

Lab practice 1: Handling the Mininet API

In the first practice of the course, we will make a brief introduction to Mininet. Mininet is probably the most widely used tool for testing software-defined networking implementations. Mininet is a network emulator that allows the user to create virtual topologies composed of networking devices, end hosts, and virtual links. The virtual network devices support the OpenFlow protocol, which will be essential for programming the desired behavior in network equipment. Additionally, terminal equipment (computers or hosts) runs a standard Linux version.

To develop this practice, you must have the Mininet virtual machine provisioned in virtualization software (e.g. VirtualBox) or a direct installation of Mininet in a Linux distribution. Ideally, you should first develop the practice in the Mininet installation you have on your computer and after you have tested your solution, submit it for evaluation.

In the lab practices, we will highlight with **green** color the commands that must be done in the bash (or console) of the Mininet virtual machine, with **blue** color the Python code and with **red** color the commands that are entered inside the Mininet application. Console responses will not be highlighted.

To begin interacting with the Mininet command-line interface, we can use the following command (commands will be executed on a Linux distribution known as Ubuntu):

```
mininet@mininet-vm:~$ sudo mn
```

After we type this command, Mininet will start running, creating a simple network and adding a controller (which we will visit in the next practice), a switch and two hosts connected to the same switch:

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
```



```
//////  
*** Adding links:  
(h1, s1) (h2, s1)  
*** Configuring hosts  
h1 h2  
*** Starting controller  
c0  
*** Starting 1 switches  
s1 ...  
*** Starting CLI:  
mininet>
```

Once we are in the command line interface, we will be able to execute some useful instructions. We invite you to investigate the information they provide or the function of each of them:

```
mininet> help  
mininet> nodes  
mininet> net  
mininet> dump  
mininet> intfs  
mininet> h1 ping -c 1 h2  
mininet> pingall  
mininet> xterm h1  
mininet> dpctl dump-flows  
mininet> exit
```

More detailed information about the commands that you can use in the Mininet console can be found in the following resource: [Mininet Walkthrough](#).

Now, Mininet allows us to emulate topologies of different types and includes some typical topologies that can be invoked from the Mininet start command. We invite you to explore the following instructions and look at the topology that is created after executing each of them:

```
mininet@mininet-vm:~ sudo mn --topo single,4  
mininet@mininet-vm:~ sudo mn --topo linear,3  
mininet@mininet-vm:~ sudo mn --topo tree,2,3
```

It is advisable to clean the elements instantiated by Mininet each time we are going to emulate a new topology, this is achieved with the command:

```
mininet@mininet-vm:~ sudo mn -c
```

Mininet Python API

Mininet has an API (Application Programming Interface) that allows us to build custom topologies with a good level of detail, taking advantage of the benefits of Python scripting (remember to review the Python course at this [link](#) if needed). We invite you to explore this functionality with the following example:

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.log import setLogLevel

class CustomTopo(Topo):
    def __init__(self, **opts):
        Topo.__init__(self, **opts)

        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')

        s1 = self.addSwitch('s1')

        self.addLink(h1, s1)
        self.addLink(h2, s1)
        self.addLink(h3, s1)
        self.addLink(h4, s1)

def runNet():
    topo = CustomTopo()
    net = Mininet(topo)
```



```
net.start()  
net.pingAll()  
net.stop()  
  
if __name__ == '__main__':  
    setLogLevel('info')  
    runNet()
```

The “Topo” class contains methods that allow adding switches, hosts, and links between them. The “Mininet” class allows interacting directly with the application and sending commands. The previous script creates a network consisting of a switch, four hosts, and they are connected as specified in *Figure 1*. Additionally, it includes a function that starts Mininet with the topology defined in the created class and performs a connectivity test on all hosts, ending with the application closure. Detailed information about the classes and methods included in the Mininet API can be found in the link: [Mininet Python API Reference Manual: Class List](#).

To run this topology in Mininet just save the above script with the .py extension in the Mininet virtual machine and run it as a python application:

```
mininet@mininet-vm:~$ sudo python CustomTopo.py
```

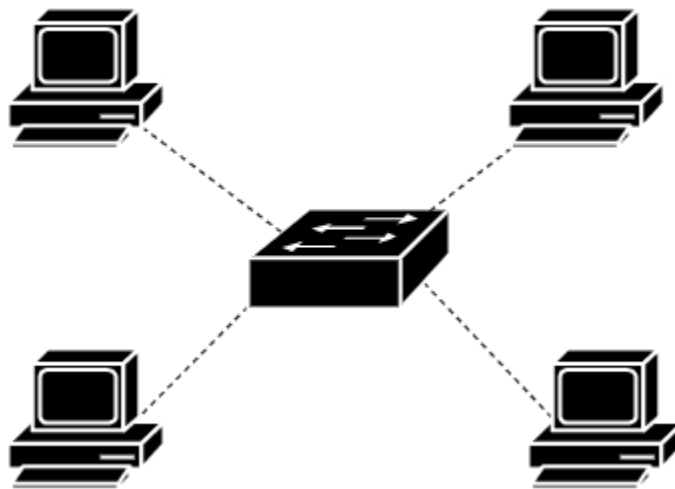


Figure 1. Topology in Mininet

Through the API we can customize the features of hosts and links, using additional classes and methods. We invite you to analyze the effect of adding the following parameters to the methods in order to create links in the above example:

```
self.addLink(h1, s1, bw=10, delay='1ms', loss=0.1,
max_queue_size=1000, use_htb=True)
self.addLink(h2, s1, bw=50, delay='3ms', loss=0.1,
max_queue_size=1000, use_htb=True)
self.addLink(h3, s1, bw=100, delay='10ms',
loss=0.1,max_queue_size=1000, use_htb=True)
self.addLink(h4, s1, bw=80, delay='50ms', loss=0.1,
max_queue_size=1000, use_htb=True)
```

For everything to work, you must import the following resources:

```
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
```

Additionally, you must modify the “runNet” function to include the links with capacity limitations:

```
def runNet():
    topo = CustomTopo()
    net = Mininet(topo, host=CPULimitedHost, link=TCLink)
    net.start()
    dumpNodeConnections(net.hosts)
    net.pingAll()
    h1, h4 = net.get('h1', 'h4')
    net.iperf((h1, h4))
    net.stop()
```

All of the above allows configuring parameters related to the link transmission speed, delay, packet loss, and buffer length. Additionally, the function that starts the application includes the “net.iperf” method, which performs performance tests between two hosts.

Activity

In order to test your knowledge, you must implement a mesh topology as shown in *Figure 2*. In this, all switches are connected to each other and there is one host connected to each switch. The number of switches can be any integer number, an example with four switches is implemented in *Figure 2*.

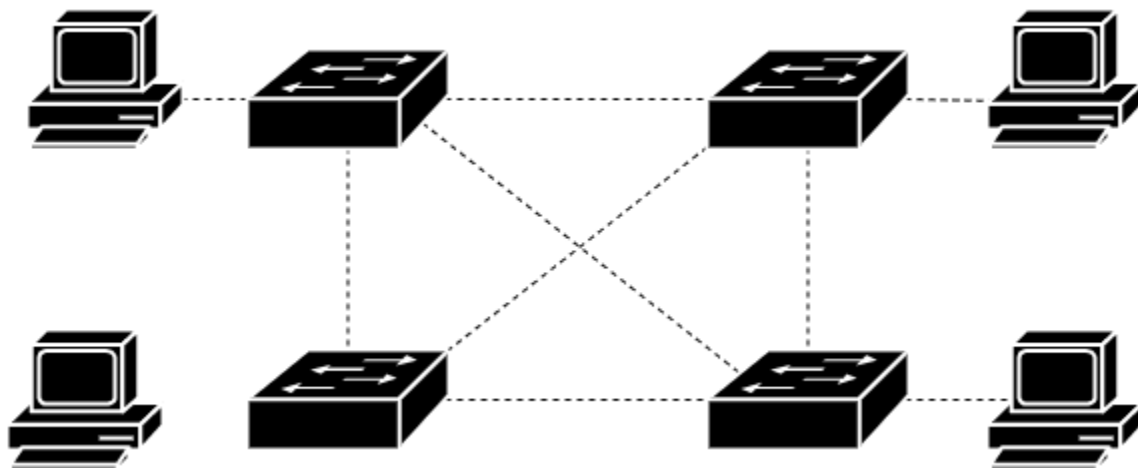



Figure 2. Mesh Topology with four switches

Mesh: If there are n switches, there must be $n*(n-1)/2$ connections between them.

If you wish to test your code on your installed Mininet virtual machine, you can add to the provided skeleton in the `runNet()` method the following lines:

```
def runNet():  
    topo = CustomTopo()  
    net = Mininet(topo)  
    net.start()  
    net.pingAll()  
    net.stop()
```



This will start the topology and execute a ping between all nodes. In this case, to avoid the broadcast storms, we recommend using switches that form a spanning tree. For this, the following line can be used to add the switches:

```
self.addSwitch('sx',cls=OVSBridge, stp=True)
```

In order to complete the practice, take the skeleton provided in the platform and include the portion of code that you consider is appropriate to comply with the activity.