Universidad
de Alcalá

Developing a Network Virtualization Framework for Testing Network Resilience Techniques

Undergraduate Thesis

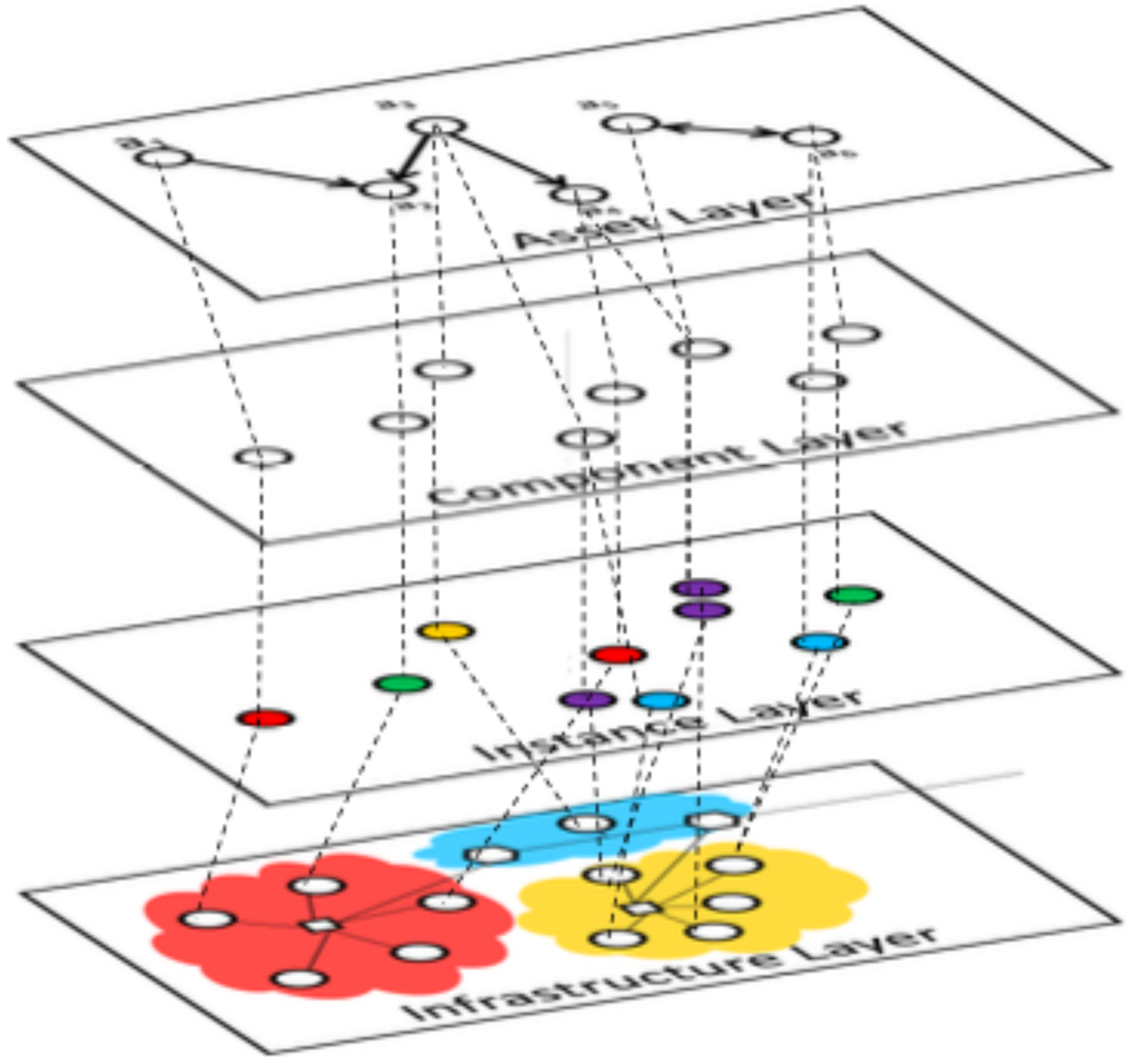
Pablo Collado Soto - 29/06/2021

Source

Source

Project’s Background

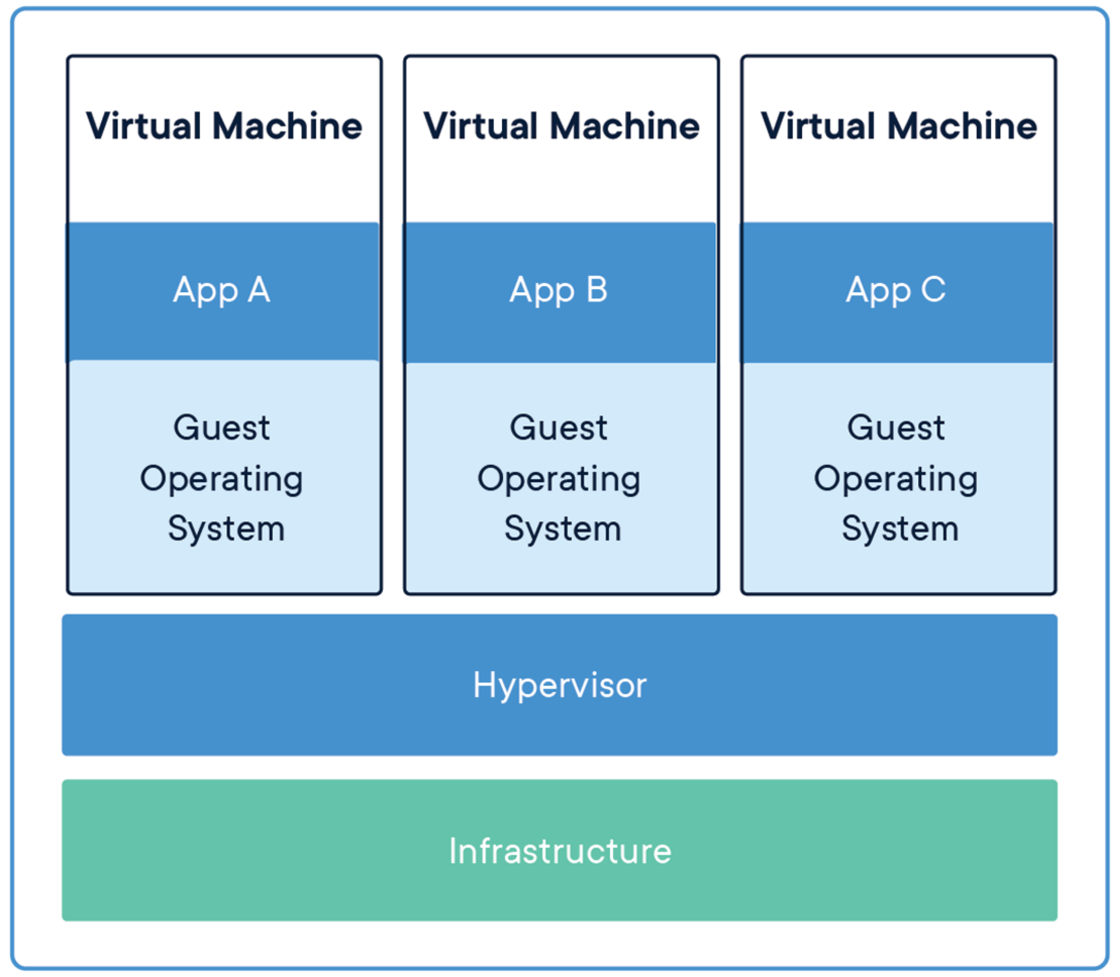
- Testing environment for the REACT framework.
- REACT -> allows the reconfiguration of networks to mitigate attacks.
- Testing environment -> fully virtual network.
- Reconfiguration -> altering network topology “on the fly”.
- Flexibility -> “in-house” solution.



Source

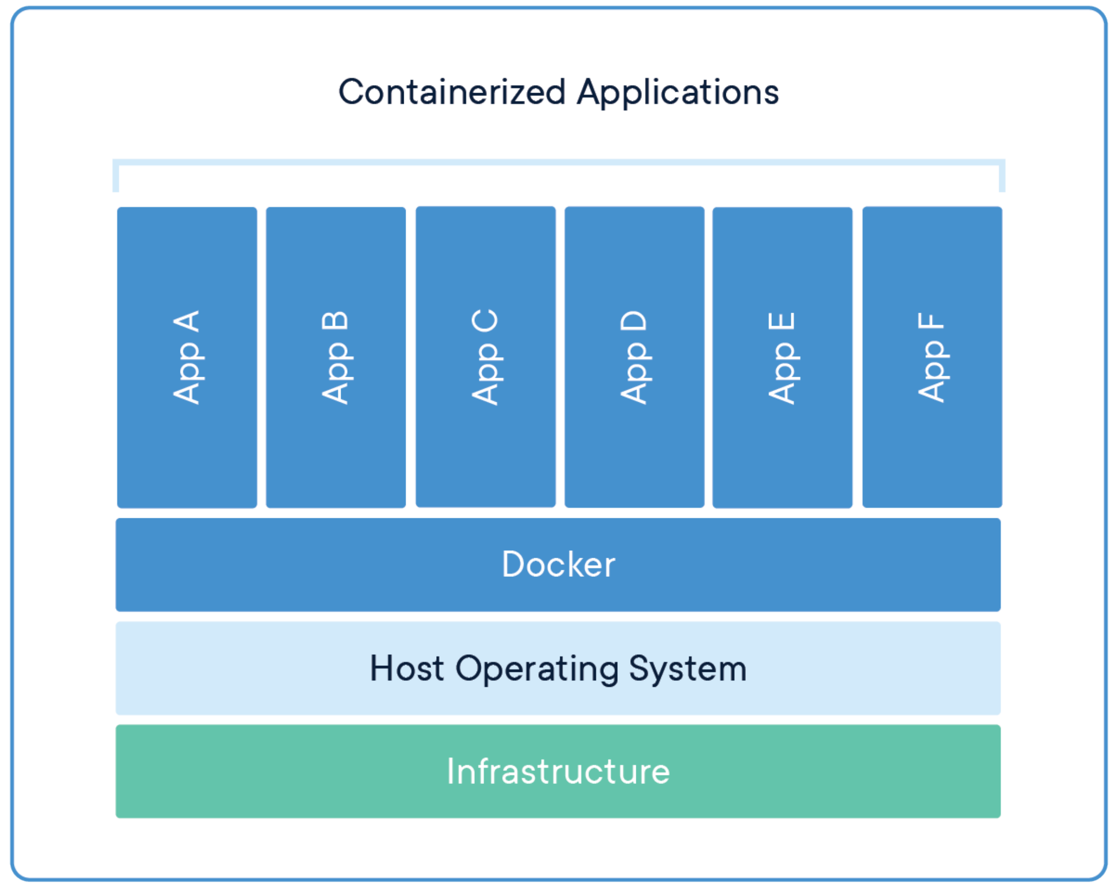
Initial Technology Choices

- *Layer 3* (L3) -> collection of `hosts` connected by `routers`.
- `Hosts` and `routers` -> virtual entities.
- Two contenders:
  - **Virtual Machines**
    - Resource intensive.
    - “Obscure” network configuration.
  - **Containers**
    - Lightweight.
    - Finer network control.



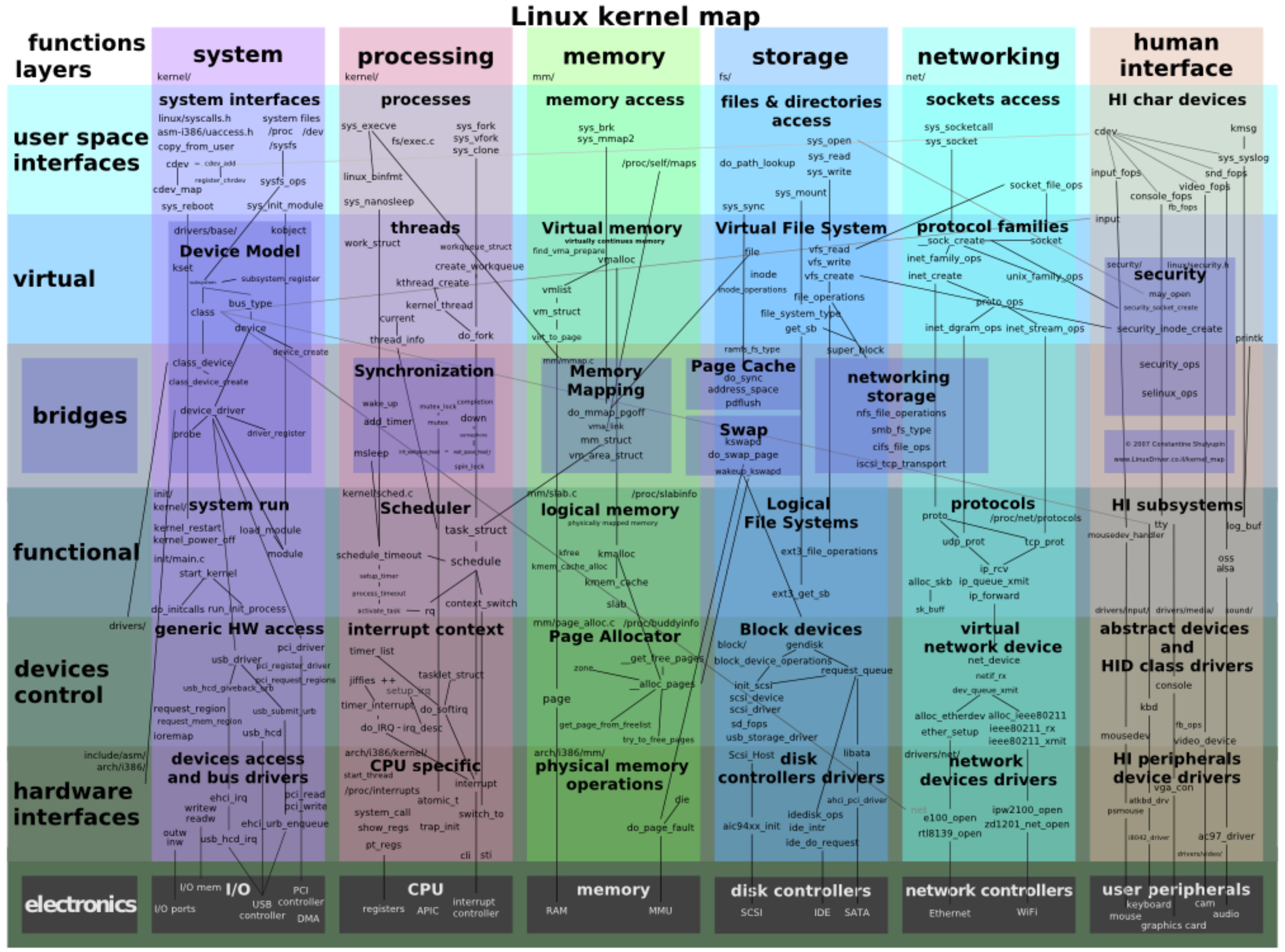
Source

Source



Initial Technology Choices (II)

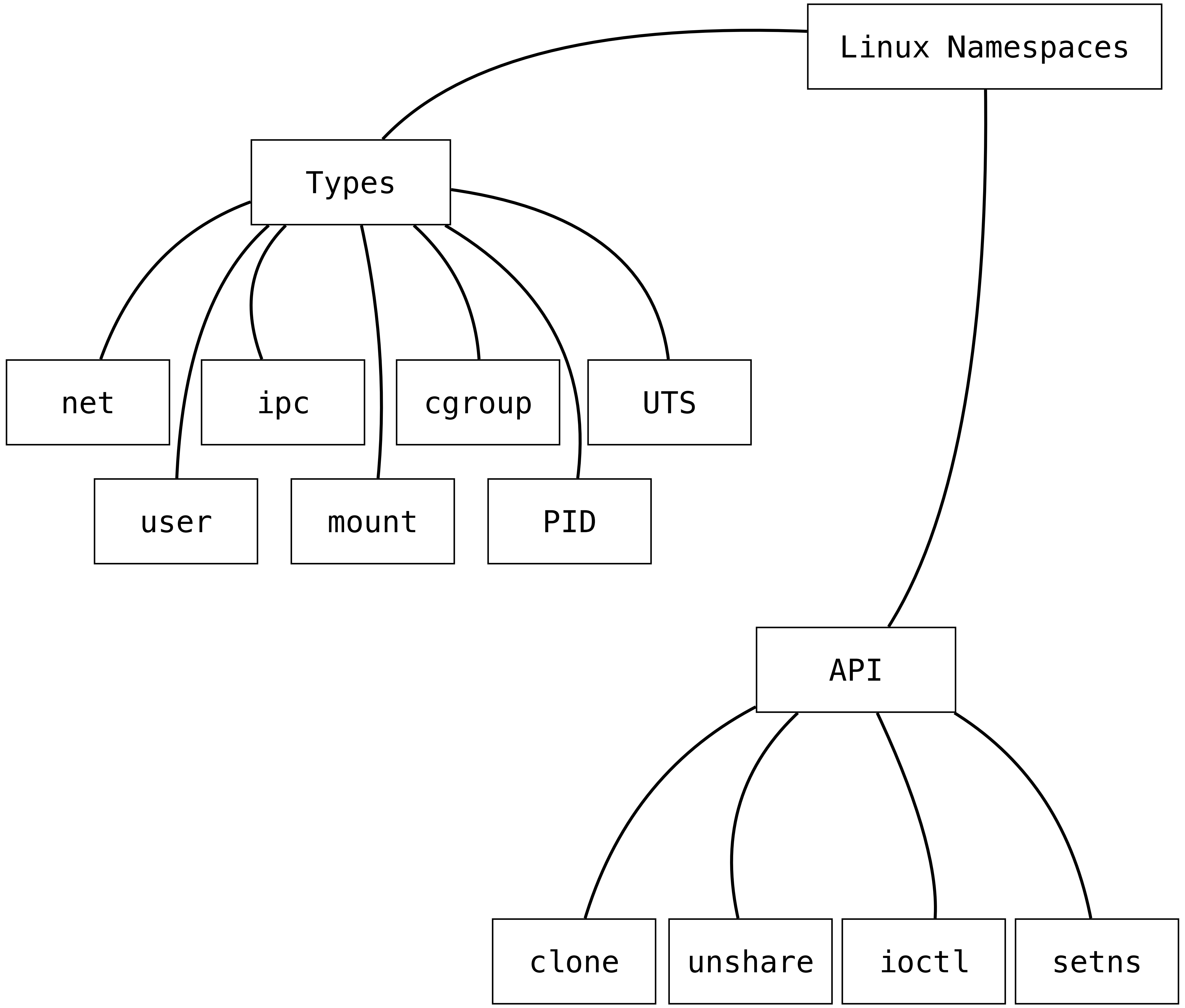
- *Layer 2* (L2) -> machines joined by `bridges`.
- `Bridge` -> “slower” `switch`, “zeroconf”, part of the `Linux kernel network stack`.
- Virtual “wires” -> `Virtual Ethernet Devices` (`veths`), part of the kernel, manifest as regular `NICs`.
- Linux kernel -> great option:
    - Host machine -> L2 backbone.
    - L3 nodes -> “virtualised” network stack supporting `veths`.
- **`namespaces`** -> several network stacks on a single machine.



Source

The Network Stack and the Namespace

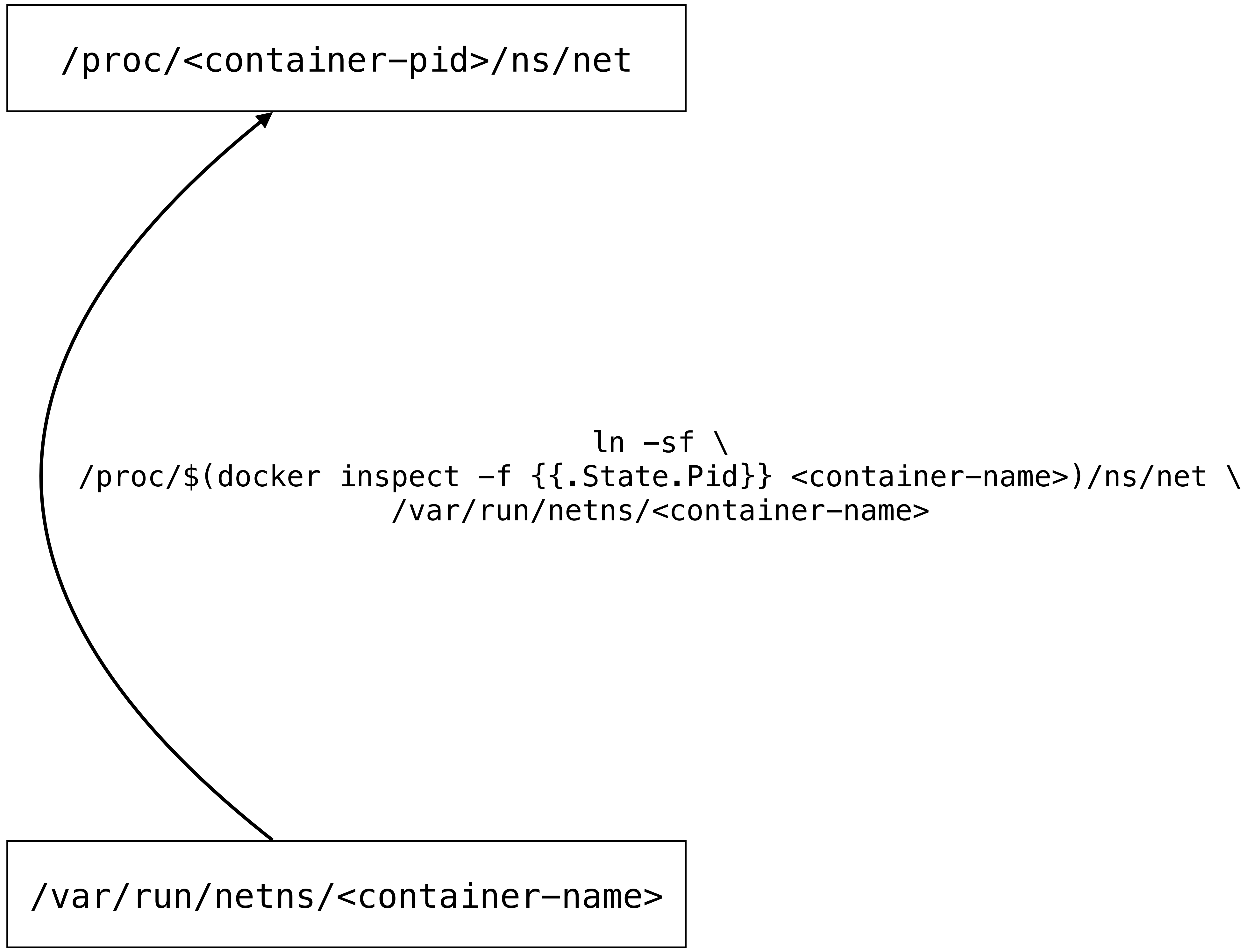
- Linux's network stack -> "blob" of code containing all the networking logic.
- Network stack -> logical entity in a network, usually 1 machine <-> 1 network stack.
- Virtual nodes -> independent network stack; two stacks = two virtual nodes.
- Namespaces -> partition kernel resources, openable objects.
- **Container** -> **its own network namespace**.



Based on this image.

Interacting With the Network Stack

- `iproute2` suite:
    - `netlink` interface with the kernel
    - Supersedes `ifconfig`
    - `CLI` interface with the user.
    - Instantiate and configure network -> on host and containers.
- `iproute2` commands -> can be run from **any** `namespace`.
- `Network` namespaces **must be** linked under `/var/run/netns/`.



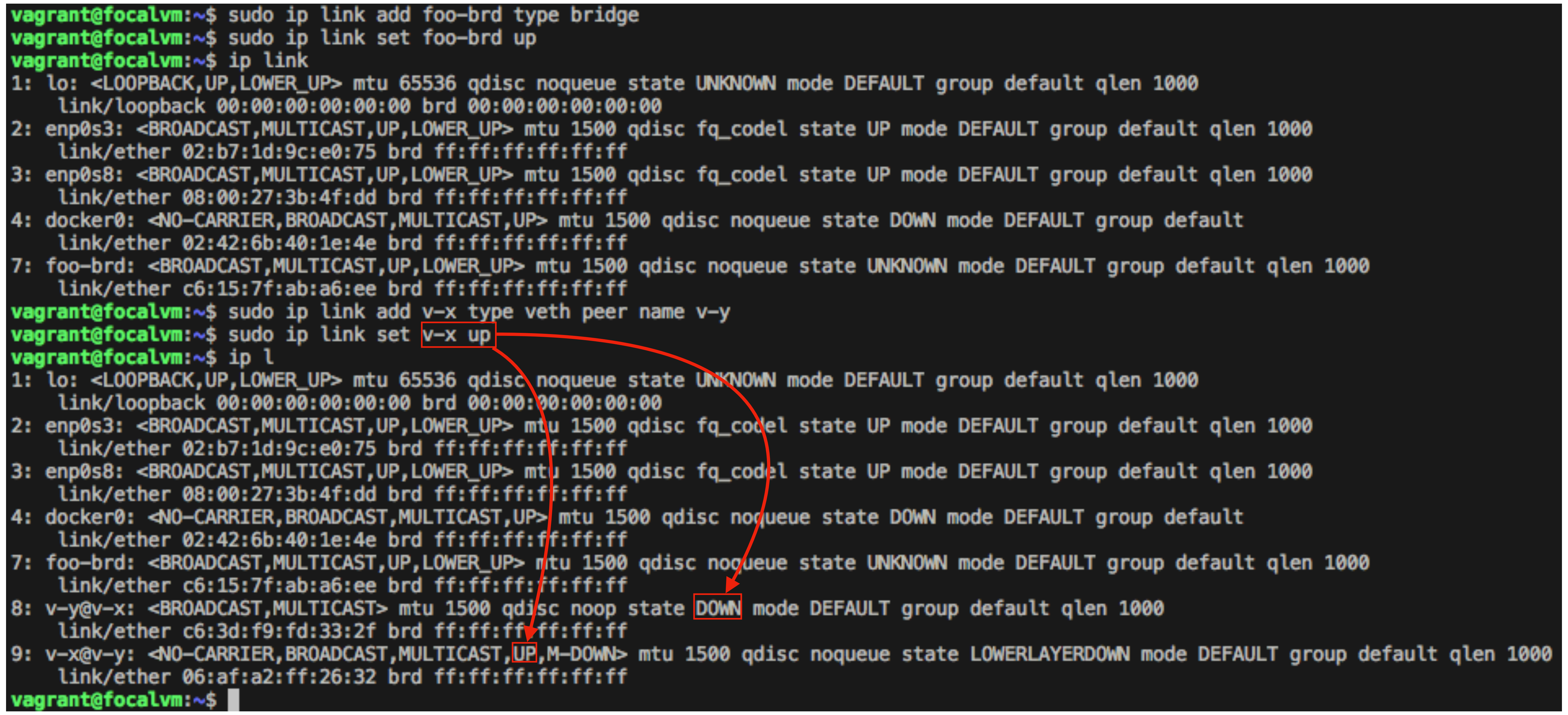
/proc/<container-pid>/ns/net

```
ln -sf \
/proc/$(docker inspect -f {{.State.Pid}} <container-name>)/ns/net \
/var/run/netns/<container-name>
```

/var/run/netns/<container-name>

Instantiating Network Elements

- **Bridges**:
  - Creation: `ip link add foo-brd type bridge`
  - Ignition: `ip link set foo-brd up`
- **Veths**:
  - Creation: `ip link add v-x type veth peer name v-y`
  - Ignition: `ip link set v-[x | y] up`
  - Connection to a bridge: `ip link set v-[x | y] master foo-brd`
  - Namespace change: `ip link set netns v-[x | y] netns-name`

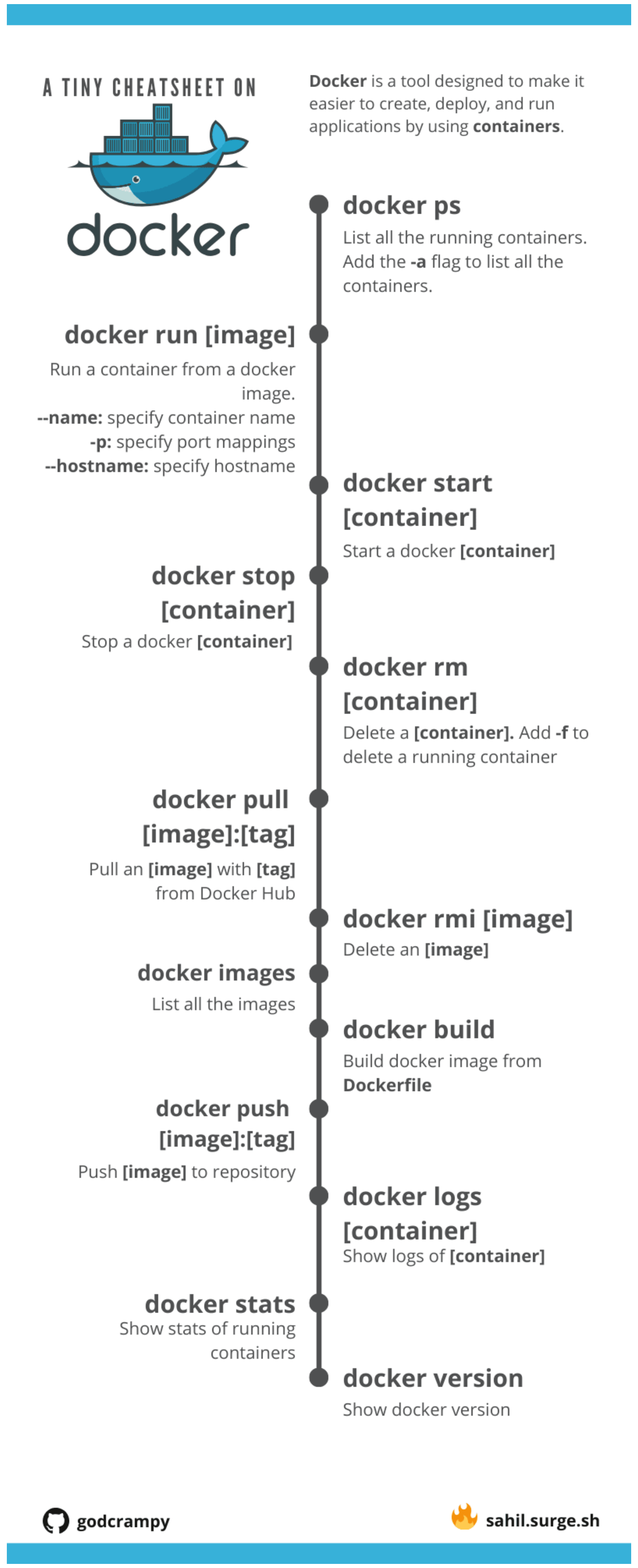


```
vagrant@focalvm:~$ sudo ip link add foo-brd type bridge
vagrant@focalvm:~$ sudo ip link set foo-brd up
vagrant@focalvm:~$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 02:b7:1d:9c:e0:75 brd ff:ff:ff:ff:ff:ff
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 08:00:27:3b:4f:dd brd ff:ff:ff:ff:ff:ff
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:6b:40:1e:4e brd ff:ff:ff:ff:ff:ff
7: foo-brd: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether c6:15:7f:ab:a6:ee brd ff:ff:ff:ff:ff:ff
vagrant@focalvm:~$ sudo ip link add v-x type veth peer name v-y
vagrant@focalvm:~$ sudo ip link set v-x up
vagrant@focalvm:~$ ip l
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 02:b7:1d:9c:e0:75 brd ff:ff:ff:ff:ff:ff
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 08:00:27:3b:4f:dd brd ff:ff:ff:ff:ff:ff
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:6b:40:1e:4e brd ff:ff:ff:ff:ff:ff
7: foo-brd: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether c6:15:7f:ab:a6:ee brd ff:ff:ff:ff:ff:ff
8: v-y@v-x: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether c6:3d:f9:fd:33:2f brd ff:ff:ff:ff:ff:ff
9: v-x@v-y: <NO-CARRIER,BROADCAST,MULTICAST,UP,M-DOWN> mtu 1500 qdisc noqueue state LOWERLAYERDOWN mode DEFAULT group default qlen 1000
    link/ether 06:af:a2:ff:26:32 brd ff:ff:ff:ff:ff:ff
vagrant@focalvm:~$
```

Instantiating Network Elements (II)

- Container -> runs an `image`.
- `Dockerfiles`
    - Define the contents of an `image`.
    - Are `built` to produce images.
- Built images -> run within a container.

```
docker run -d --name <container-name> --network none --cap-add … <img-name>
```



A TINY CHEATSHEET ON docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using **containers**.

docker ps
List all the running containers. Add the **-a** flag to list all the containers.

docker run [image]
Run a container from a docker image.
--name: specify container name
-p: specify port mappings
--hostname: specify hostname

docker start [container]
Start a docker **[container]**

docker stop [container]
Stop a docker **[container]**

docker rm [container]
Delete a **[container]**. Add **-f** to delete a running container

docker pull [image]:[tag]
Pull an **[image]** with **[tag]** from Docker Hub

docker rmi [image]
Delete an **[image]**

docker images
List all the images

docker build
Build docker image from **Dockerfile**

docker push [image]:[tag]
Push **[image]** to repository

docker logs [container]
Show logs of **[container]**

docker stats
Show stats of running containers

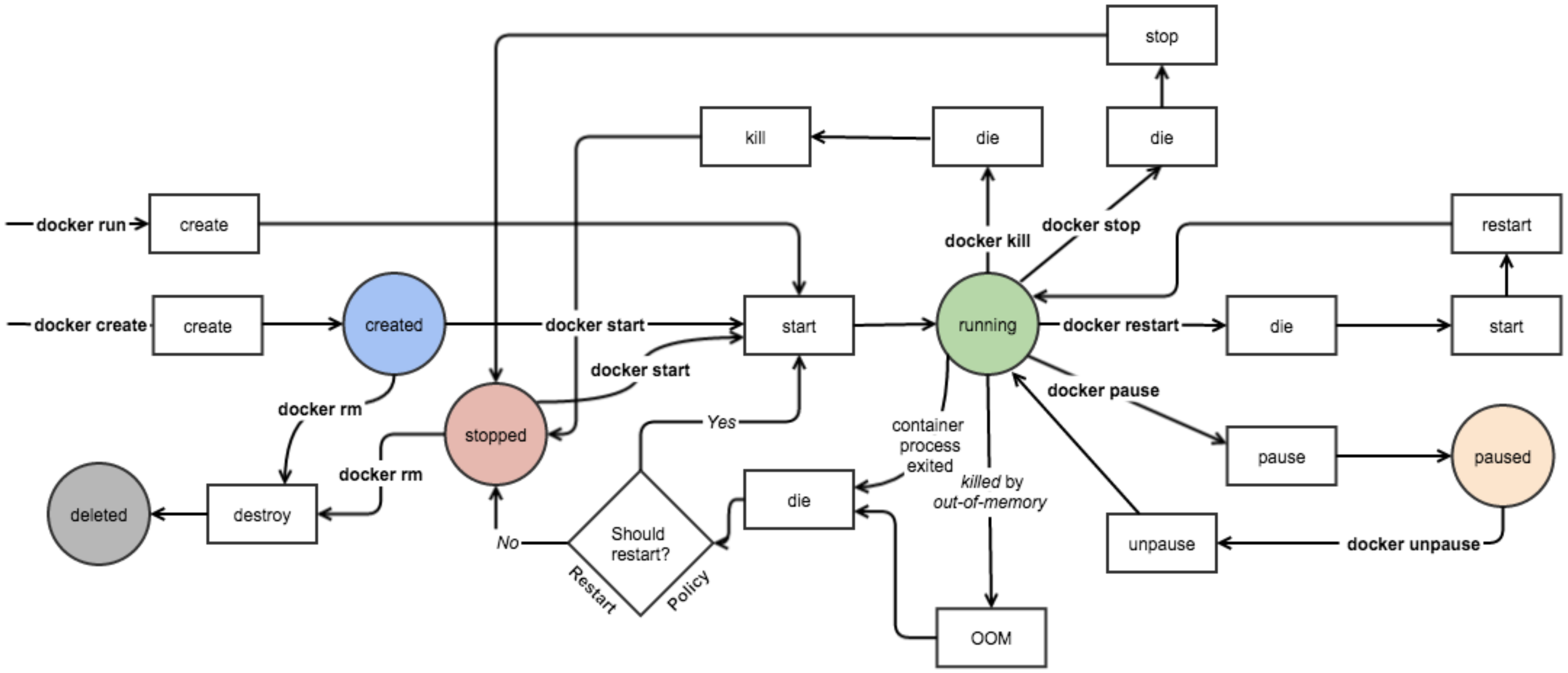
docker version
Show docker version

godcrampy **sahil.surge.sh**

Source

A Container’s Lifecycle

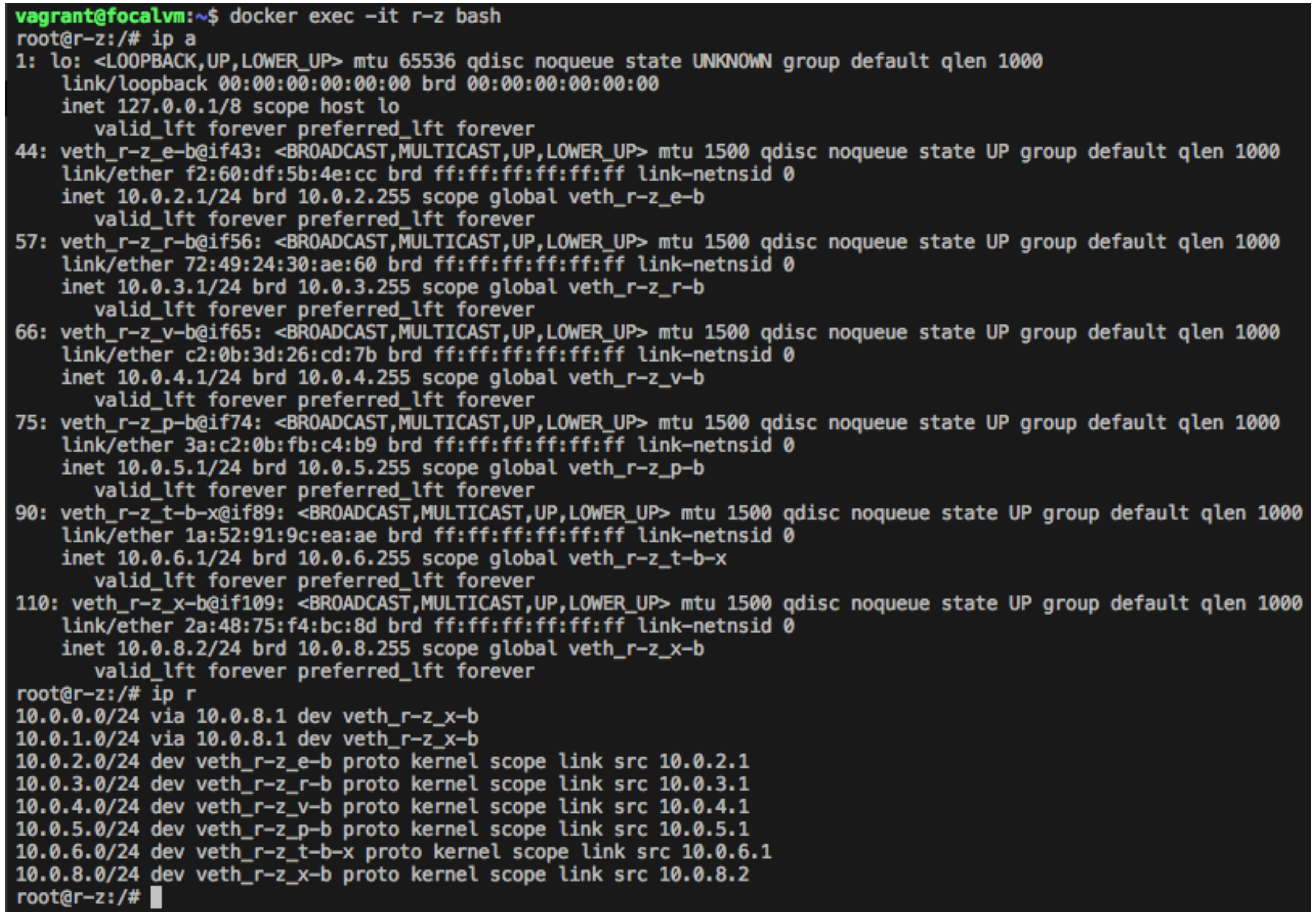
- `sshd` daemon -> keeps the container alive.
- `sshd` will not exit -> containers will `run` indefinitely.



Source

Routing and Addressing

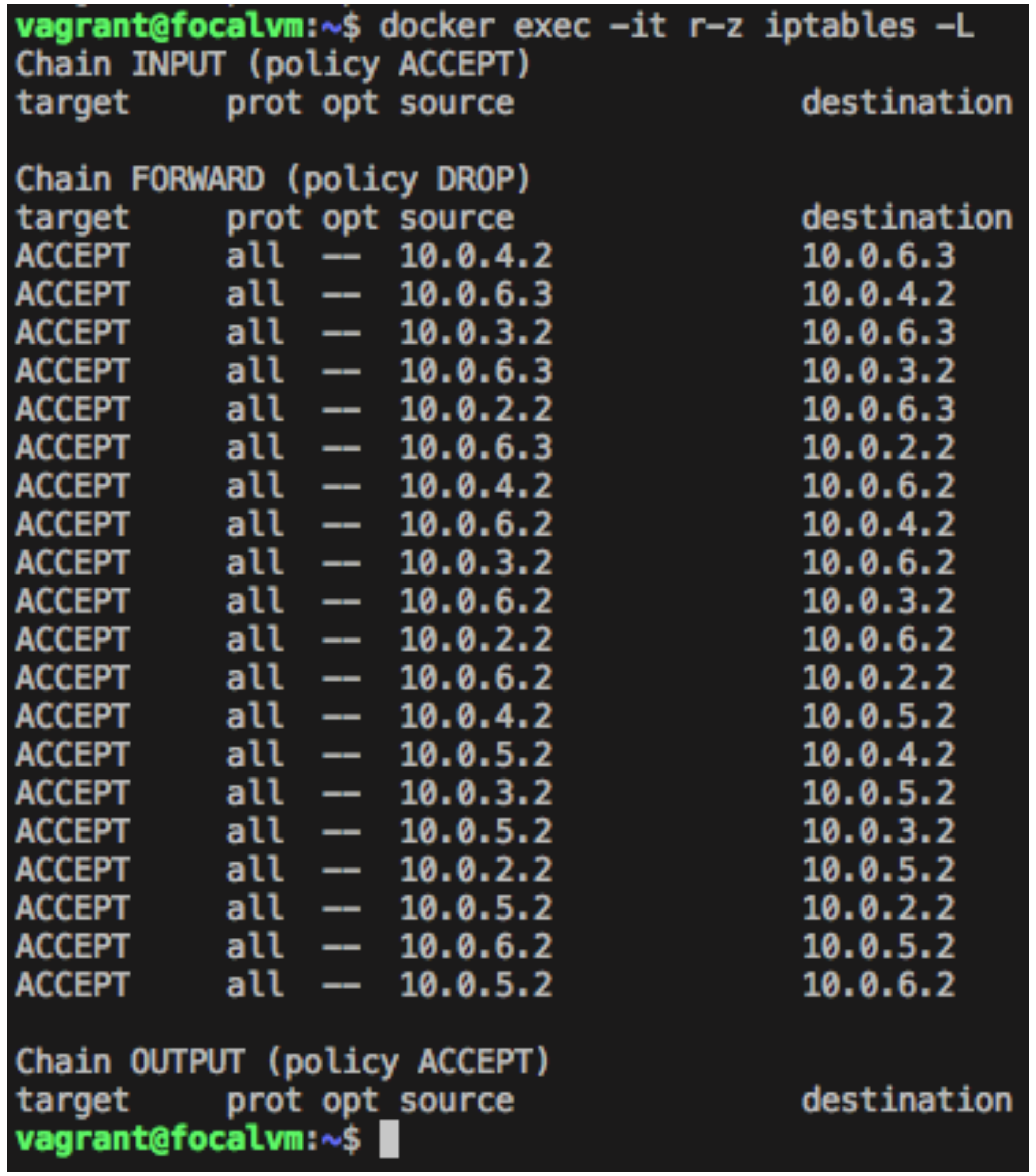
- `iproute2` -> manages addressing and routing.
- Option `–n <ns–name>`:
  - Run a command within any namespace.
  - Enables container configuration form the `root namespace`.
- Machines on same subnet -> automatic route.
- Address assignment:
  - `ip –n <cont> a add <IPv4>/<netmask> brd + dev <veth>`
- Route assignment:
  - `ip –n <cont> route add default via <gateway–IPv4>`
- Namespace -> associated container's name; configuration of containers through their name.



```
vagrant@focalvm:~$ docker exec -it r-z bash
root@r-z:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
44: veth_r-z_e-b@if43: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether f2:60:df:5b:4e:cc brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.2.1/24 brd 10.0.2.255 scope global veth_r-z_e-b
       valid_lft forever preferred_lft forever
57: veth_r-z_r-b@if56: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 72:49:24:30:ae:60 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.3.1/24 brd 10.0.3.255 scope global veth_r-z_r-b
       valid_lft forever preferred_lft forever
66: veth_r-z_v-b@if65: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether c2:0b:3d:26:cd:7b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.4.1/24 brd 10.0.4.255 scope global veth_r-z_v-b
       valid_lft forever preferred_lft forever
75: veth_r-z_p-b@if74: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 3a:c2:0b:fb:c4:b9 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.5.1/24 brd 10.0.5.255 scope global veth_r-z_p-b
       valid_lft forever preferred_lft forever
90: veth_r-z_t-b-x@if89: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 1a:52:91:9c:ea:ae brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.6.1/24 brd 10.0.6.255 scope global veth_r-z_t-b-x
       valid_lft forever preferred_lft forever
110: veth_r-z_x-b@if109: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 2a:48:75:f4:bc:8d brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.8.2/24 brd 10.0.8.255 scope global veth_r-z_x-b
       valid_lft forever preferred_lft forever
root@r-z:/# ip r
10.0.0.0/24 via 10.0.8.1 dev veth_r-z_x-b
10.0.1.0/24 via 10.0.8.1 dev veth_r-z_x-b
10.0.2.0/24 dev veth_r-z_e-b proto kernel scope link src 10.0.2.1
10.0.3.0/24 dev veth_r-z_r-b proto kernel scope link src 10.0.3.1
10.0.4.0/24 dev veth_r-z_v-b proto kernel scope link src 10.0.4.1
10.0.5.0/24 dev veth_r-z_p-b proto kernel scope link src 10.0.5.1
10.0.6.0/24 dev veth_r-z_t-b-x proto kernel scope link src 10.0.6.1
10.0.8.0/24 dev veth_r-z_x-b proto kernel scope link src 10.0.8.2
root@r-z:/# 
```

Firewall Configuration

- Topology needs -> forbid some connections.
- `iptables ->` chosen firewall implementation.
- Configuration -> based on `chains` defined by:
  - What packets are checked against it: `INPUT`, `OUTPUT`, `FORWARD`.
  - Default policy: `DROP`, `ACCEPT`.
  - Rules -> associated to a targets: `DROP`, `ACCEPT`.
- Firewall rules are applied within a given `namespace`.

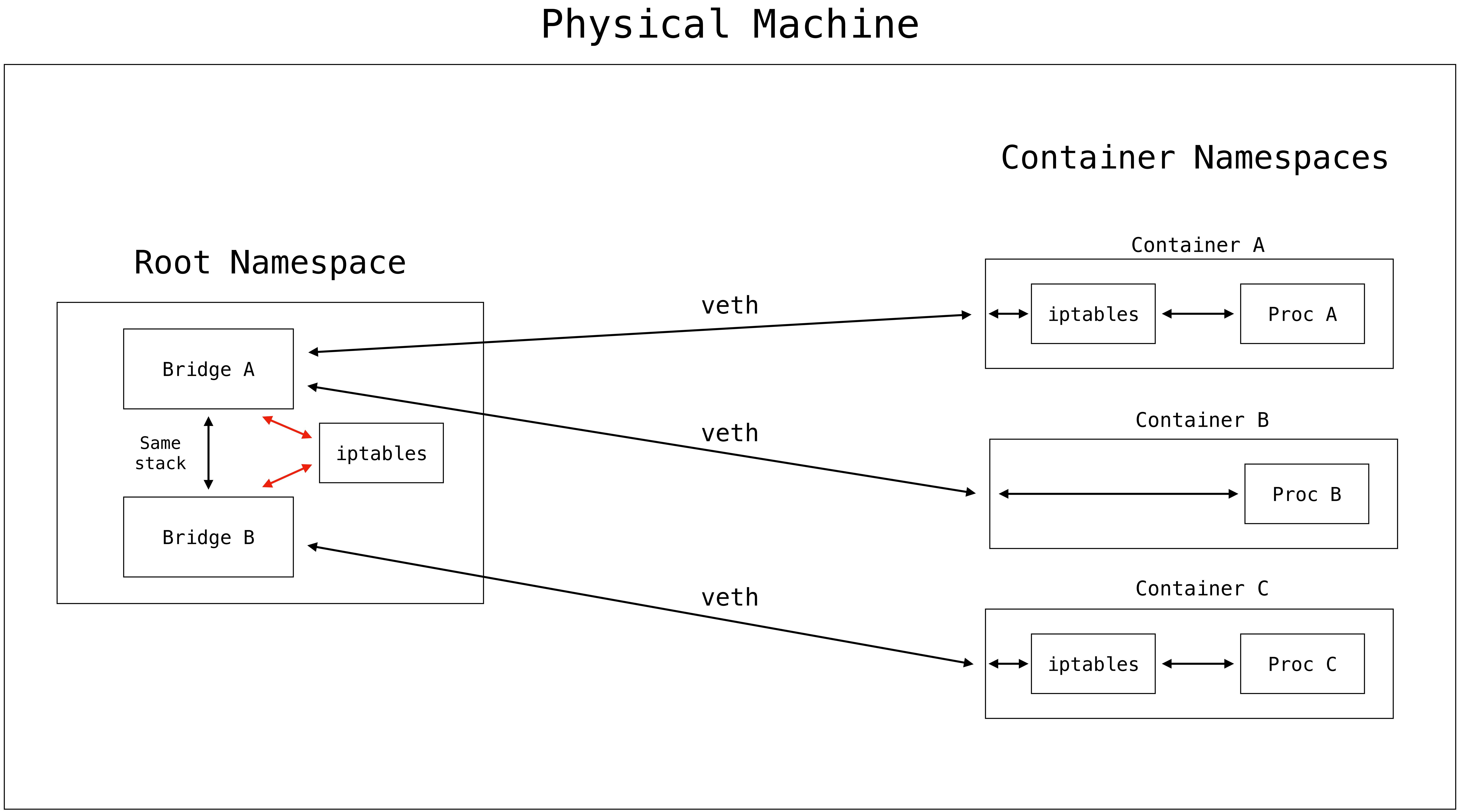


```
vagrant@focalvm:~$ docker exec -it r-z iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  10.0.4.2             10.0.6.3
ACCEPT     all  --  10.0.6.3             10.0.4.2
ACCEPT     all  --  10.0.3.2             10.0.6.3
ACCEPT     all  --  10.0.6.3             10.0.3.2
ACCEPT     all  --  10.0.2.2             10.0.6.3
ACCEPT     all  --  10.0.6.3             10.0.2.2
ACCEPT     all  --  10.0.4.2             10.0.6.2
ACCEPT     all  --  10.0.6.2             10.0.4.2
ACCEPT     all  --  10.0.3.2             10.0.6.2
ACCEPT     all  --  10.0.6.2             10.0.3.2
ACCEPT     all  --  10.0.2.2             10.0.6.2
ACCEPT     all  --  10.0.6.2             10.0.2.2
ACCEPT     all  --  10.0.4.2             10.0.5.2
ACCEPT     all  --  10.0.5.2             10.0.4.2
ACCEPT     all  --  10.0.3.2             10.0.5.2
ACCEPT     all  --  10.0.5.2             10.0.3.2
ACCEPT     all  --  10.0.2.2             10.0.5.2
ACCEPT     all  --  10.0.5.2             10.0.2.2
ACCEPT     all  --  10.0.6.2             10.0.5.2
ACCEPT     all  --  10.0.5.2             10.0.6.2

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
vagrant@focalvm:~$ 
```

Subtleties

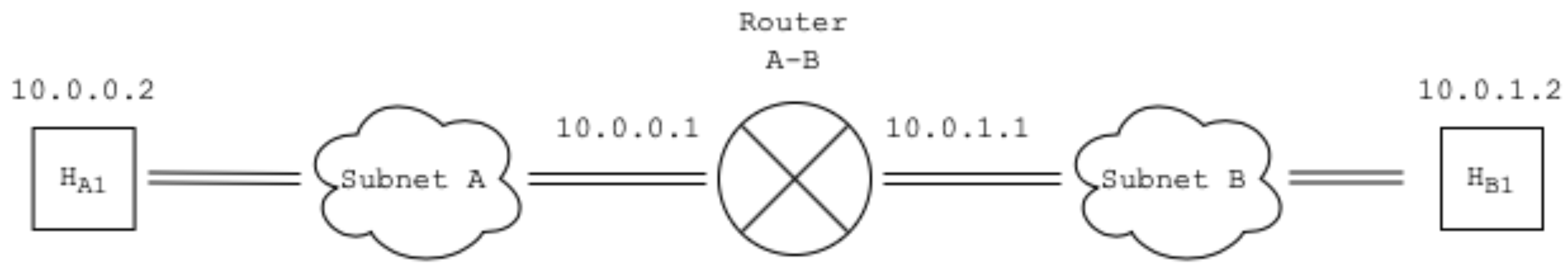
- Frames passing through `bridges` -> **checked** against `iptables` by default.
- Disabling the behaviour through `sysctl`:
  - Temporarily: `sysctl -w net.bridge.bridge-nf-call-iptables=0`
  - Permanently: Editing `/etc/sysctl.conf` and running `sysctl -p`.
- Machine -> **not allowed** to forward packets at L3 by default.
- Enabling the behaviour through `sysctl`:
  - Temporarily: `sysctl -w net.ipv4.ip_forward=1`
  - Permanently: Editing `/etc/sysctl.conf` and running `sysctl -p`.

“Physical” Topology



A Simple Topology

- Listing 3.2 -> `bash script` instantiating a simple topology.
- Network’s size increase -> script complexity and length grow quickly.
- Script -> no way of automatically altering the topology.
- System needs -> automatically instantiate and manage a network.



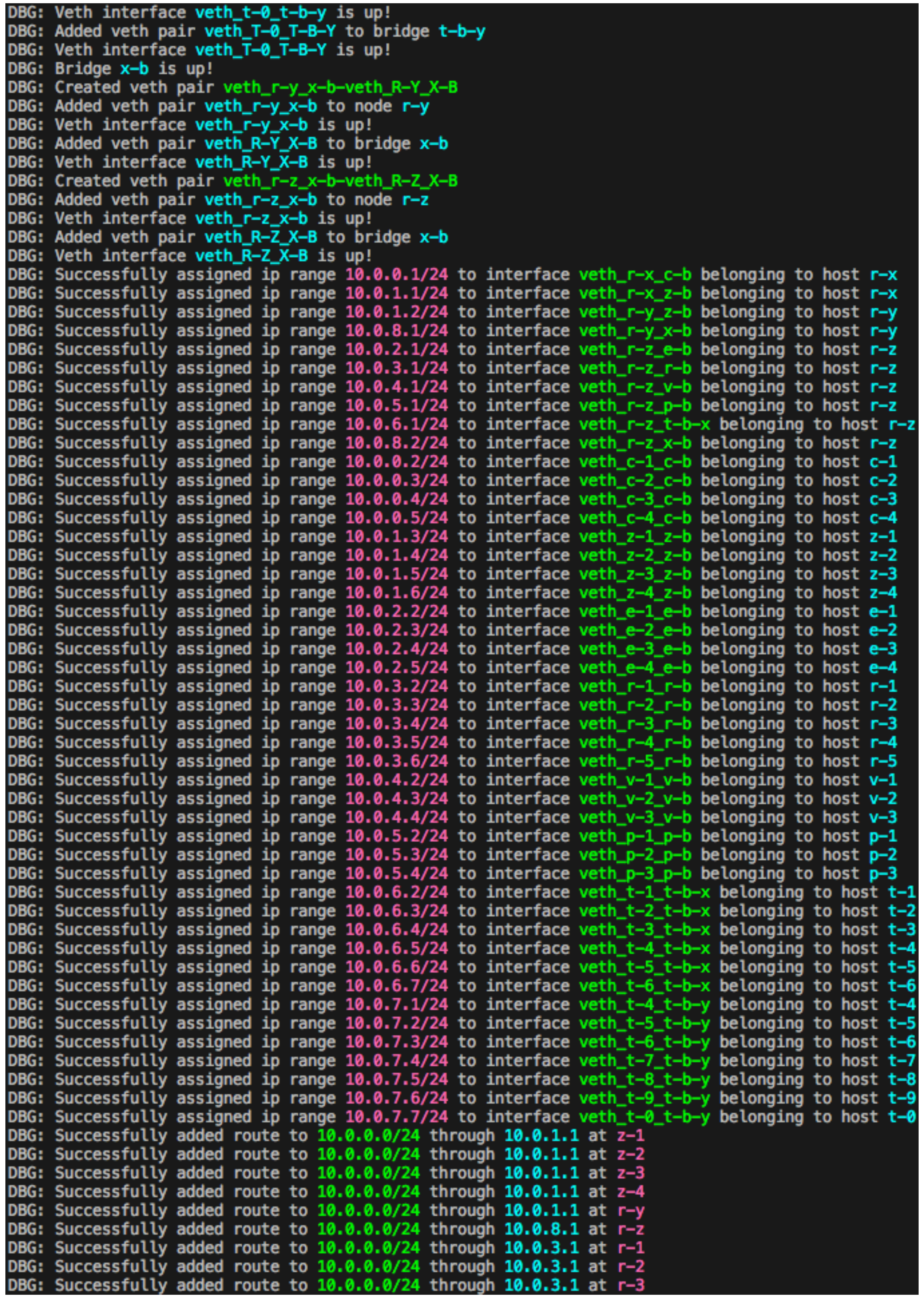
Sample Topology

- **Subnet A -> 10.0.0.0/24**
- **Subnet B -> 10.0.1.0/24**

H_{XY} -> Host Y belonging to subnet X

Our Tool

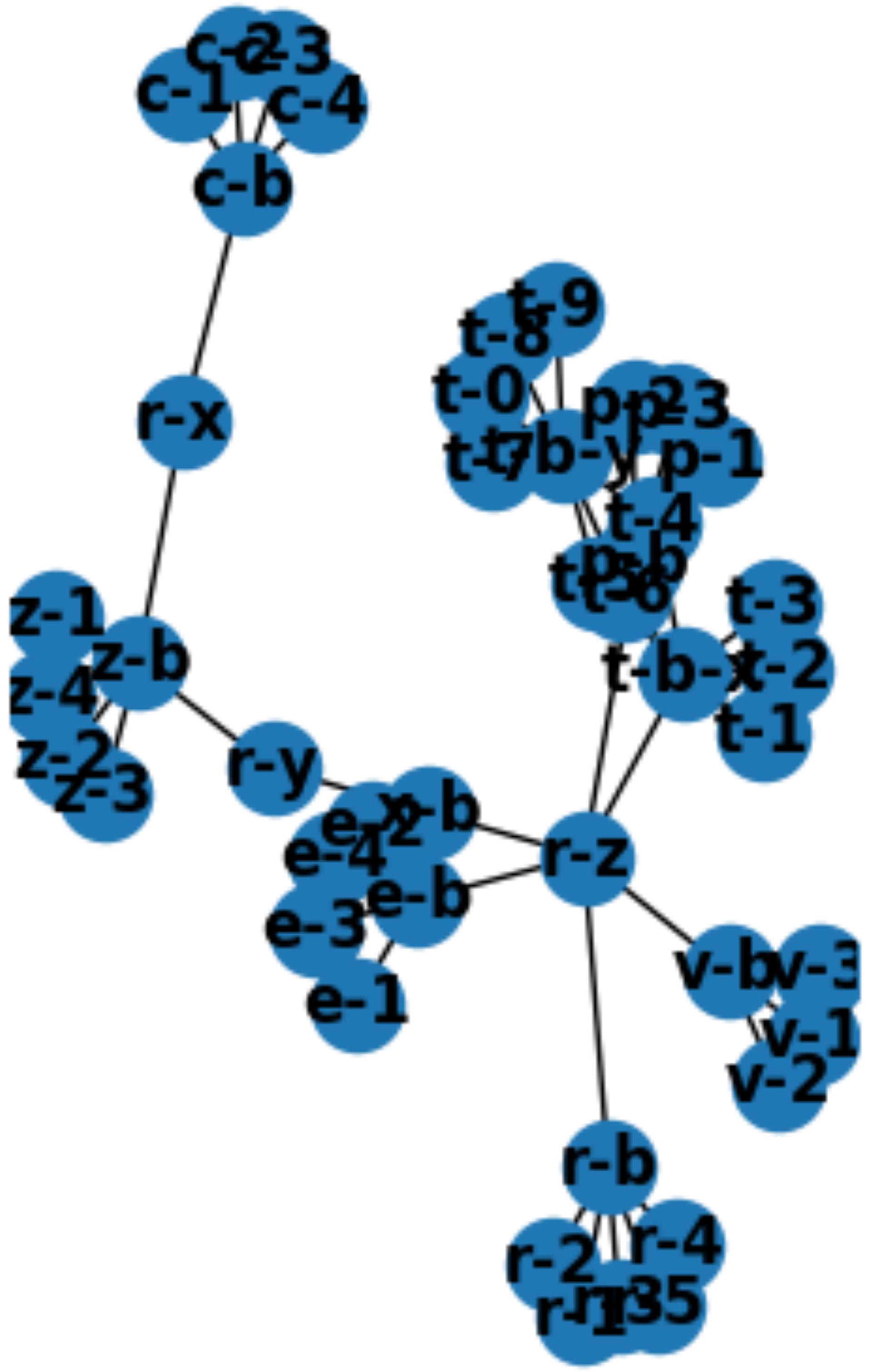
- Solution -> `python3` program overcoming limitations.
- Leverages *Object Oriented Programming*.
- Interface for `iproute2` and `docker -> os.system()`.
- `NetworkX` -> model networks as graphs: flexible network management.
- Alternative -> independent solution of `NetworkX`; less robust.
- User input -> graph representing desired network.
- User interface -> simple `CLI`.
- Basic commands:
    - `mvnode`: Allows to move a node to a new place within the network.
    - `lsnet`: Shows a graphical representation of the current network.



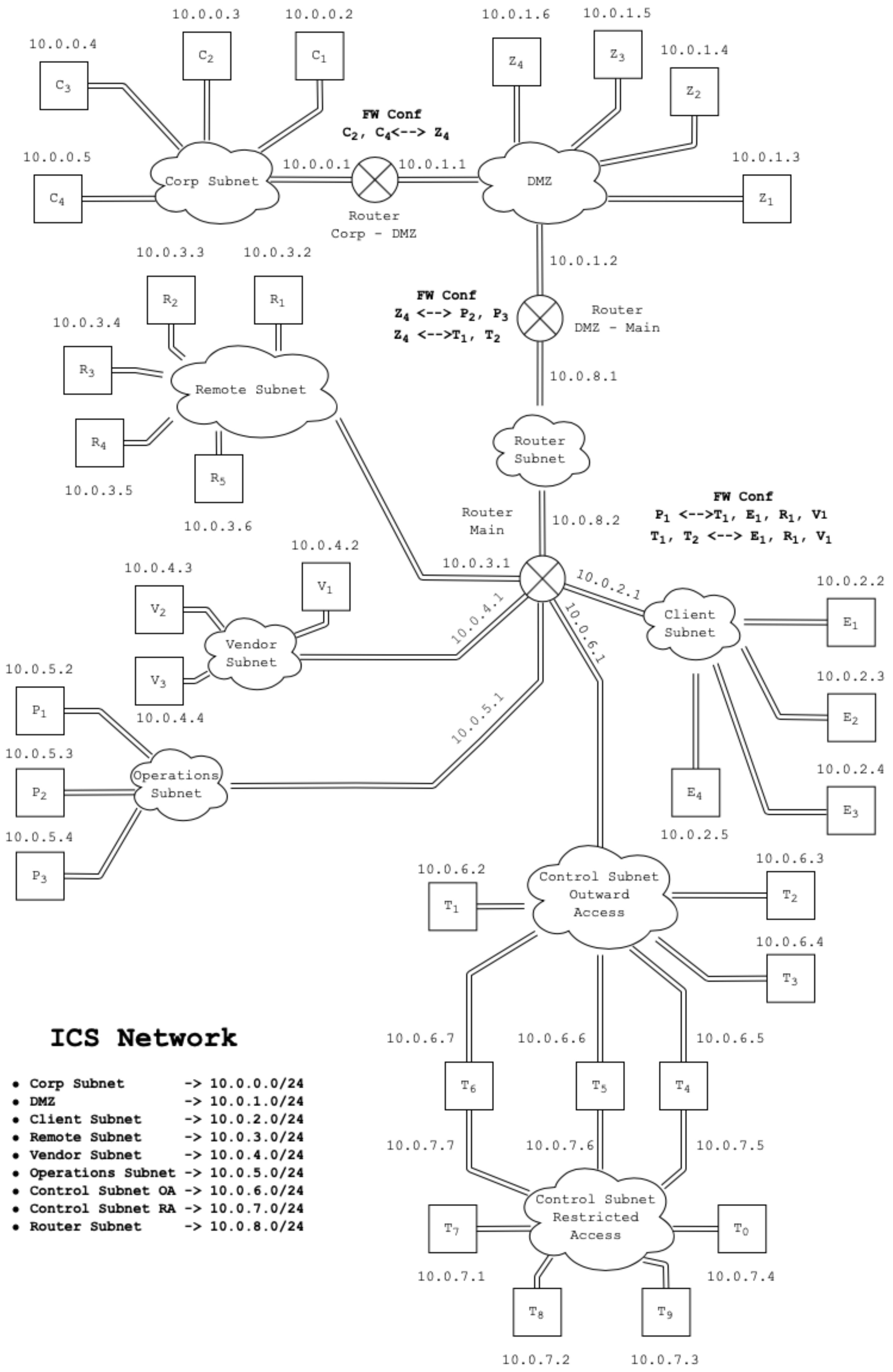
```
DBG: Veth interface veth_t-0_t-b-y is up!
DBG: Added veth pair veth_T-0_T-B-Y to bridge t-b-y
DBG: Veth interface veth_T-0_T-B-Y is up!
DBG: Bridge x-b is up!
DBG: Created veth pair veth_r-y_x-b-veth_R-Y_X-B
DBG: Added veth pair veth_r-y_x-b to node r-y
DBG: Veth interface veth_r-y_x-b is up!
DBG: Added veth pair veth_R-Y_X-B to bridge x-b
DBG: Veth interface veth_R-Y_X-B is up!
DBG: Created veth pair veth_r-z_x-b-veth_R-Z_X-B
DBG: Added veth pair veth_r-z_x-b to node r-z
DBG: Veth interface veth_r-z_x-b is up!
DBG: Added veth pair veth_R-Z_X-B to bridge x-b
DBG: Veth interface veth_R-Z_X-B is up!
DBG: Successfully assigned ip range 10.0.0.1/24 to interface veth_r-x_c-b belonging to host r-x
DBG: Successfully assigned ip range 10.0.1.1/24 to interface veth_r-x_z-b belonging to host r-x
DBG: Successfully assigned ip range 10.0.1.2/24 to interface veth_r-y_z-b belonging to host r-y
DBG: Successfully assigned ip range 10.0.8.1/24 to interface veth_r-y_x-b belonging to host r-y
DBG: Successfully assigned ip range 10.0.2.1/24 to interface veth_r-z_e-b belonging to host r-z
DBG: Successfully assigned ip range 10.0.3.1/24 to interface veth_r-z_r-b belonging to host r-z
DBG: Successfully assigned ip range 10.0.4.1/24 to interface veth_r-z_v-b belonging to host r-z
DBG: Successfully assigned ip range 10.0.5.1/24 to interface veth_r-z_p-b belonging to host r-z
DBG: Successfully assigned ip range 10.0.6.1/24 to interface veth_r-z_t-b-x belonging to host r-z
DBG: Successfully assigned ip range 10.0.8.2/24 to interface veth_r-z_x-b belonging to host r-z
DBG: Successfully assigned ip range 10.0.0.2/24 to interface veth_c-1_c-b belonging to host c-1
DBG: Successfully assigned ip range 10.0.0.3/24 to interface veth_c-2_c-b belonging to host c-2
DBG: Successfully assigned ip range 10.0.0.4/24 to interface veth_c-3_c-b belonging to host c-3
DBG: Successfully assigned ip range 10.0.0.5/24 to interface veth_c-4_c-b belonging to host c-4
DBG: Successfully assigned ip range 10.0.1.3/24 to interface veth_z-1_z-b belonging to host z-1
DBG: Successfully assigned ip range 10.0.1.4/24 to interface veth_z-2_z-b belonging to host z-2
DBG: Successfully assigned ip range 10.0.1.5/24 to interface veth_z-3_z-b belonging to host z-3
DBG: Successfully assigned ip range 10.0.1.6/24 to interface veth_z-4_z-b belonging to host z-4
DBG: Successfully assigned ip range 10.0.2.2/24 to interface veth_e-1_e-b belonging to host e-1
DBG: Successfully assigned ip range 10.0.2.3/24 to interface veth_e-2_e-b belonging to host e-2
DBG: Successfully assigned ip range 10.0.2.4/24 to interface veth_e-3_e-b belonging to host e-3
DBG: Successfully assigned ip range 10.0.2.5/24 to interface veth_e-4_e-b belonging to host e-4
DBG: Successfully assigned ip range 10.0.3.2/24 to interface veth_r-1_r-b belonging to host r-1
DBG: Successfully assigned ip range 10.0.3.3/24 to interface veth_r-2_r-b belonging to host r-2
DBG: Successfully assigned ip range 10.0.3.4/24 to interface veth_r-3_r-b belonging to host r-3
DBG: Successfully assigned ip range 10.0.3.5/24 to interface veth_r-4_r-b belonging to host r-4
DBG: Successfully assigned ip range 10.0.3.6/24 to interface veth_r-5_r-b belonging to host r-5
DBG: Successfully assigned ip range 10.0.4.2/24 to interface veth_v-1_v-b belonging to host v-1
DBG: Successfully assigned ip range 10.0.4.3/24 to interface veth_v-2_v-b belonging to host v-2
DBG: Successfully assigned ip range 10.0.4.4/24 to interface veth_v-3_v-b belonging to host v-3
DBG: Successfully assigned ip range 10.0.5.2/24 to interface veth_p-1_p-b belonging to host p-1
DBG: Successfully assigned ip range 10.0.5.3/24 to interface veth_p-2_p-b belonging to host p-2
DBG: Successfully assigned ip range 10.0.5.4/24 to interface veth_p-3_p-b belonging to host p-3
DBG: Successfully assigned ip range 10.0.6.2/24 to interface veth_t-1_t-b-x belonging to host t-1
DBG: Successfully assigned ip range 10.0.6.3/24 to interface veth_t-2_t-b-x belonging to host t-2
DBG: Successfully assigned ip range 10.0.6.4/24 to interface veth_t-3_t-b-x belonging to host t-3
DBG: Successfully assigned ip range 10.0.6.5/24 to interface veth_t-4_t-b-x belonging to host t-4
DBG: Successfully assigned ip range 10.0.6.6/24 to interface veth_t-5_t-b-x belonging to host t-5
DBG: Successfully assigned ip range 10.0.6.7/24 to interface veth_t-6_t-b-x belonging to host t-6
DBG: Successfully assigned ip range 10.0.7.1/24 to interface veth_t-4_t-b-y belonging to host t-4
DBG: Successfully assigned ip range 10.0.7.2/24 to interface veth_t-5_t-b-y belonging to host t-5
DBG: Successfully assigned ip range 10.0.7.3/24 to interface veth_t-6_t-b-y belonging to host t-6
DBG: Successfully assigned ip range 10.0.7.4/24 to interface veth_t-7_t-b-y belonging to host t-7
DBG: Successfully assigned ip range 10.0.7.5/24 to interface veth_t-8_t-b-y belonging to host t-8
DBG: Successfully assigned ip range 10.0.7.6/24 to interface veth_t-9_t-b-y belonging to host t-9
DBG: Successfully assigned ip range 10.0.7.7/24 to interface veth_t-0_t-b-y belonging to host t-0
DBG: Successfully added route to 10.0.0.0/24 through 10.0.1.1 at z-1
DBG: Successfully added route to 10.0.0.0/24 through 10.0.1.1 at z-2
DBG: Successfully added route to 10.0.0.0/24 through 10.0.1.1 at z-3
DBG: Successfully added route to 10.0.0.0/24 through 10.0.1.1 at z-4
DBG: Successfully added route to 10.0.0.0/24 through 10.0.1.1 at r-y
DBG: Successfully added route to 10.0.0.0/24 through 10.0.8.1 at r-z
DBG: Successfully added route to 10.0.0.0/24 through 10.0.3.1 at r-1
DBG: Successfully added route to 10.0.0.0/24 through 10.0.3.1 at r-2
DBG: Successfully added route to 10.0.0.0/24 through 10.0.3.1 at r-3
```

Tool Modules

- Tool architecture -> 3 separate modules:
  - **`virt_net`**
    - Instantiation, configuration and addressing of network elements.
    - Classes representing network elements: `veths` and `routers`.
  - **`graph_interpreter`**
    - Translates graphs provided by users to classes defined in `virt_net`.
    - Contains the logic of features such as:
      - Node movement.
      - Network's routing.
  - **`net_ctrl`**
    - Implements the `CLI` presented to users.
    - Commands are converted to calls to the `graph_interpreter` module.



Our Most Complex Topology



ICS Network

- **Corp Subnet -> 10.0.0.0/24**
- **DMZ -> 10.0.1.0/24**
- **Client Subnet -> 10.0.2.0/24**
- **Remote Subnet -> 10.0.3.0/24**
- **Vendor Subnet -> 10.0.4.0/24**
- **Operations Subnet -> 10.0.5.0/24**
- **Control Subnet OA -> 10.0.6.0/24**
- **Control Subnet RA -> 10.0.7.0/24**
- **Router Subnet -> 10.0.8.0/24**

Based on figure 4 of “Improving ICS cyber resilience through optimal diversification of network resources”. Online article.

Proof of Concept

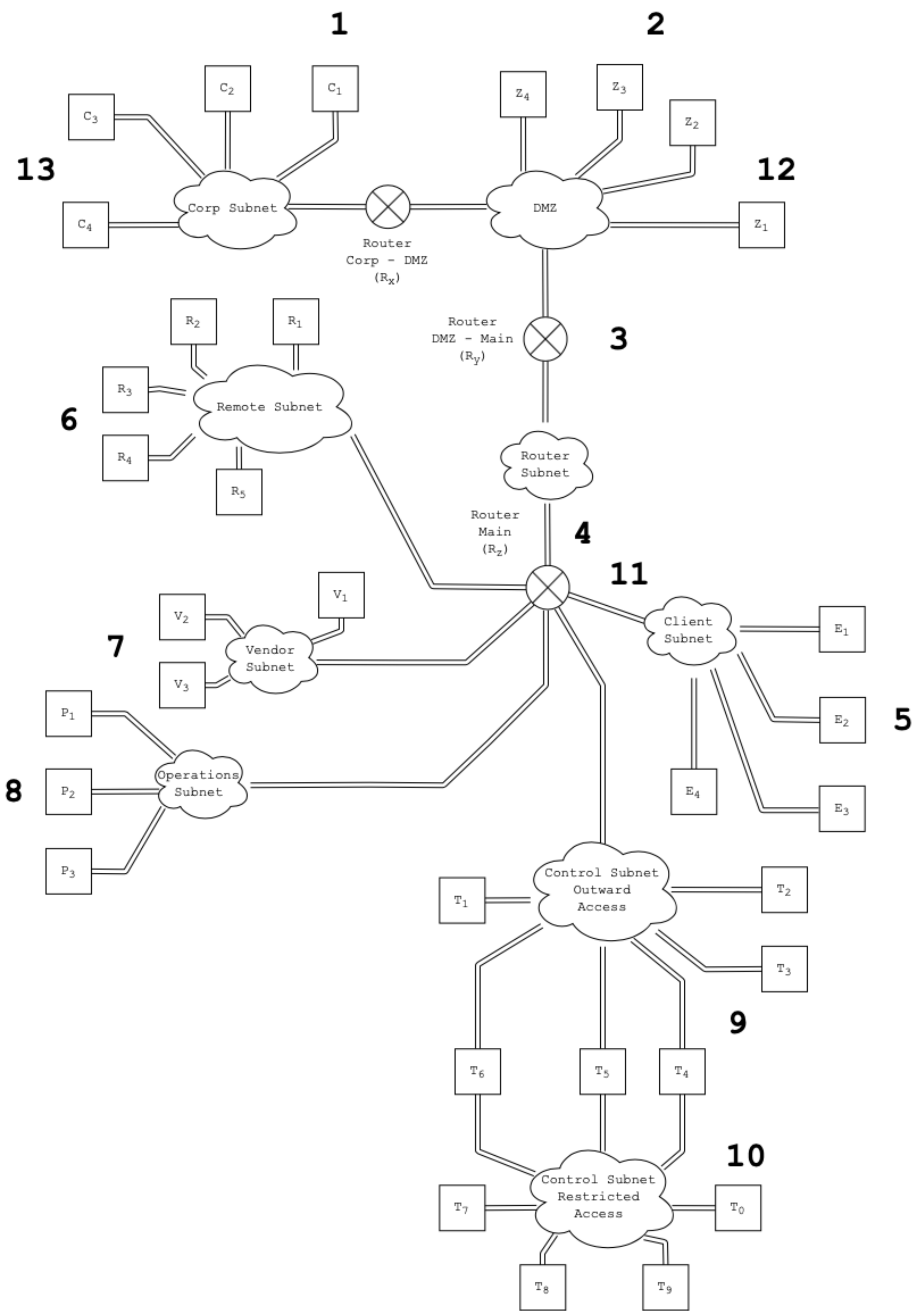
- Simulate an attack -> monitor the network's state during and after.
- Objective -> show our tool can be used for its intended purpose.
- We define the *Quality of Service* of the network as:

$$QoS(t) = \frac{pings(t)}{pings(0)};\ \ QoS(t) \in [0,\ 1]\ \forall\ t \in [0,\ \infty)$$

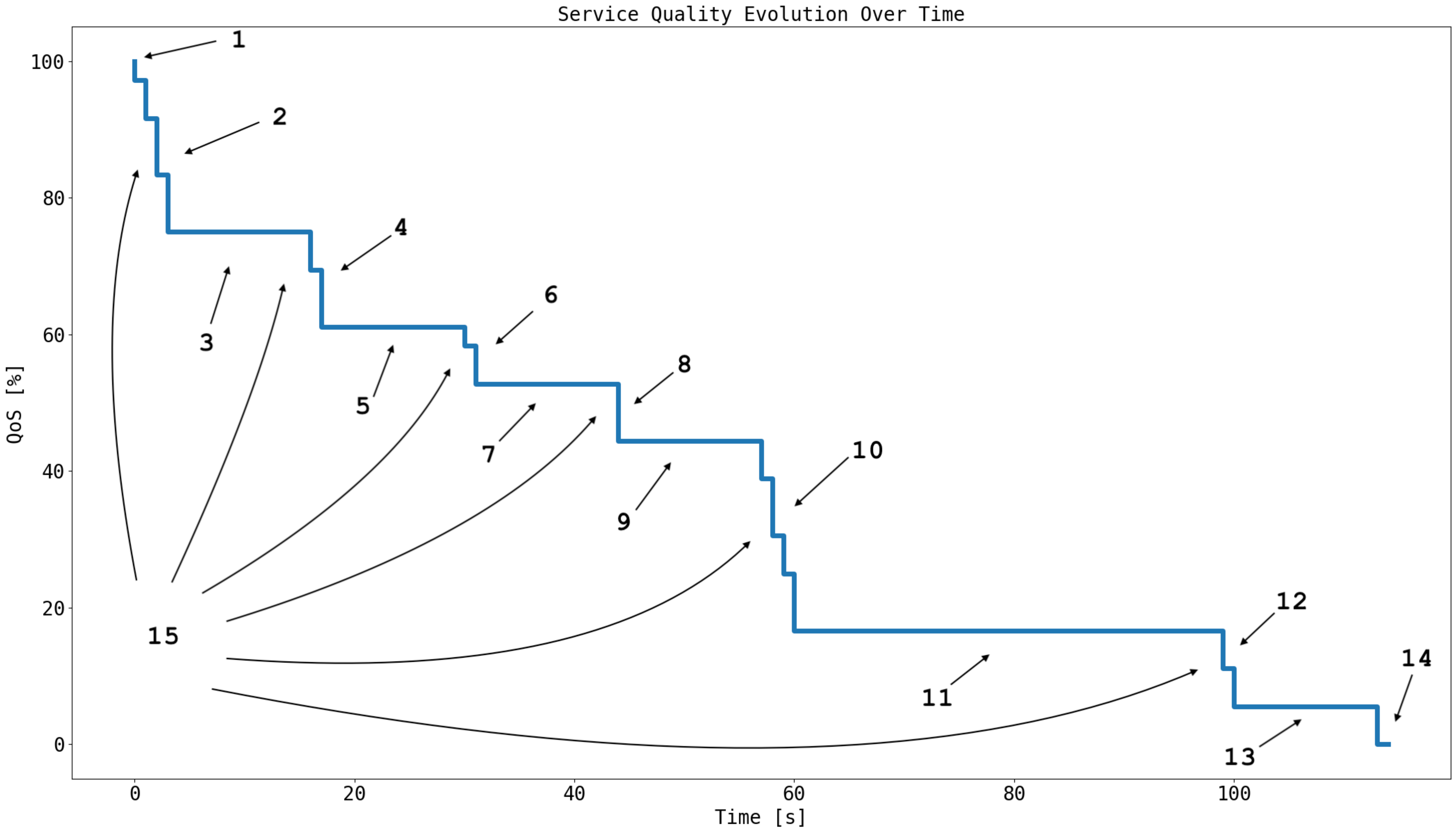
- Attack -> diminishes the *QoS*.
- Information of interest -> effect of topology changes on the *QoS* evolution.

Proof of Concept (II)

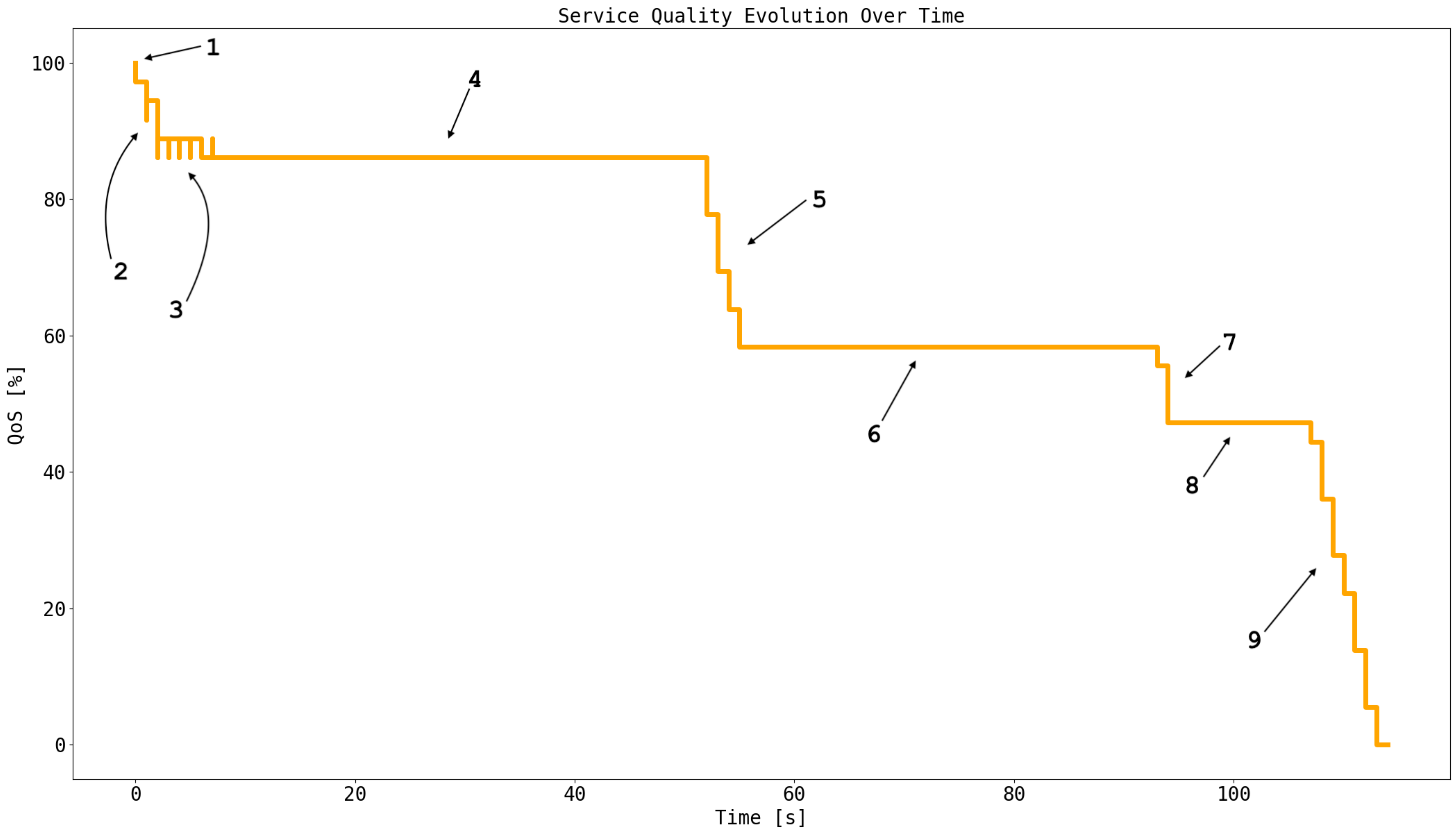
- Exploit -> weak `root` password (1234).
- Attack:
  - `Bash script`.
  - Logs into victims remotely through `ssh`.
  - Recursively copies itself to machines.
  - Base case -> no more subnets to attack.
  - Its progress can be known *a priori*.



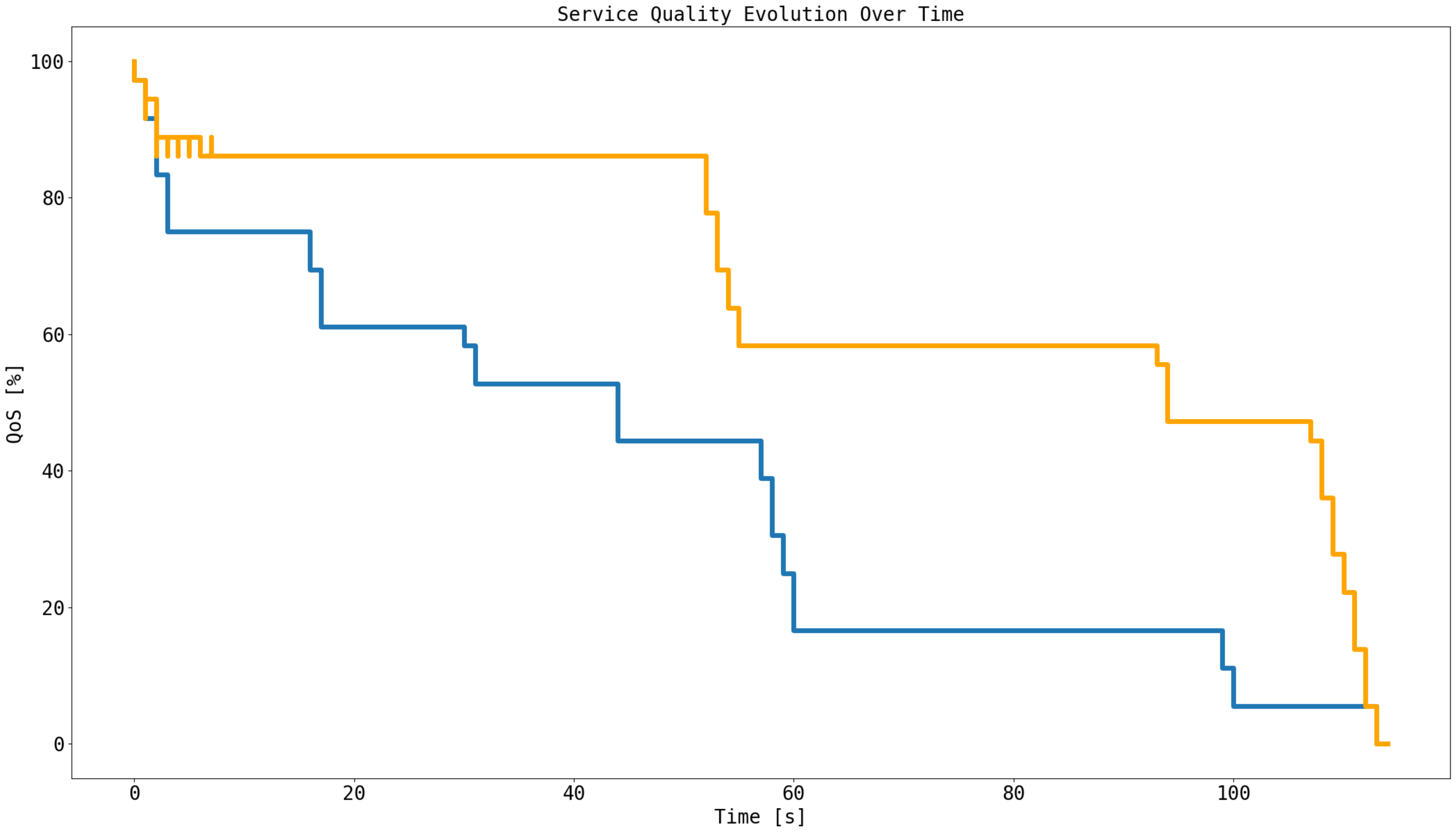
QoS Evolution on a Static Network



QoS Evolution on a Dynamic Network



Comparison of *QoS* Evolution



Closing Thoughts and Future Work

- Huge challenge.
- Learn about many different technologies and how to mesh them.
- Gained deeper understanding of how networking works “behind the scenes”.
- Learned how to manage our time and plan more efficiently.
- Possible future use:
  - Aide professors in the realm of networking.
  - Provide a powerful and lightweight alternative to VMs for network emulation.

Thank You for Your Time and Attention

Questions?

Source