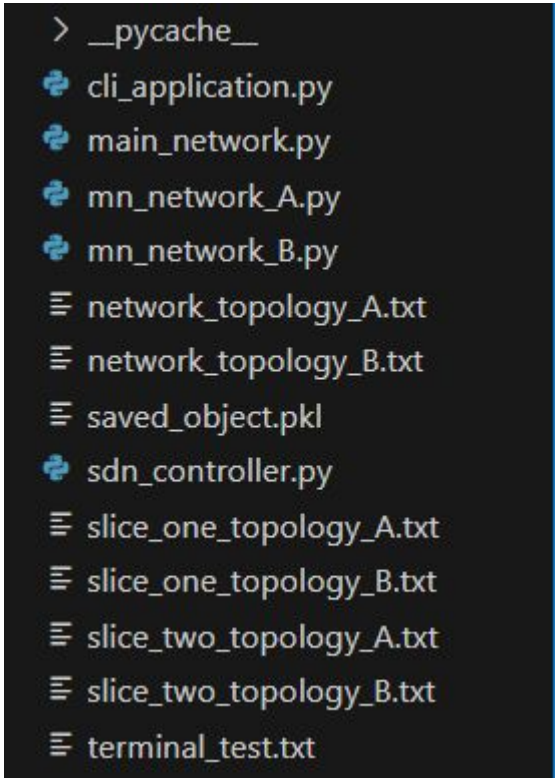
On Demand SDN Slices in ComNetsEmu

Group member: Luca Caccavale (Mat. 218724) Corso di laurea triennale in Ingegneria dell'informatica, delle comunicazioni ed elettronica. (Percorso informatica)

Goal of the project

- GOAL: to implement a network slicing approach to enable dynamic activation/de-activation of network slices via CLI/GUI commands
- One SDN controller (e.g. RYU) is enough
- On demand means that he user can activate and de-activate different slices
- You decide how slices are described (templates are possible), however you should allow to identify flows, topology and % of link capacity for each slice

Folder structure



1. main_network.py
   a. python file where the data structures such as classes and methods are defined to implement the slice activation and deactivation operations.
2. cli_application.py
   a. python file where the command line interface is defined. The commands are associated with the functions present in main_network.
3. mn_network_A.py, mn_network_B.py
   a. python file to generate the topology
4. snd_controller.py
   a. simple SDN controller
5. saved_object.pkl
   a. memory of the CLI application
6. network_topology_A.txt, slice_one_topology_A.txt, slice_two_topology_A.txt
   a. text files that contain the description of the network topology or slices (set of hosts and routers and the connections between them)
7. terminal_test.txt
   a. file text where the command executed by the CLI application in the shell are writed in order to understand how the CLI application works

how it all works? Example:

Command of the CLI application:

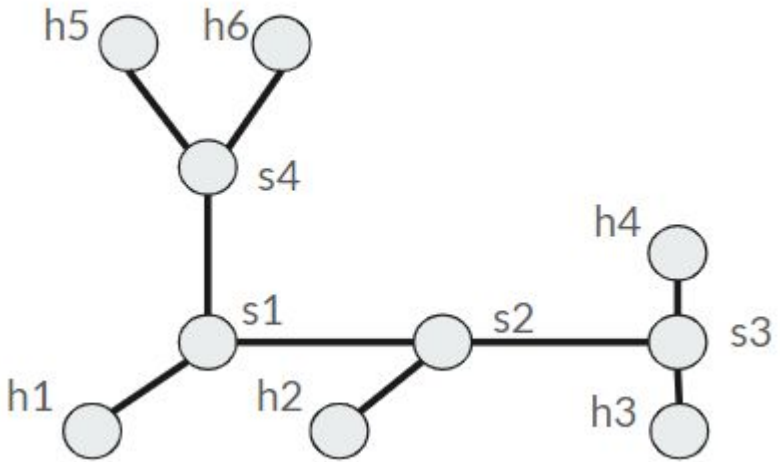
- python3 cli_application.py add -t .\network_topology.txt
- python3 cli_application.py add -s .\slice_one_topology.txt
- python3 cli_application.py activate -s .\slice_one_topology.txt
- python3 cli_application.py deactivate -s .\slice_one_topology.txt
- python3 cli_application.py show
- python3 cli_application.py delete -t .\network_topology.txt
- python3 cli_application.py delete -s .\slice_one_topology.txt

Possible operations:

- add a new topology
- add a new slice
- activate/deactivate a slice
- delete a topology (this action will also delete all the slices)
- delete a slice
- show the network topology and slices added and its state (activated or deactivated)

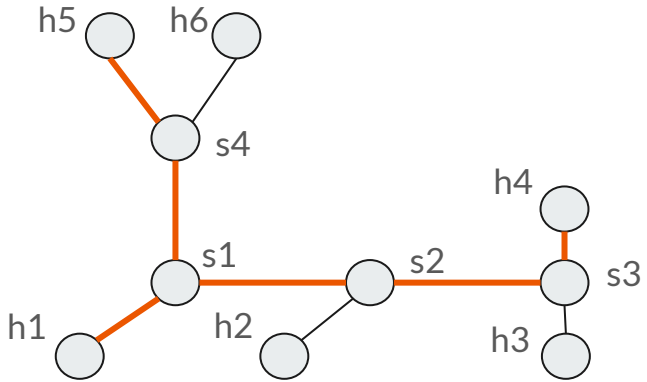
Description of the topology

```
HostToRouter
(h1,s1)
(h2,s2)
(h3,s3)
(h4,s3)
(h5,s4)
(h6,s4)
RouterToRouter
(s1-eth1,s4-eth1)
(s1-eth2,s2-eth1)
(s3-eth1,s2-eth2)
```

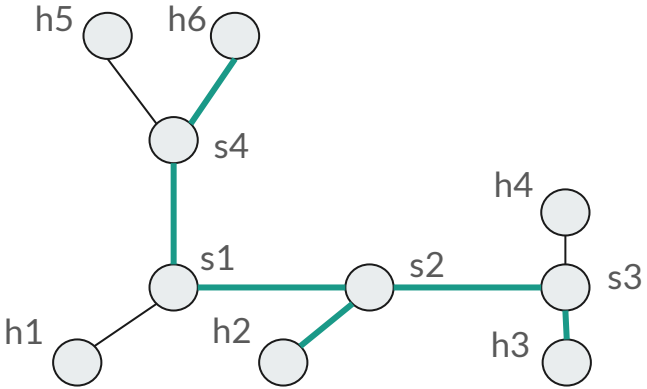


Description of the slices

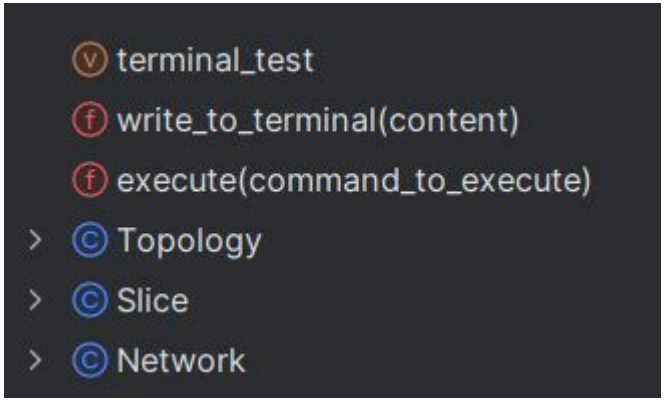
```
HostToRouter
(h1,s1)
(h4,s3)
(h5,s4)
RouterToRouter
(s1-eth1-10,s4-eth1-10)
(s1-eth2-50,s2-eth1-50)
(s3-eth1-50,s2-eth2-50)
```




```
HostToRouter
(h2,s2)
(h3,s3)
(h6,s4)
RouterToRouter
(s1-eth1-50,s4-eth1-50)
(s1-eth2-50,s2-eth1-50)
(s3-eth1-10,s2-eth2-10)
```



Structure of the python file main_network.py



- **write_to_terminal(content)**: function used to write in file text specified in the variable named terminal_test;
- **execute()**: function used fo the execution of the command;
- **Topology**, **Slice**, **Network**: classes that contain the data structure of the network and slices a run-time, and methods used to generate the command to add new flows in the switches.

class Topology


```
HostToRouter
(h1,s1)
(h4,s3)
(h5,s4)
RouterToRouter
(s1-eth1-10,s4-eth1-10)
(s1-eth2-50,s2-eth1-50)
(s3-eth1-50,s2-eth2-50)
```

parse_file(self, file_name)

call back:
parse_host_router()
parse_router_router()

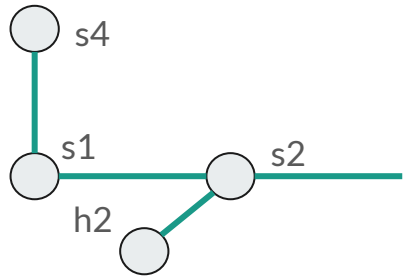
```
host_router_links
[ ['h1','r1'] , ['h2','r2'] , ... ]
router_router_links
[ [ ['s1','eth1'] , ['s2','eth1'] ] , ... ]
```

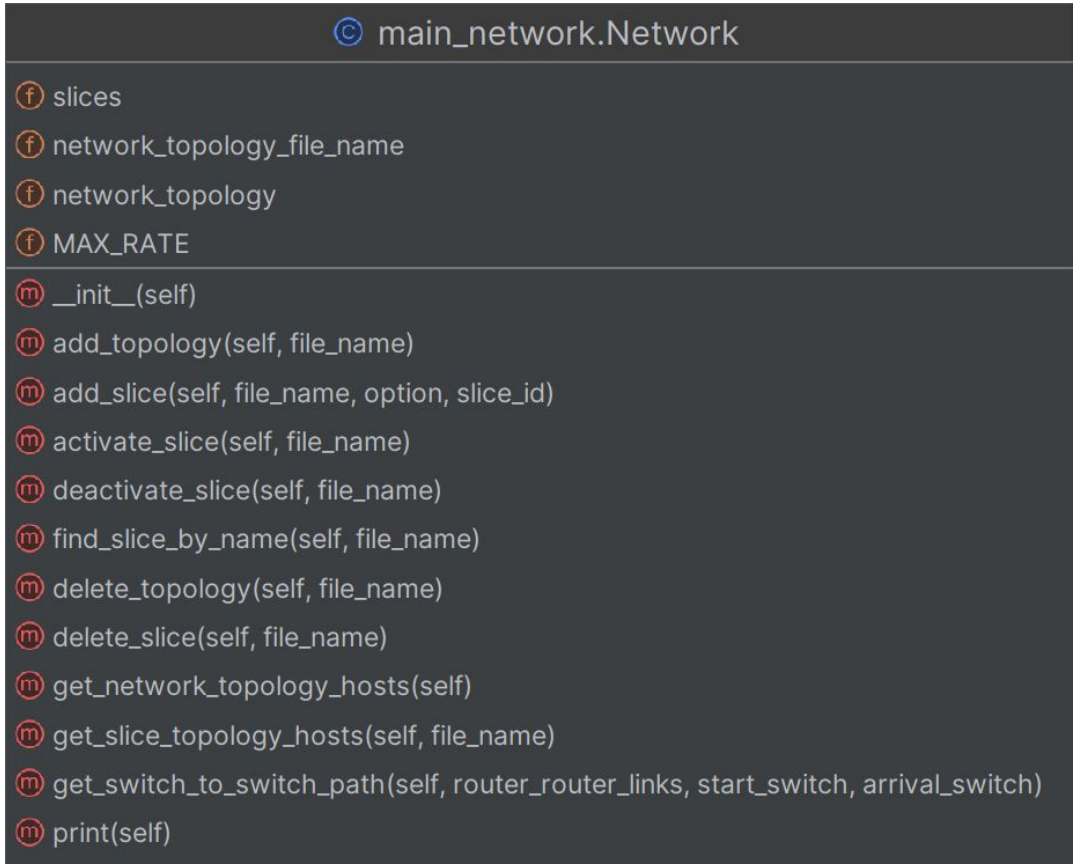
main_network.Slice

- slice_name
- slice_topology
- slice_state
- slice_id
- __init__(self)
- __init__(self, file_name, slice_id)
- get_hosts(self)
- activate_queue(self,MAX_RATE)
- deactivate_queue(self)
- get_slice_topology(self)
- get_host_router_links(self)
- get_router_router_links(self)
- print(self)

class Slice

- **activate_queue(self, MAX_RATE)**: function used to create all the queues that belongs to a slice. Example: `['s1','eth1'] , ['s2','eth1']` if s1 is linked to s2 and viceversa s2 with s1, with respectively port eth1 and port eth2, I will generate two queue.
- The queue id will be formed by two parts:
  - the slice id. (e.g. 123)
  - the number of the port (e.g. 1)
- S1 in the picture below will have two queue and their id will be
  - 1231 (for packet toward s2 if the port is eth1)
  - 1232 (for packet toward s4 if the port is eth2)





class Network

1) **add_topology()**: initial state of the network. Network stopped, no communication between hosts.
2) **activate_slice()**:
   a) I re-enable communication between hosts,
   b) create queues,
   c) assign queues to nw_src nw_dst flows,
   d) isolate the slice (slice hosts cannot communicate outside)
3) **get_switch_to_switch_path()**: analyze the data structure router_router_links and given a start_switch and an end_switch return the path between the two switches

activating a slice: what do I expect?

A. Considering an initial state where no slice is active and therefore the network is completely stopped (no one can communicate with anyone).
B. Then I have to allow communication to be possible between the specified hosts.
    a. For each Ri router belonging to Sk:
        i. I create a queue in Ri for each port
    b. For each Ri belonging to Sk & for each Hj connected to Ri:
        i. I add a rule in all the router through which the packet will pass to associate the packets ranging from Hj to Hf belonging to Sk to the newly created queues (j different from f)
        ii. I add a rule in Ri to drop packets that go from Hj to Hf not belonging to Sk (j different from f)

Example of generated commands: initial state of the network

"Considering an initial state where no slice is active and therefore the network is completely stopped (no one can communicate with anyone)"

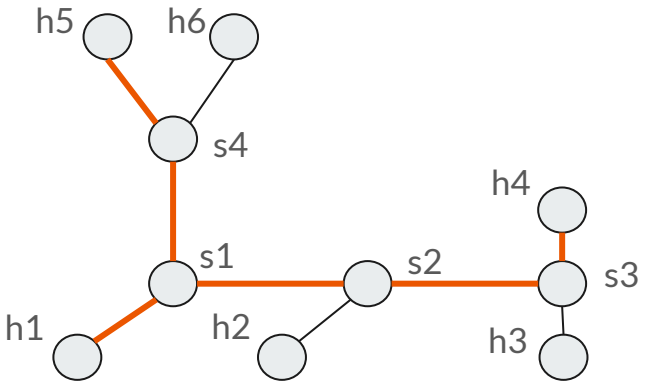
```
sudo ovs-ofctl add-flow s1 ip,priority=65500,nw_src=10.0.0.1,nw_dst=0.0.0.0/0,idle_timeout=0,actions=drop
sudo ovs-ofctl add-flow s1 ip,priority=65500,nw_src=10.0.0.1,nw_dst=0.0.0.0/0,idle_timeout=0,actions=drop
sudo ovs-ofctl add-flow s2 ip,priority=65500,nw_src=10.0.0.2,nw_dst=0.0.0.0/0,idle_timeout=0,actions=drop
sudo ovs-ofctl add-flow s3 ip,priority=65500,nw_src=10.0.0.3,nw_dst=0.0.0.0/0,idle_timeout=0,actions=drop
sudo ovs-ofctl add-flow s3 ip,priority=65500,nw_src=10.0.0.4,nw_dst=0.0.0.0/0,idle_timeout=0,actions=drop
sudo ovs-ofctl add-flow s4 ip,priority=65500,nw_src=10.0.0.5,nw_dst=0.0.0.0/0,idle_timeout=0,actions=drop
sudo ovs-ofctl add-flow s4 ip,priority=65500,nw_src=10.0.0.6,nw_dst=0.0.0.0/0,idle_timeout=0,actions=drop
```

Example of generated commands: activation of a slice

"Then I have to allow communication to be possible between the specified hosts".

I remove the rules added before:

```
sudo ovs-ofctl del-flows s1 ip,nw_src=10.0.0.1
sudo ovs-ofctl del-flows s3 ip,nw_src=10.0.0.4
sudo ovs-ofctl del-flows s4 ip,nw_src=10.0.0.5
```



activation of a slice: creation of the queue

a. For each Ri router belonging to Sk:
   i. I create a queue in Ri for each port

sudo ovs-vsctl set port **s1-eth1** qos=@newqos -- --id=@newqos create QoS is_a_slice=linux-htb other-config:max-rate=10000000 queues:**1231**=@**1**q -- --id=@**1**q create queue other-config:min-rate=1000000 other-config:**max-rate=7000000**

sudo ovs-vsctl set port **s4-eth1** qos=@newqos -- --id=@newqos create QoS is_a_slice=linux-htb other-config:max-rate=10000000 queues:**1231**=@**1**q -- --id=@**1**q create queue other-config:min-rate=1000000 other-config:**max-rate=7000000**

sudo ovs-vsctl set port **s1-eth2** qos=@newqos -- --id=@newqos create QoS is_a_slice=linux-htb other-config:max-rate=10000000 queues:**1232**=@**2**q -- --id=@**2**q create queue other-config:min-rate=1000000 other-config:**max-rate=5000000**

activation of a slice: queue assignment

N.B. given a sending host and a receiving host I have to assign the packets to **all the queues that these packets will have to pass through**. (Hence the need for a route search function)

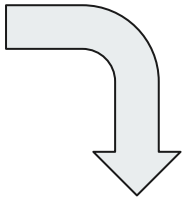
```
sudo ovs-ofctl add-flow s1
ip,priority=65500,nw_src=10.0.0.1,nw_dst=10.0.0.4,idle_timeout=0,actions=set_queue:1232,normal

sudo ovs-ofctl add-flow s2
ip,priority=65500,nw_src=10.0.0.1,nw_dst=10.0.0.4,idle_timeout=0,actions=set_queue:1232,normal

sudo ovs-ofctl add-flow s1
ip,priority=65500,nw_src=10.0.0.1,nw_dst=10.0.0.5,idle_timeout=0,actions=set_queue:1231,normal
```

activation of a slice: isolation of the slice

1. For each Ri router belonging to Sk:
   a. add a rule in Ri to drop packets that go from Hj to Hf not belonging to Sk (j different from f)



```
sudo ovs-ofctl add-flow s1 ip,priority=65500,nw_src=10.0.0.1,nw_dst=10.0.0.2,idle_timeout=0,actions=drop
sudo ovs-ofctl add-flow s1 ip,priority=65500,nw_src=10.0.0.1,nw_dst=10.0.0.3,idle_timeout=0,actions=drop
sudo ovs-ofctl add-flow s1 ip,priority=65500,nw_src=10.0.0.1,nw_dst=10.0.0.6,idle_timeout=0,actions=drop
```

deactivating a slice: what do I expect?

A. I need to remove the set of rules added following the activation of a slice and restore the network to the state before the slice was activated

More on: `def get_switch_to_switch_path(self, router_router_links, start_switch, arrival_switch)`

```
# This function return a structure like this: [('s4', 'eth7'), ('s1', 'eth66'), ('s3', 'eth3')]
# If the start_switch is s4 and the arrival_switch is s2, the structure show where the
# packet should be forwarded: from s4 on port eth7 to reach s1, from s1 on port eth66 to reach s3
# and finally from s3 on port eth7 to reach s2
```

`router_router_links`

(s1-eth5,**s4**-eth7)
(s1-eth66,s3-eth2)
(s3-eth3,**s2**-eth4)

End

Thanks for the attention