UNIVERSITÀ
DI TRENTO

# NETWORK SLICING

SOFWARIZED AND VIRTUALIZED MOBILE NETWORKS - NETWORKING MOD 2
PROF. FABRIZIO GRANELLI

ELIF RUVEYHA OZMEN[238946]
ALESSANDRO MIGOTTO[238774]

A.A. 2023/2024

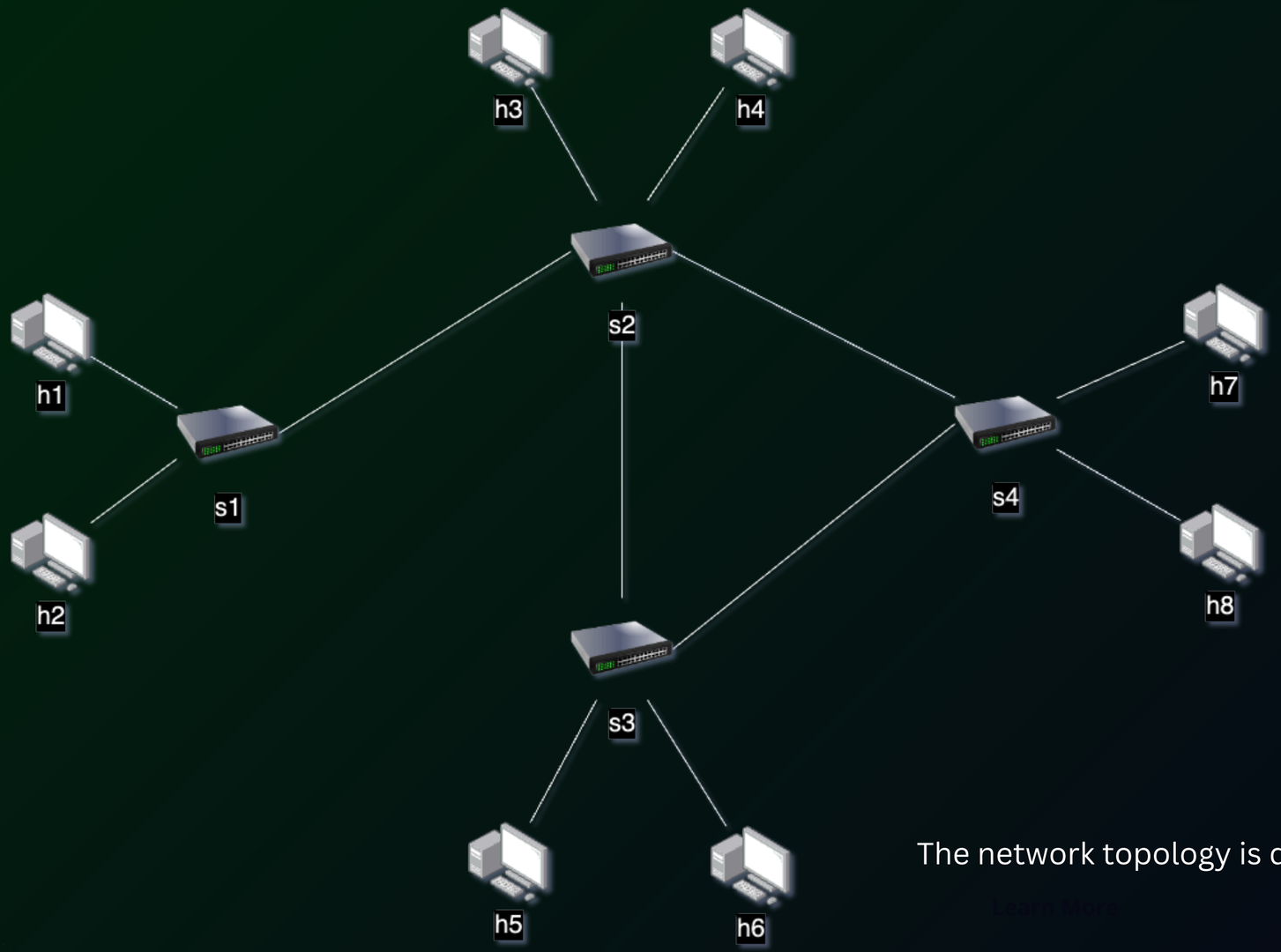
# PROJECT GOAL

The primary objective of this project is to facilitate the dynamic management of network slices within a Software-Defined Networking (SDN) environment.

This entails enabling users to activate and deactivate different slices on-demand, providing flexibility and adaptability to changing network requirements.

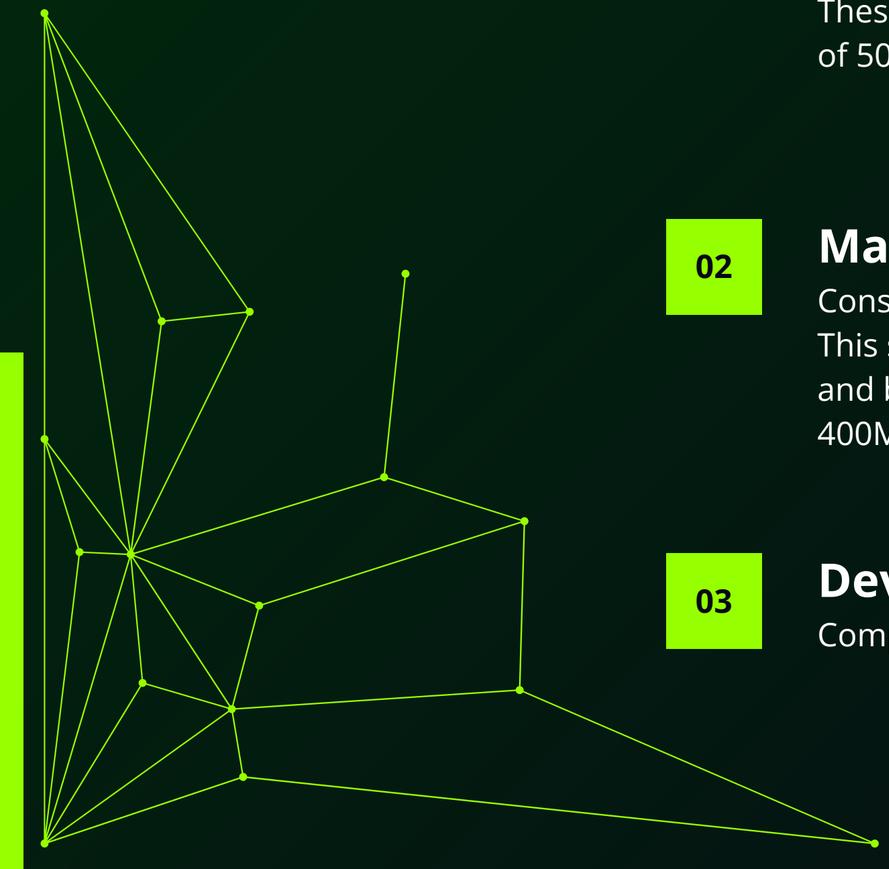
The user is provided with a Flask User Interface (UI) with which they can easily manage activation and deactivation of network slices through buttons, and graphically visualize them.

# NETWORK TOPOLOGY




The network topology is defined in /network.py

# SLICES



01

## Production slice:

Includes hosts h1, h2, h3, h4, and h8.
These hosts are part of the production environment and have bandwidth allocation of 500Mbps.

02

## Management slice:

Consists of hosts h5 and h7.
This slice is dedicated to management tasks and employs different routing paths and bandwidth allocation for UDP and TCP traffic. 300Mbps for TCP traffic and 400Mbps for UDP traffic.

03

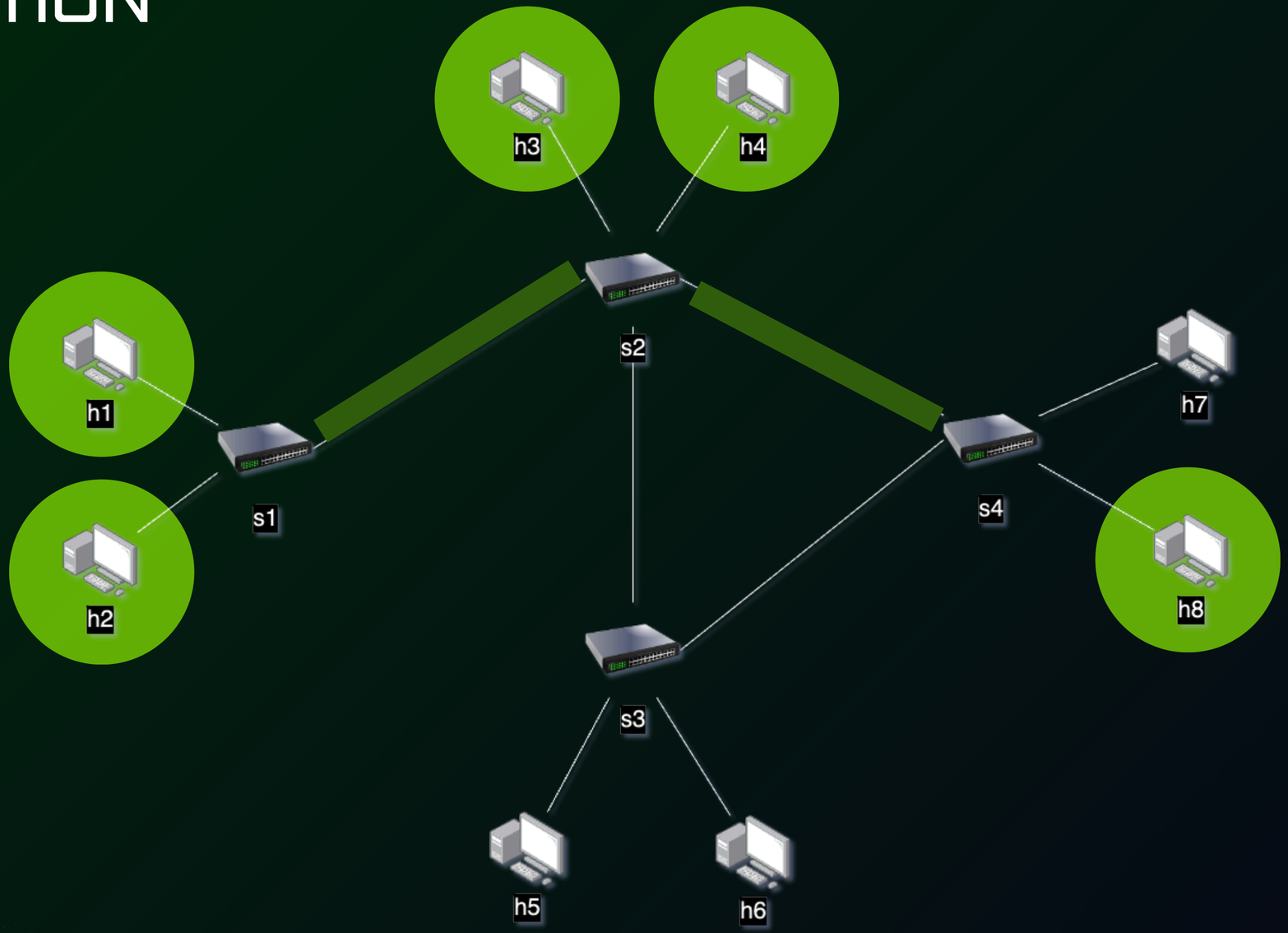
## Development slice:

Comprises hosts h1 and h6, designated for development purposes.

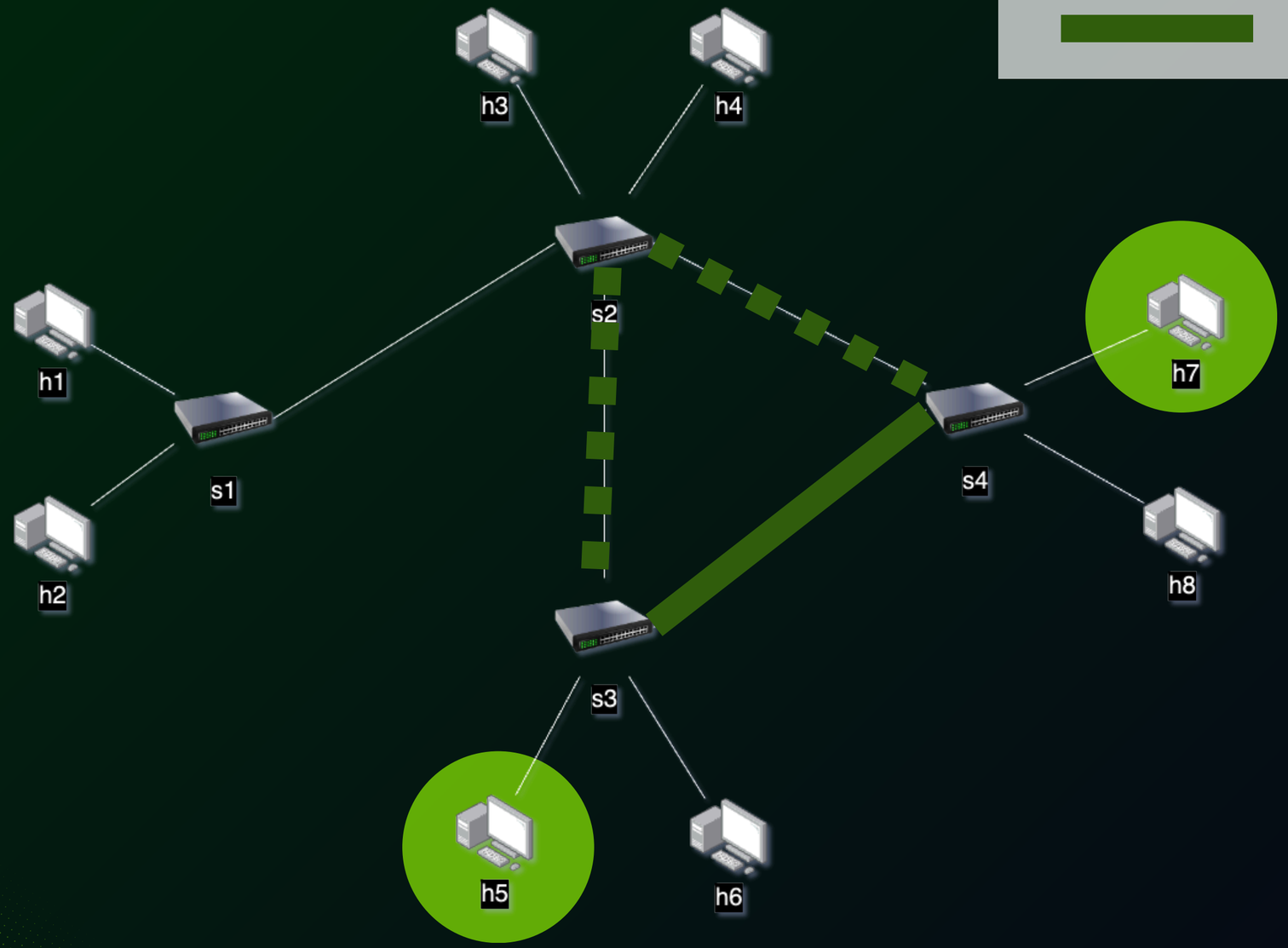
UNIVERSITÀ
DI TRENTO

# PRODUCTION SLICE

500 MBPS

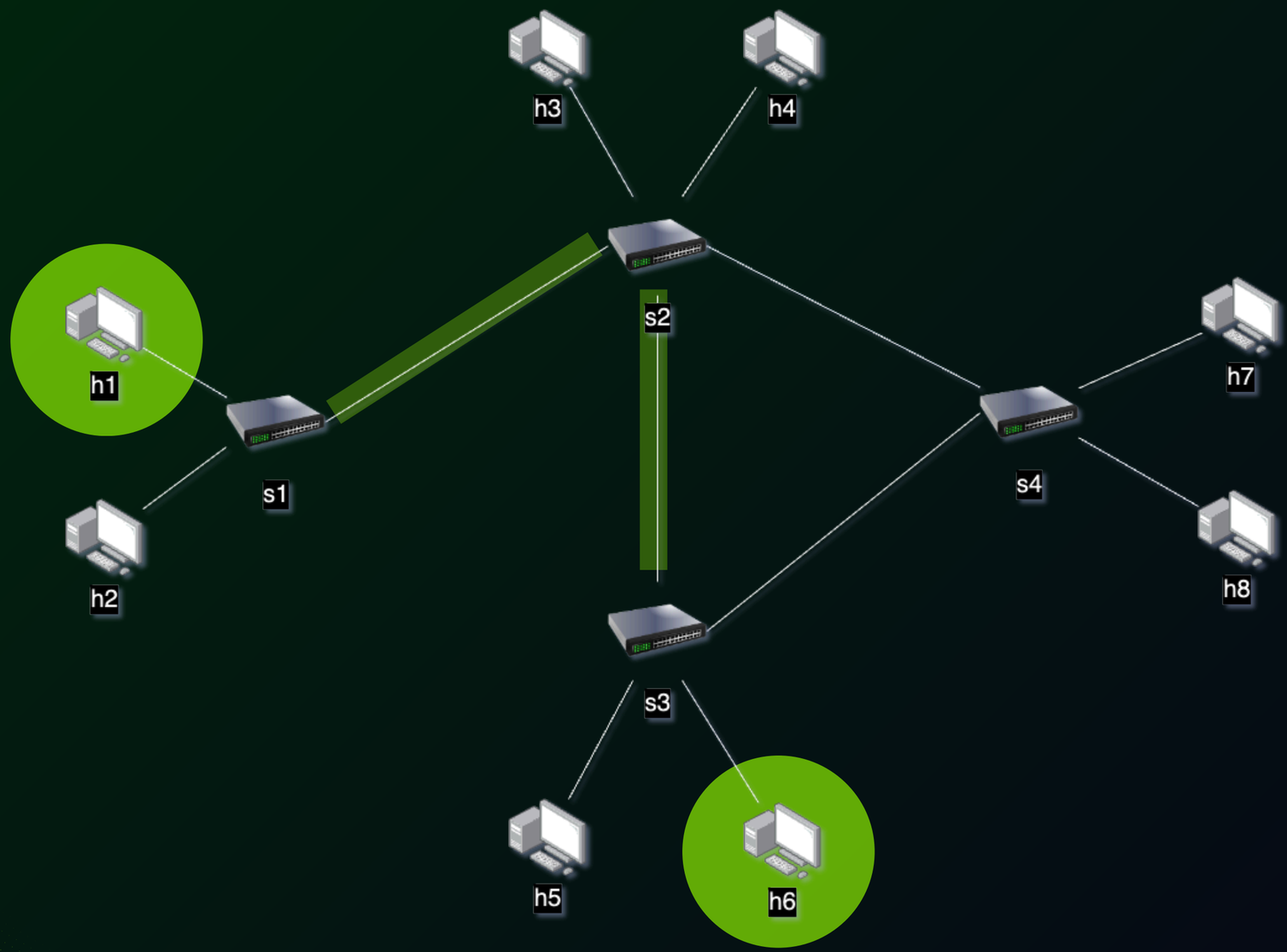


# MANAGEMENT SLICE

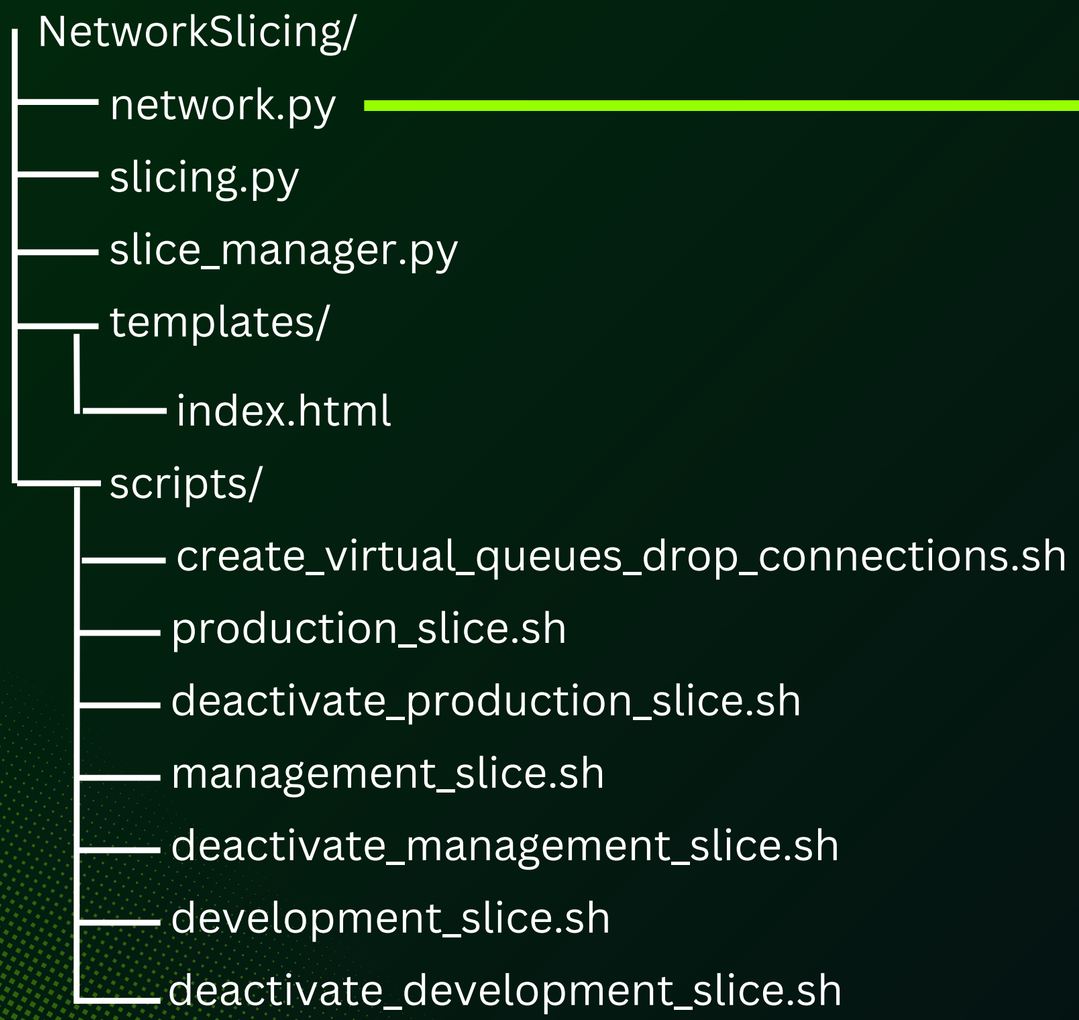


# DEVELOPMENT SLICE

200 MBPS



# HOW IT WORKS




**network.py**

Defines the network topology
Connects to the controller
Runs the create_virtual_queues_drop_connections.sh script

**create_virtual_queues_drop_connections.sh**

Creates all the virtual queues
Drops all the connections between hosts as there is no connection initially

UNIVERSITÀ DI TRENTO

# HOW IT WORKS

- NetworkSlicing/
  - network.py
  - slicing.py
  - slice_manager.py
  - templates/
    - index.html
  - scripts/
    - create_virtual_queues_drop_connections.sh
    - production_slice.sh
    - deactivate_production_slice.sh
    - management_slice.sh
    - deactivate_management_slice.sh
    - development_slice.sh
    - deactivate_development_slice.sh

**slice_manager.py**

A Flask application that allows users to activate or deactivate services through a web interface, executing the corresponding activation and deactivation scripts.

**index.html**

Defines the web interface featuring topology visualization and SVG diagrams for dynamic updates.
The activate and deactivate buttons are linked to the corresponding routes defined in slice_manager.py.

**<slice_name>_slice.sh**

Adds flow rules to the switches associated with the hosts of each slice.

**deactivate_<slice_name>_slice.sh**

Adds flow rules to the switches associated with the hosts of each slice, setting the action to "drop" to disconnect the hosts.

# HOW IT WORKS

- NetworkSlicing/
  - network.py
  - slicing.py
  - slice_manager.py
  - templates/
    - index.html
  - scripts/
    - create_virtual_queues_drop_connections.sh
    - production_slice.sh
    - deactivate_production_slice.sh
    - management_slice.sh
    - deactivate_management_slice.sh
    - development_slice.sh
    - deactivate_development_slice.sh

**slicing.py**

The Ryu controller application that:

- Defines the slices.
- Maps MAC addresses to switch ports in *mac_to_port* for each switch.

**packet_in_handler event handling**

Extracts packet information and handles the packet forwarding with the rules. Also it does slice specific handling for UDP and TCP protocols in management slice.

# WALKTHROUGH

- Run the ComNetsEmu VM. Follow the instructions for setup.

  *https://www.granelli-lab.org/researches/relevant-projects/comnetsemu-labs*

- Clone this repository to your ComNetsEmu virtual machine

  *$ git clone https://github.com/OrdnasselaOttogim/networkSlicing.git*

- Navigate to the cloned repository directory.

  *$ cd /networkSlicing*

- Ensure that Flask is installed. You can install Flask using pip install if it is not already installed:

  *$ pip install flask*

- Run the Ryu manager along with the interface:

  *$ ryu-manager slicing.py*

# WALKTHROUGH

- Open another terminal window (in the same ComNetsEmu VM instance) and navigate to the same directory

- Start Mininet and build the network topology using the provided network configuration. This command will set up the network according to the specified topology and slices:

  *$ sudo python3 network.py*

- You can now use the interface to activate or deactivate network slices as needed.

- Additionally, you can perform tests using the Mininet terminal to verify the behaviour of the network under different conditions (e.g. with *ping* or *iperf* commands, see subsequent Test Section).

# TEST RESULTS

Initially there is no connection between hosts, because no slice has been activated yet. By default, the flow tables of each switch are instructed to drop the packets.

Here is an example of a flow rule where switch 1 drops packets coming from h1 and destined for h2:
sudo ovs-ofctl add-flow s1 ip,priority=500,nw_src=10.0.0.1,nw_dst=10.0.0.2,idle_timeout=0,actions=drop

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X X X X
h2 -> X X X X X X X
h3 -> X X X X X X X
h4 -> X X X X X X X
h5 -> X X X X X X X
h6 -> X X X X X X X
h7 -> X X X X X X X
h8 -> X X X X X X X
*** Results: 100% dropped (0/56 received)
```

UNIVERSITÀ
DI TRENTO

# TEST RESULTS

When production slice is activated

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 X X X h8
h2 -> h1 h3 h4 X X X h8
h3 -> h1 h2 h4 X X X h8
h4 -> h1 h2 h3 X X X h8
h5 -> X X X X X X X
h6 -> X X X X X X X
h7 -> X X X X X X X
h8 -> h1 h2 h3 h4 X X X
*** Results: 64% dropped (20/56 received)
```

As displayed, h1 is able to ping h2, h3, h4, h8, and vice versa, respectively.
The other hosts remain unreachable as before.

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['478 Mbits/sec', '481 Mbits/sec']
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['478 Mbits/sec', '480 Mbits/sec']
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['478 Mbits/sec', '481 Mbits/sec']
mininet> iperf h1 h8
*** Iperf: testing TCP bandwidth between h1 and h8
*** Results: ['478 Mbits/sec', '480 Mbits/sec']
mininet> iperf h2 h1
*** Iperf: testing TCP bandwidth between h2 and h1
*** Results: ['478 Mbits/sec', '481 Mbits/sec']
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['478 Mbits/sec', '480 Mbits/sec']
mininet> iperf h2 h4
*** Iperf: testing TCP bandwidth between h2 and h4
*** Results: ['478 Mbits/sec', '480 Mbits/sec']
mininet> iperf h2 h8
*** Iperf: testing TCP bandwidth between h2 and h8
*** Results: ['478 Mbits/sec', '479 Mbits/sec']
mininet> iperf h3 h1
*** Iperf: testing TCP bandwidth between h3 and h1
*** Results: ['478 Mbits/sec', '481 Mbits/sec']
mininet> iperf h3 h2
*** Iperf: testing TCP bandwidth between h3 and h2
*** Results: ['478 Mbits/sec', '479 Mbits/sec']
```

```
mininet> iperf h3 h4
*** Iperf: testing TCP bandwidth between h3 and h4
*** Results: ['478 Mbits/sec', '480 Mbits/sec']
mininet> iperf h3 h8
*** Iperf: testing TCP bandwidth between h3 and h8
*** Results: ['478 Mbits/sec', '481 Mbits/sec']
mininet> iperf h4 h1
*** Iperf: testing TCP bandwidth between h4 and h1
*** Results: ['478 Mbits/sec', '481 Mbits/sec']
mininet> iperf h4 h2
*** Iperf: testing TCP bandwidth between h4 and h2
*** Results: ['478 Mbits/sec', '479 Mbits/sec']
mininet> iperf h4 h3
*** Iperf: testing TCP bandwidth between h4 and h3
*** Results: ['478 Mbits/sec', '481 Mbits/sec']
mininet> iperf h4 h8
*** Iperf: testing TCP bandwidth between h4 and h8
*** Results: ['478 Mbits/sec', '480 Mbits/sec']
mininet> iperf h8 h1
*** Iperf: testing TCP bandwidth between h8 and h1
*** Results: ['478 Mbits/sec', '479 Mbits/sec']
mininet> iperf h8 h2
*** Iperf: testing TCP bandwidth between h8 and h2
*** Results: ['478 Mbits/sec', '493 Mbits/sec']
mininet> iperf h8 h3
*** Iperf: testing TCP bandwidth between h8 and h3
*** Results: ['478 Mbits/sec', '481 Mbits/sec']
mininet> iperf h8 h4
*** Iperf: testing TCP bandwidth between h8 and h4
*** Results: ['478 Mbits/sec', '481 Mbits/sec']
```

UNIVERSITÀ DI TRENTO

# TEST RESULTS

- When management slice is activated h5 and h7 can communicate with a bandwidth of 300 Mbps for TCP.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X X X X
h2 -> X X X X X X X
h3 -> X X X X X X X
h4 -> X X X X X X X
h5 -> X X X X X h7 X
h6 -> X X X X X X X
h7 -> X X X X h5 X X
h8 -> X X X X X X X
*** Results: 96% dropped (2/56 received)
```

```
mininet> iperf h5 h7
*** Iperf: testing TCP bandwidth between h5 and h7
*** Results: ['287 Mbits/sec', '289 Mbits/sec']
mininet> iperf h7 h5
*** Iperf: testing TCP bandwidth between h7 and h5
*** Results: ['287 Mbits/sec', '289 Mbits/sec']
```

- UDP bandwidth testing with *iperf* command
- h7 as server listening on port 5555 and h5 as client
- Bandwidth of 400 Mbps for UDP

```
------------------------------------------------------------
Server listening on UDP port 5555
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.5 port 5555 connected with 10.0.0.7 port 51823
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  5]  0.0- 1.0 sec  45.5 MBytes   382 Mbits/sec   0.030 ms    9/32477 (0.028%)
[  5]  1.0- 2.0 sec  45.9 MBytes   385 Mbits/sec   0.034 ms    0/32706 (0%)
[  5]  2.0- 3.0 sec  45.9 MBytes   385 Mbits/sec   0.033 ms    0/32720 (0%)
[  5]  3.0- 4.0 sec  45.8 MBytes   384 Mbits/sec   0.038 ms    0/32693 (0%)
[  5]  4.0- 5.0 sec  45.6 MBytes   383 Mbits/sec   0.041 ms    0/32551 (0%)
[  5]  5.0- 6.0 sec  45.8 MBytes   384 Mbits/sec   0.142 ms   11/32652 (0.034%)
[  5]  6.0- 7.0 sec  45.7 MBytes   383 Mbits/sec   0.037 ms    0/32590 (0%)
[  5]  7.0- 8.0 sec  45.8 MBytes   384 Mbits/sec   0.034 ms    0/32642 (0%)
[  5]  8.0- 9.0 sec  45.7 MBytes   383 Mbits/sec   0.031 ms   70/32663 (0.21%)
[  5]  0.0-10.0 sec   457 MBytes   384 Mbits/sec   0.168 ms  118/326232 (0.036%)
```

```
------------------------------------------------------------
Client connecting to 10.0.0.5, UDP port 5555
Sending 1470 byte datagrams, IPG target: 14.02 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.7 port 51823 connected with 10.0.0.5 port 5555
[  5] WARNING: did not receive ack of last datagram after 10 tries.
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec   457 MBytes   384 Mbits/sec
[  5] Sent 326232 datagrams
```

# TEST RESULTS

— When only development slice is activated h6 and h1 can communicate with each other with the bandwidth of 200Mbps

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X h6 X X
h2 -> X X X X X X X
h3 -> X X X X X X X
h4 -> X X X X X X X
h5 -> X X X X X X X
h6 -> h1 X X X X X X
h7 -> X X X X X X X
h8 -> X X X X X X X
*** Results: 96% dropped (2/56 received)
```

```
mininet> iperf h1 h6
*** Iperf: testing TCP bandwidth between h1 and h6
*** Results: ['191 Mbits/sec', '208 Mbits/sec']
mininet> iperf h6 h1
*** Iperf: testing TCP bandwidth between h6 and h1
*** Results: ['153 Mbits/sec', '155 Mbits/sec']
```

# CONTRIBUTORS

Elif Ruveyha Ozmen [238946]
elifruveyha.ozmen@studenti.unitn.it

Alessandro Migotto [238774]
alessandro.migotto@studenti.unitn.it

# THANK YOU