Traffic engineering

2023/2024

Introduction to network virtualization and softwarization

Part #A

Virtual networking using Linux namespaces

Eliana Neuza da Silva

Fernando Mira da Silva

May 2025

1. Introduction

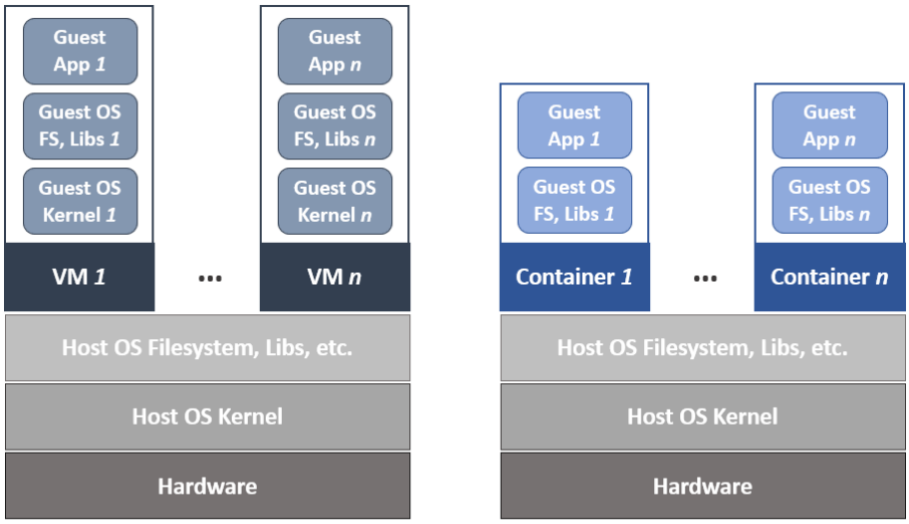
This is the first part of the third TE project, which will deal with low level network virtualization tools in Linux. The final report must include Parts #A and #B, and the final report must be delivered until June 15th, Sunday. This first part must be completed in two sessions (2 June, Monday, and 5 June, Thursday).

Before starting the lab work it is important to understand certain concepts that will be briefly addressed below.

1.1 Full virtualization vs Lightweight virtualization

Full virtualization (shown in Figure 1 A) usually means virtual machines (VM) running on top of a hypervisor which may already run on top of a host operating system (OS). Each VM needs to run its own guest OS with its own kernel managing their virtual resources. VMs provide the best level of isolation for virtual environments but at the cost of having to run a full OS, with its own kernel, for each VM. This may result in a higher overhead for processing but also a less efficient usage of resources.

Lightweight virtualization (shown in Figure 1 B) is a type of virtualization that does not require a hypervisor and guest operating systems. Instead, virtualization is created at the host OS level, by having the kernel itself isolate processes and resources. This means that processes can run with access to separate resources, while still running on the same kernel. An increasingly popular usage of this type of virtualization is containerization. Containers package the environment (libraries, configuration files, etc.) needed to run applications isolated at the file system level. The kernel provides each container its own resources (interfaces, ports, sockets, …) so it can use them without conflicting with other processes.



A) Virtual Machines *B) Containers*

Figure 1 – VMs (A) and containers (B) running on a host OS

1.2 Namespaces

The Linux kernel way of providing isolation is through what is called namespaces. A namespace creates an abstraction between system resources making the processes inside each of their own namespace appear to have their own isolated instance of the global resource[1]. By allowing isolation of global system resources between independent processes they become an important basis to running containers (lightweight virtualization).

There are seven types of namespaces[1]: Network (NET), Control group (Cgroup), Interprocess Communication (IPC), Mount (mnt), User ID (USER), Process ID (PID), and UNIX Time Sharing (UTS). For this lab we will focus on network namespaces as they provide a way of isolating network resources and processes using them, essentially enabling the creation of complete network topologies.

2. Goals

The goal of this lab project is to test fundamental tools for the development of programmable networks. In this scope, this project will be devoted to learn how to emulate network topologies using the Linux kernel network virtualization functionalities, such as namespaces, virtual interfaces and bridges.

4. Setup and Testing

Use any Linux distribution (it can be a virtual machine) with `root` access or a user with `sudo` privileges. The following instructions are the basics on namespaces networking in Linux that you will need to create the topologies of the three lab exercises.

Create a network namespace:

```
ip netns add <NSNAME>
```

Delete a specific namespace:

```
ip netns delete <NSNAME>
```

Show the list of current namespaces:

```
ip netns list
```

Show the list of network devices/interfaces:

```
ip link list
```

Assign interface to a network namespace (default namespace is 1):

```
ip link set dev <DEVNAME> netns <NSNAME>
```

Run a command inside a network namespace:

```
ip netns exec <NSNAME> <COMMAND>
```

Show list of interfaces in a specific namespace:

```
ip netns exec <NSNAME> ip link list
```

Assign virtual ethernet (`veth`) interfaces to namespaces[2]:

[1] https://man7.org/linux/man-pages/man7/namespaces.7.html

[2] veth interfaces come in pairs, whatever comes in on one veth interface will come out of the other peer veth interface.

To start, create the veth interfaces (both `veth` and physical interfaces belong to the "default" / "global" namespace):

```
ip link add veth0 type veth peer name veth1
```

Second, connect the interfaces to specific namespace (connected the "global" namespace to the NSNAME namespace):

```
ip link set veth1 netns <NSNAME>
```

Configure the veth interfaces for network connectivity:

First, assign an IP address to the interface:

```
ip netns exec <NSNAME> ip addr add 10.1.1.1/24 dev veth1
```

Second, bring the interface up:

```
ip netns exec <NSNAME> ip link set dev veth1 up
```

Show list of interfaces' addresses in the "global" namespace:

```
ip addr list
```

Show the interfaces and addresses in a specific namespace:

```
ip netns exec <NSNAME> ip addr list
```

Show the routing table entries:

```
ip route list
```

4.1 Two-node network

The first exercise will be to build a network with two nodes, one will be our host and the other a new namespace, as shown in Figure 2. The final objective is to be able to, from the default namespace, check connectivity to the `examplens` namespace.

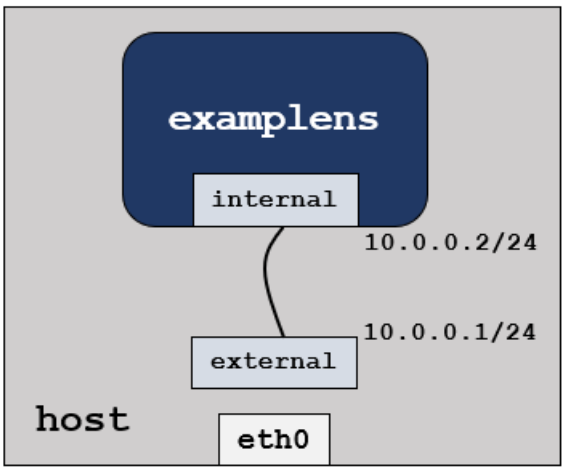


Figure 2 – Two-node scenario network.

Use the commands in the beginning of this section to create the namespace, add the interface pair, move one of them into the namespace and assign both their respective IP. Test connectivity between the namespaces using an appropriate command. Report the results.

4.2 Three-node network

For the second exercise you will build a network topology with three nodes connected in a chain, as shown in Figure 3.

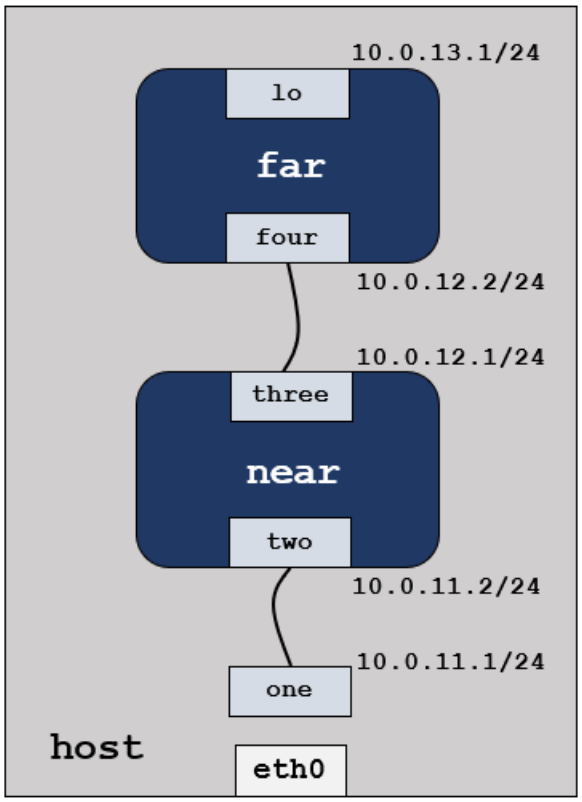


Figure 2 – Three-node in line scenario network.

Follow the same scheme as you did, for the first exercise, in creating namespaces and adding interfaces in with their respective IP address.

Add rules to the host and namespaces routing tables matching the networks (check the commands given in the beginning of this section). Don't forget to add routes to answer back the requests.

Even after adding routing rules, you will need to enable the IP Forwarding capability of Linux for namespaces that will have to do routing. You can do so with the following command:

```
ip netns exec <NSNAME> sysctl -w net.ipv4.ip_forward=1
```

The final objective is to have connectivity between all namespaces and be able to, from the default namespace, reach the `lo` interface of the `far` namespace. Test connectivity between the namespaces using an appropriate command. Report the results.

4.3 Full network emulation

For the third and final exercise you will emulate the topology presented in Figure 4.

You will use namespaces to emulate the network nodes. Assign namespaces to the three different networks represented in Figure 3 (you will choose all the networks' IPs).

Notice that in the previous exercise, the `near` namespace acted as a router. In this exercise you will use the same logic to implement routing abilities to the network.

Test and report different connectivity paths.

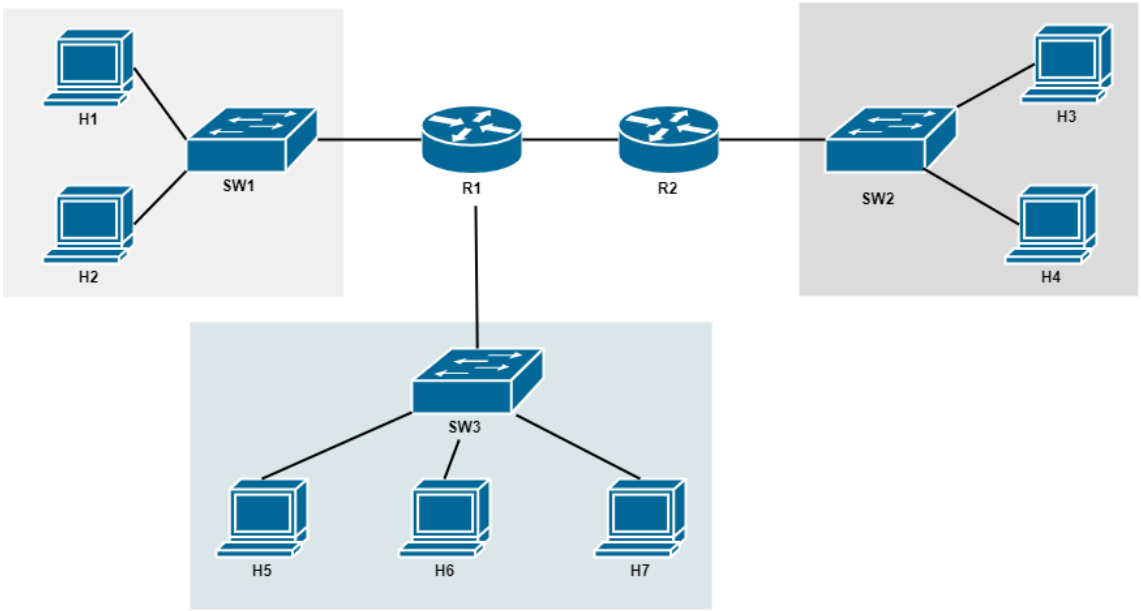


Figure 3 – Example topology with three different networks.