Softwarized and Virtualized Mobile Networks Exercises

Vinci Nicolò

10 July 2022

Exercises proposed for the new network topology system. The Docker images are already provided and they are:

- *dev_host*: image used for a generic host.
- *dev_server*: image used for running a Docker in Docker virtualization.

1 Level easy

1.1 Question

Build a network topology yaml file with:

- Automatic MAC addresses enabled.
- Automatic ARP tables enabled.
- Local Controller.
- Switch sw0.
- Client c0 attached to sw0.
- Server s0 attached to sw0.

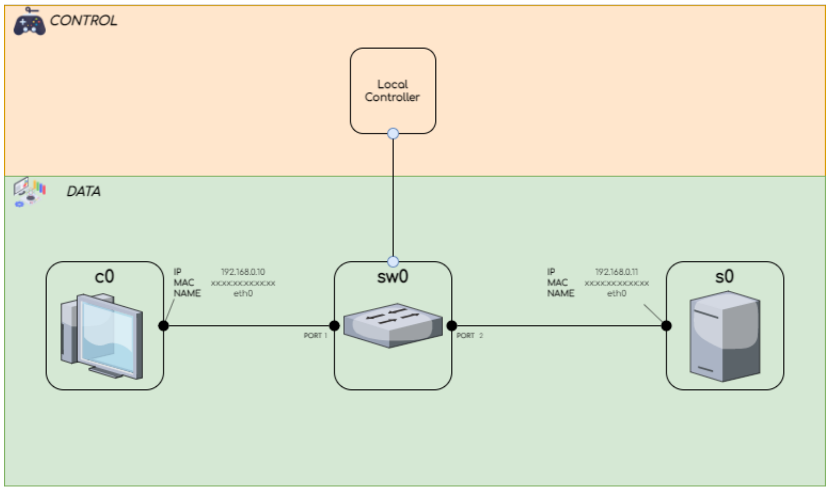


Figure 1: Network easy

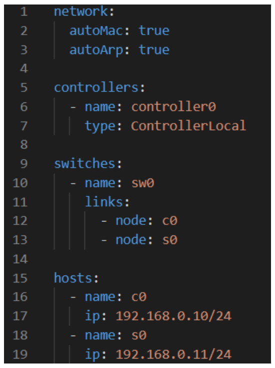
1.2 Solution

The solution is located at *src/exercises/easy* folder in the file *topology.yaml*. SSH in the Comnetsemu virtual machine.

```
$ cd SVMN/comnetsemu
$ vagrant ssh
$ cd comnetsmu/app/morphing_slices
```

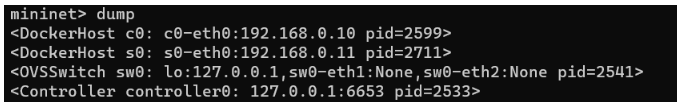
Run the network from solution.yaml file:

```
$ sudo python3 topology.py ——file exercises/easy/solution.yaml
```



```
network:
  autoMac: true
  autoArp: true

controllers:
  - name: controller0
    type: ControllerLocal

switches:
  - name: sw0
    links:
      - node: c0
      - node: s0

hosts:
  - name: c0
    ip: 192.168.0.10/24
  - name: s0
    ip: 192.168.0.11/24
```

Figure 2: Network topology solution



```
mininet> dump
<DockerHost c0: c0-eth0:192.168.0.10 pid=2599>
<DockerHost s0: s0-eth0:192.168.0.11 pid=2711>
<OVSSwitch sw0: lo:127.0.0.1,sw0-eth1:None,sw0-eth2:None pid=2541>
<Controller controller0: 127.0.0.1:6653 pid=2533>
```

Figure 3: Mininet dump

2 Level medium

2.1 Question

Build a network topology yaml file with:

- Automatic MAC addresses disabled.
- Automatic ARP tables disabled.
- Local Controller.
- Switch sw0.

- Client c0 with custom MAC address attached to sw0 defining link bandwidth and delay.
- Server s0 with custom MAC address attached to sw0 defining link bandwidth and delay.
- Switch sw1.
- Client c1 with custom MAC address attached to sw1 defining link bandwidth and delay.
- Server s1 with custom MAC addresses attached to sw0 and sw1 defining link bandwidths and delays.

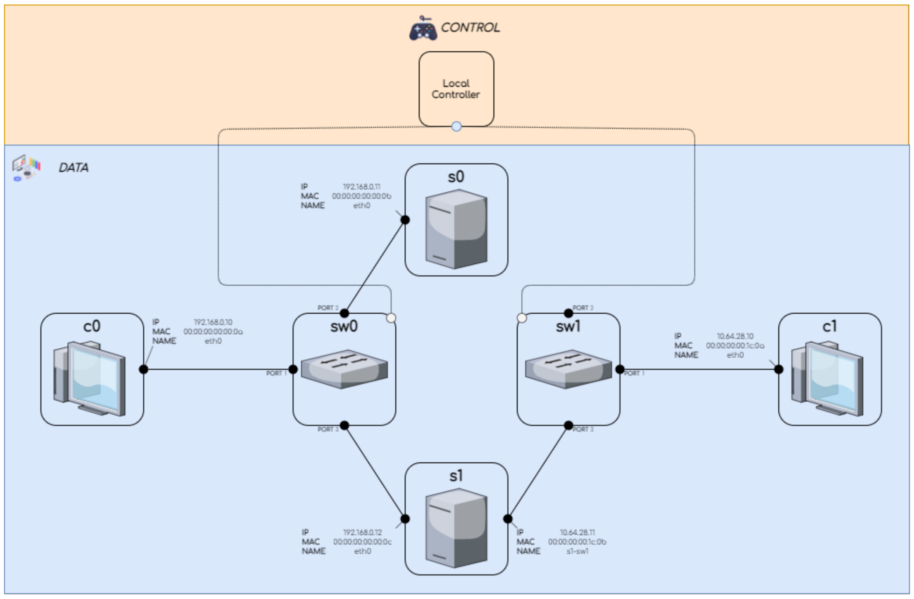


Figure 4: Network medium

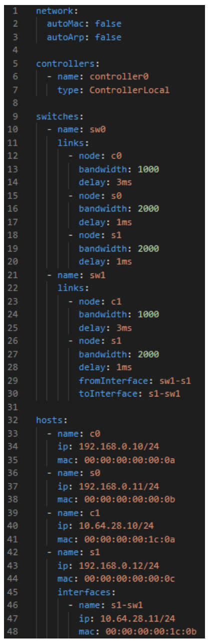
2.2 Solution

The solution is located at *src/exercises/medium* folder in the file *topology.yaml*. SSH in the Comnetsemu virtual machine.

```
$ cd SVMN/comnetsemu
$ vagrant ssh
$ cd comnetsmu/app/morphing_slices
```

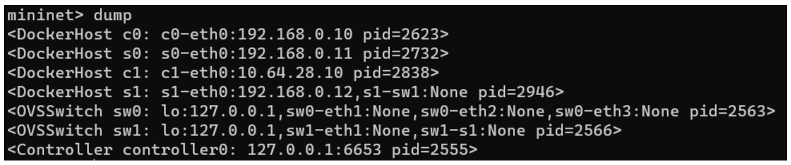
Run the network from solution.yaml file:

```
$ sudo python3 topology.py ——file exercises/medium/solution.yaml
```



```yaml
network:
  autoMac: false
  autoArp: false

controllers:
  - name: controller0
    type: ControllerLocal

switches:
  - name: sw0
    links:
      - node: c0
        bandwidth: 1000
        delay: 3ms
      - node: s0
        bandwidth: 2000
        delay: 1ms
      - node: s1
        bandwidth: 2000
        delay: 1ms
  - name: sw1
    links:
      - node: c1
        bandwidth: 1000
        delay: 3ms
      - node: s1
        bandwidth: 2000
        delay: 1ms
        fromInterface: sw1-s1
        toInterface: s1-sw1

hosts:
  - name: c0
    ip: 192.168.0.10/24
    mac: 00:00:00:00:00:0a
  - name: s0
    ip: 192.168.0.11/24
    mac: 00:00:00:00:00:0b
  - name: c1
    ip: 10.64.28.10/24
    mac: 00:00:00:00:1c:0a
  - name: s1
    ip: 192.168.0.12/24
    mac: 00:00:00:00:00:0c
    interfaces:
      - name: s1-sw1
        ip: 10.64.28.11/24
        mac: 00:00:00:00:1c:0b
```

Figure 5: Network topology solution



```
mininet> dump
<DockerHost c0: c0-eth0:192.168.0.10 pid=2623>
<DockerHost s0: s0-eth0:192.168.0.11 pid=2732>
<DockerHost c1: c1-eth0:10.64.28.10 pid=2838>
<DockerHost s1: s1-eth0:192.168.0.12,s1-sw1:None pid=2946>
<OVSSwitch sw0: lo:127.0.0.1,sw0-eth1:None,sw0-eth2:None,sw0-eth3:None pid=2563>
<OVSSwitch sw1: lo:127.0.0.1,sw1-eth1:None,sw1-s1:None pid=2566>
<Controller controller0: 127.0.0.1:6653 pid=2555>
```

Figure 6: Mininet dump

3 Level difficult

3.1 Question

Build a network topology yaml file with:

- Automatic MAC addresses disabled.
- Automatic ARP tables disabled.
- Remote Controller with IP 127.0.0.1 and default port 6653.
- Switch sw0.
- Client c0 with custom MAC address attached to sw0 defining link bandwidth and delay.

- Server s0 with custom MAC address attached to sw0 defining link bandwidth and delay.
- Switch sw1.
- Client c1 with custom MAC address attached to sw1 defining link bandwidth and delay.
- Server s1 with custom MAC addresses attached to sw0 and sw1 defining link bandwidth and delay.
- Server s1 and server s0 run a Docker container (*dev_server* image).

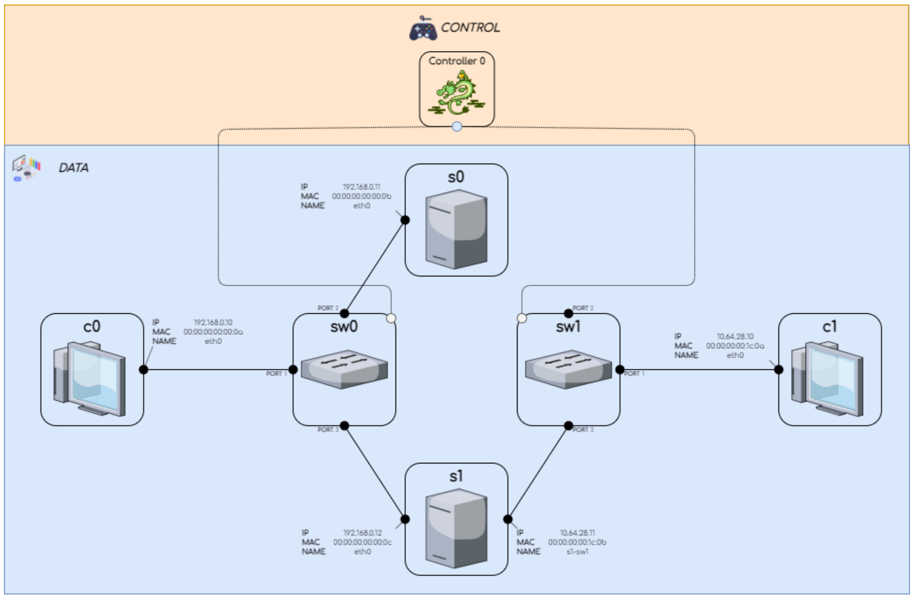


Figure 7: Network difficult

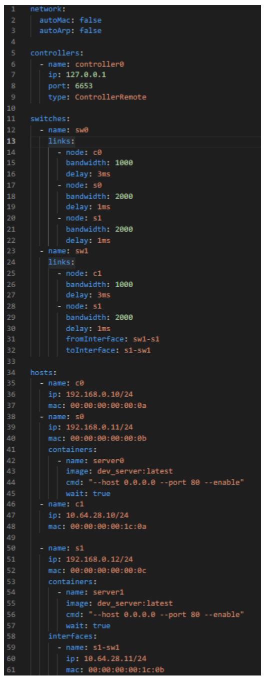
3.2 Solution

The solution is located at *src/exercises/difficult* folder in the file *topology.yaml*. SSH in the Comnetsemu virtual machine.

```
$ cd SVMN/comnetsemu
$ vagrant ssh
$ cd comnetsmu/app/morphing_slices
```

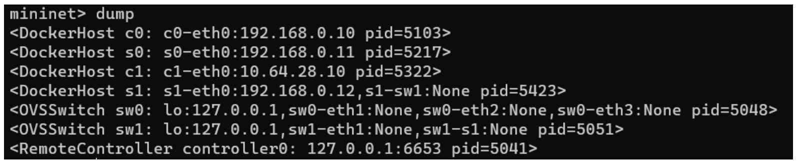
Run the network from solution.yaml file:

```
$ sudo python3 topology.py ——file exercises/difficult/solution.yaml
```



```yaml
network:
  autoMac: false
  autoArp: false

controllers:
  - name: controller0
    ip: 127.0.0.1
    port: 6653
    type: ControllerRemote

switches:
  - name: sw0
    links:
      - node: c0
        bandwidth: 1000
        delay: 3ms
      - node: s0
        bandwidth: 2000
        delay: 1ms
      - node: s1
        bandwidth: 2000
        delay: 1ms
  - name: sw1
    links:
      - node: c1
        bandwidth: 1000
        delay: 3ms
      - node: s1
        bandwidth: 2000
        delay: 1ms
        fromInterface: sw1-s1
        toInterface: s1-sw1

hosts:
  - name: c0
    ip: 192.168.0.10/24
    mac: 00:00:00:00:00:0a
  - name: s0
    ip: 192.168.0.11/24
    mac: 00:00:00:00:00:0b
    containers:
      - name: server0
        image: dev_server:latest
        cmd: "--host 0.0.0.0 --port 80 --enable"
        wait: true
  - name: c1
    ip: 10.64.28.10/24
    mac: 00:00:00:00:1c:0a

  - name: s1
    ip: 192.168.0.12/24
    mac: 00:00:00:00:00:0c
    containers:
      - name: server1
        image: dev_server:latest
        cmd: "--host 0.0.0.0 --port 80 --enable"
        wait: true
    interfaces:
      - name: s1-sw1
        ip: 10.64.28.11/24
        mac: 00:00:00:00:1c:0b
```

Figure 8: Network topology solution



```
mininet> dump
<DockerHost c0: c0-eth0:192.168.0.10 pid=5103>
<DockerHost s0: s0-eth0:192.168.0.11 pid=5217>
<DockerHost c1: c1-eth0:10.64.28.10 pid=5322>
<DockerHost s1: s1-eth0:192.168.0.12,s1-sw1:None pid=5423>
<OVSSwitch sw0: lo:127.0.0.1,sw0-eth1:None,sw0-eth2:None,sw0-eth3:None pid=5048>
<OVSSwitch sw1: lo:127.0.0.1,sw1-eth1:None,sw1-s1:None pid=5051>
<RemoteController controller0: 127.0.0.1:6653 pid=5041>
```

Figure 9: Mininet dump

4 Regular network

4.1 Question

Build a network topology yaml file with:

- Automatic MAC addresses enables.
- Automatic ARP tables disabled.
- Local Controller.
- Switch sw0.
- Client c0 attached to sw0.

- Client c1 attached to sw0.
- Switch sw1.
- Client c2 attached to sw1.
- Client c3 attached to sw1.
- Host r0 attached to sw0 and sw1. It will be the router.

Then, set up the network to make the two sub-networks communicate.

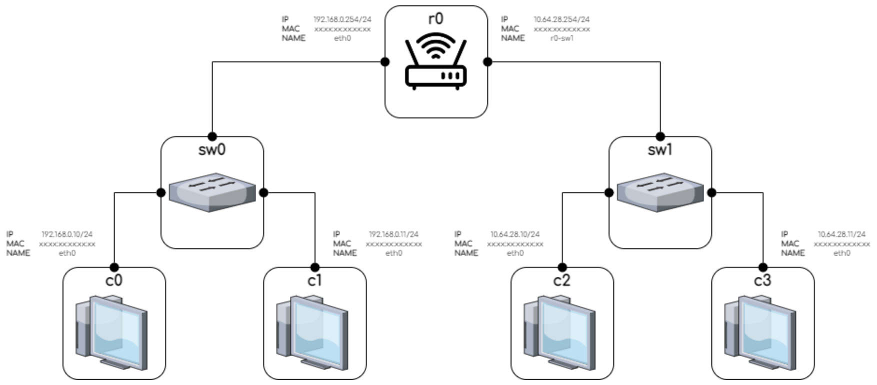


Figure 10: Regular network

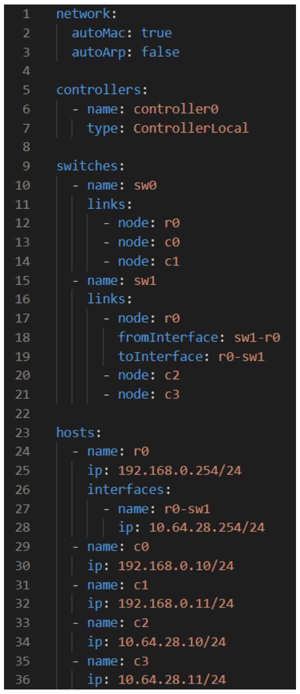
4.2 Solution

The solution is located at *src/exercises/regular* folder in the file *topology.yaml*. SSH in the Comnetsemu virtual machine.

```
$ cd SVMN/comnetsemu
$ vagrant ssh
$ cd comnetsmu/app/morphing_slices
```

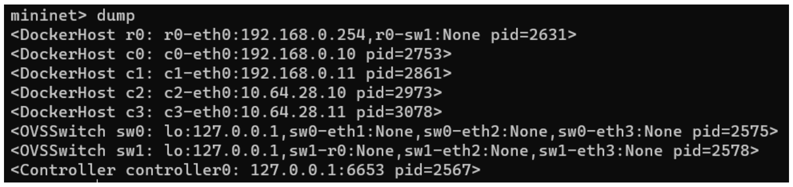
Run the network from solution.yaml file:

```
$ sudo python3 topology.py ——file exercises/regular/solution.yaml
```



```
network:
  autoMac: true
  autoArp: false

controllers:
  - name: controller0
    type: ControllerLocal

switches:
  - name: sw0
    links:
      - node: r0
      - node: c0
      - node: c1
  - name: sw1
    links:
      - node: r0
        fromInterface: sw1-r0
        toInterface: r0-sw1
      - node: c2
      - node: c3

hosts:
  - name: r0
    ip: 192.168.0.254/24
    interfaces:
      - name: r0-sw1
        ip: 10.64.28.254/24
  - name: c0
    ip: 192.168.0.10/24
  - name: c1
    ip: 192.168.0.11/24
  - name: c2
    ip: 10.64.28.10/24
  - name: c3
    ip: 10.64.28.11/24
```

Figure 11: Regular network topology solution



```
mininet> dump
<DockerHost r0: r0-eth0:192.168.0.254,r0-sw1:None pid=2631>
<DockerHost c0: c0-eth0:192.168.0.10 pid=2753>
<DockerHost c1: c1-eth0:192.168.0.11 pid=2861>
<DockerHost c2: c2-eth0:10.64.28.10 pid=2973>
<DockerHost c3: c3-eth0:10.64.28.11 pid=3078>
<OVSSwitch sw0: lo:127.0.0.1,sw0-eth1:None,sw0-eth2:None,sw0-eth3:None pid=2575>
<OVSSwitch sw1: lo:127.0.0.1,sw1-r0:None,sw1-eth2:None,sw1-eth3:None pid=2578>
<Controller controller0: 127.0.0.1:6653 pid=2567>
```

Figure 12: Mininet dump

Do the follow steps in the mininet console:

1. Enable forwarding for the router r0.

```
$ r0 sysctl net.ipv4.ip_forward=1
```

2. Add routing rule to c0 and c1.

```
$ c0 ip route add 10.64.28.0/24 via 192.168.0.254
$ c1 ip route add 10.64.28.0/24 via 192.168.0.254
```

3. Add routing rule to c2 and c3.

```
$ c2 ip route add 192.168.0.0/24 via 10.64.28.254
$ c3 ip route add 192.168.0.0/24 via 10.64.28.254
```

Note:
Interfaces added manually like *r0-sw1* for the router r0 are not recognized by mininet in the pingall command. By the way, if the ping is launched directly, it works.

```
$ c3 ping —c 2 192.168.0.10
```