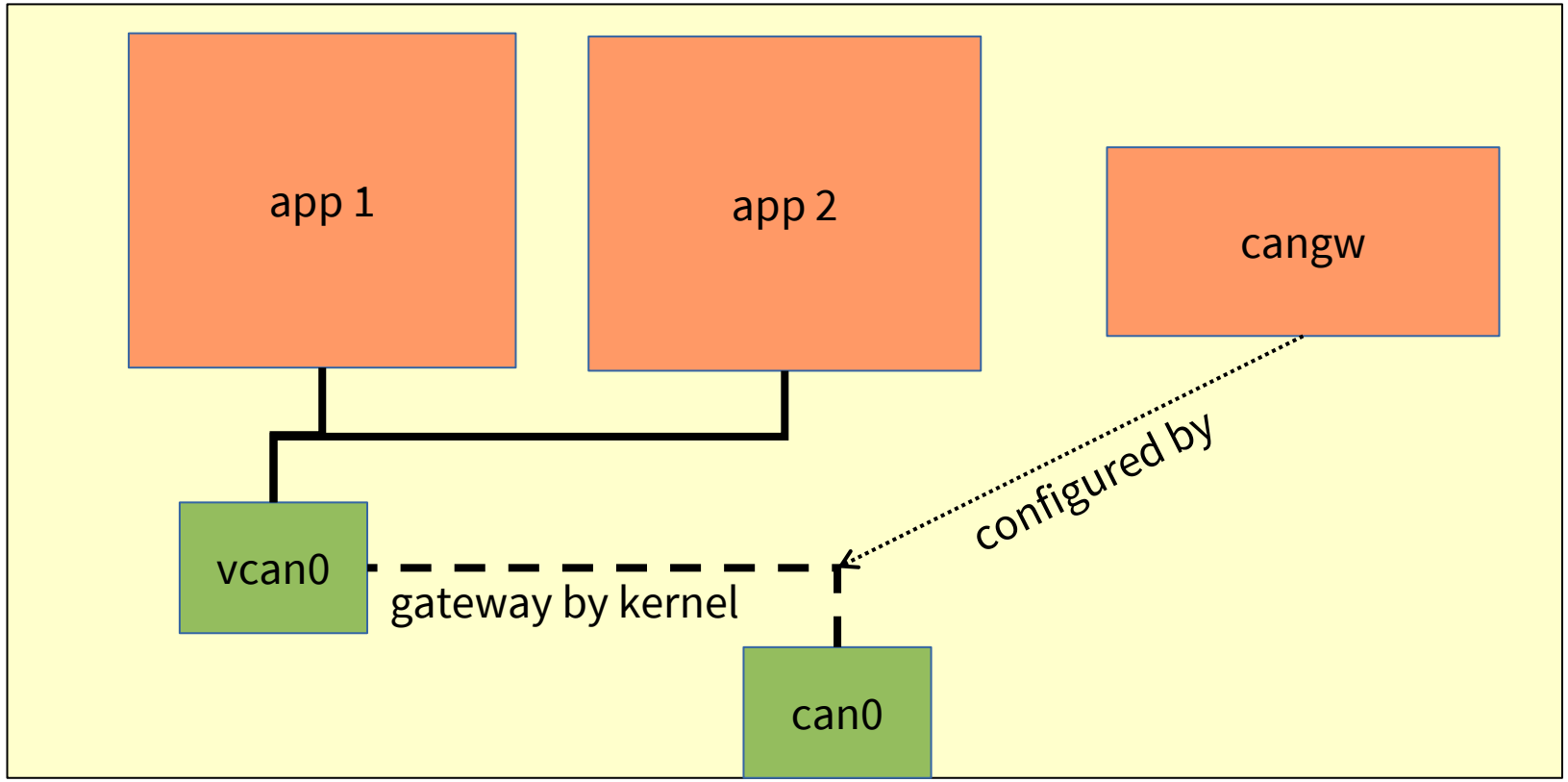
Solution 2: vxcan with Docker

- 1. Terminal:
```
docker run --rm -it --name cantest ubuntu:20.04
apt-get update && apt-get install -y can-utils
```
- 2. Terminal:
```
DOCKERPID=$(docker inspect -f '{{ .State.Pid }}' cantest)
sudo ip link add vxcan0 type vxcan peer name vxcan1 netns $DOCKERPID
sudo ip link set vxcan0 up
sudo nsenter -t $DOCKERPID -n ip link set vxcan1 up
```
- 1. Terminal:
```
candump vxcan1
```
- 2. Terminal:
```
cansend vxcan0 123#1122
```

SocketCAN gateway

- CAN_GW functionality controlled via cangw program

```
ip link add dev vcan0 type vcan
sudo modprobe can-gw
cangw -A -s vcan0 -d can0 -e
cangw -A -s can0 -d vcan0 -e
```



Solution 3: vxcan and cangw with Docker

- 1. Terminal:

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

- 2. Terminal:

```
DOCKERPID=$(docker inspect -f '{{ .State.Pid }}' cantest)
sudo ip link add vxcan0 type vxcan peer name vxcan1 netns $DOCKERPID
sudo modprobe can-gw
cangw -A -s can0 -d vxcan0 -e
cangw -A -s vxcan0 -d can0 –e
sudo ip link set vxcan0 up
sudo ip link set can0 type can bitrate 125000
sudo ip link set can0 up
sudo nsenter -t $DOCKERPID -n ip link set vxcan1 up
```

- 1. Terminal:

```
candump vxcan1
```

- 2. Terminal:

```
cansend can0 123#1122
```

Comparison vcan vs. vxcan

vcan (virtual CAN)	vxcan (virtual CAN tunnel)
Similar to real CAN device	pair of devices to create tunnel between network namespaces
Multiple applications can connect to same device	No local echo, single application only
No bitrate setting required	
Exists in single network namespace	Each device of pair can belong to different network namespaces

Advantages of SocketCAN with Docker

- All advantages of Docker containers:
  - Separation of systems,
    Limitation of pollution (rm -rf /) at container boundaries
  - Each container has a separate IP address exactly the same if the application in each container would run on a different host/embedded system
  - Reproducable application environments, created by Dockerfile recipe
  - Lighter than virtual machines
  - Works on embedded devices too (like Raspberry Pi)
- Security by separation, no direct access of physical CAN device can0 and other HW interfaces
- Virtual machines would require one CAN HW interface per VM,
  multiple Docker containers can be connected to a single CAN HW interface

Disadvantages of SocketCAN with Docker

- Requires some manual interaction
- Little run-time overhead, because of CAN message bridging

Alternatives to SocketCAN with Docker

- Tunnel SocketCAN over some IP protocol

References

- CANopen demo project: https://github.com/systec-dk/canopen-demo
- Forwarding CAN Bus traffic to a Docker container using vxcan on Raspberry Pi: https://www.lagerdata.com/blog/forwarding-can-bus-traffic-to-a-docker-container-using-vxcan-on-raspberry-pi
- Oliver Hartkopp, Design & separation of CAN applications: https://wiki.automotivelinux.org/_media/agl-distro/agl2018-socketcan.pdf
- Christian Gagneraud, can4docker: https://gitlab.com/chgans/can4docker
- Daniel Krüger, SocketCAN – CAN-Treiberschnittstelle unter Linux: https://chemnitzer.linux-tage.de/2012/vortraege/1044
- Christian Sandberg, CANopen for Python: https://github.com/christiansandberg/canopen
- Martin Willi, Kernelpatch Move device back to init netns on owning netns delete: https://lore.kernel.org/linux-can/20210302122423.872326-1-martin@strongswan.org/

Conclusion

- SocketCAN with Docker works
- Usability could be increased

Thanks for your attention!