Building a simple SDN network using Mininet

Part 1: Preparing the environment

In order to be able to build an SDN network we are going to use Mininet (already prepared and given as VM with this lab).

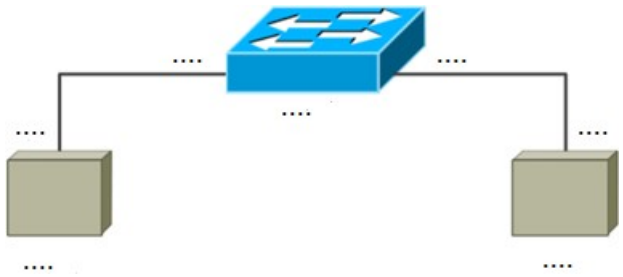
Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine (VM, cloud or native). Mininet is also a great way to develop, share, and experiment with OpenFlow and Software-Defined Networking systems. Mininet is actively developed and supported, and is released under a permissive BSD Open Source license.

Steps to prepare the environment are described in this web page : http://mininet.org/download/

Part 2: Get started

1. Create a simple network on the Mininet VM, as shown in the figure bellows. Use the command ***sudo mn –h*** to see all options.

`sudo mn --mac --controller="none”` will create the following topology.



2. Try the following commands in the Mininet CLI to get an understanding of the network and fill in the blanks:

```
mininet> nodes

mininet> net

mininet> dump
```

3. Attempt pings between the hosts. Do the pings succeed ? Why ?

```
mininet> h1 ping h2
mininet> h2 ping h1
```

4. Check the flow table of the switch, Can you explain the reason for what you see?

```
mininet> dpctl dump-flows
```

Part 3 : Setting up different networks

Mininet offers different network topologies: minimal, single, reserved, linear and tree. In this part, we are going to build these different topologies. Do not forget to clear at each step all existing topology and commands using `sudo mn -c`

1. Minimal

Minimal topology is very simple topology that contains 1 OpenFlow switch and 2 hosts. It also creates links between switch and two hosts.

Use the command `sudo mn --topo minimal` and report the configuration with different links and used interfaces.

2. Single

It is a simple topology with one OpenFlow switch and k hosts. It also creates a link between switch and k hosts. Use the command `sudo mn --topo single,4` to create a network with 4 hosts. You should report the topology and indicate different links, used interfaces and configuration.

3. Reversed

Use the command `sudo mn --topo reversed,4` to create a reversed network topology. Indicate the difference between the actual topology and the single topology.

4. Linear

Linear topology contains k switches and k hosts. It also creates a link between each switch and each host and among the switches.

Use the command `sudo mn --topo linear,4` to create a topology with 4 switches and 4 hosts. You should report the topology and indicate different links, used interfaces and configuration.

5. Tree

Tree topology contains k levels and 2 hosts are attached to per switch.

Use the command `mn --topo tree,3` , to create a topology with 3 levels. You should report the topology and indicate different links, used interfaces and configuration.

Part 4 : Setting up a customized network

Using Mininet, you can easily create customized topologies. To create a customized topology, you need just to write a few lines of Python code. You can also easily create very complex flexible, robust topology.

```
from mininet.topo import Topo
from mininet.cli import CLI

class MyFirstTopo(Topo):

        def __init__( self ):
                Topo.__init__( self )
                h1 = self.addHost( 'h1' )
                h2 = self.addHost( 'h2' )
                h3 = self.addHost( 'h3' )
                h4 = self.addHost( 'h4' )

                leftSwitch = self.addSwitch( 's1' )
                rightSwitch = self.addSwitch( 's2' )
                self.addLink( h1, leftSwitch )
                self.addLink( h2, leftSwitch )
                self.addLink( leftSwitch, rightSwitch )
                self.addLink( rightSwitch, h3 )
                self.addLink( rightSwitch, h4 )

topos = { 'myfirsttopo': ( lambda: MyFirstTopo() ) }
```

You can also configured that topology based on the parameters that are to be pass to it, and reuse that topology for multiple experiments.

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class SingleSwitchTopo(Topo):

    "Single switch connected to n hosts."

    def __init__(self, n=2, **opts):
        Topo.__init__(self, **opts)
        switch = self.addSwitch('s1')
        for h in range(n):
            #Each host gets 50%/n of system CPU
            host = self.addHost('h%s' % (h + 1), cpu=.5/n)
            #10 Mbps, 5ms delay, 0% Loss, 1000 packet queue
            self.addLink(host, switch, bw=10, delay='5ms', loss=0,
max_queue_size=1000, use_htb=True)
def perfTest():
    "Create network and run simple performance test"
     topo = SingleSwitchTopo(n=4)
     net = Mininet(topo=topo,host=CPULimitedHost, link=TCLink)
     net.start()
     print "Dumping host connections"
     dumpNodeConnections(net.hosts)
     print "Testing network connectivity"
     net.pingAll()
     print "Testing bandwidth between h1 and h4"
     h1, h4 = net.get('h1', 'h4')
     net.iperf((h1, h4))
     net.stop()

 if __name__=='__main__':
     setLogLevel('info')
     perfTest()
```

In the given Python code, There are number of classes, functions, methods and variables.

- Topo: It is a base class for Mininet topologies.
- Add Host (name, cpu = f): It is used for adding a host to the topology which contains two parameters. First parameter specifies the name of host and second specifies the fraction of overall system CPU resources that is to be allocated to the virtual host.
- Add Switch(): It is used for adding a switch to the topology and returns the switch name, for example s1.
- Add Link (node1, node2, **link options list): It is used for adding a bidirectional link which contains three parameters. The first, second parameter specify the host and switch name respectively and third parameter specify the dictionary that contain number of options such as bandwidth, delay and loss characteristics, with a maximum queue size.
- start(): It is used for starting your network.

- stop(): It is used for stopping your network.
- Mininet: It is used as a main class to create and manage a network.
- net. hosts: It is used to show all the hosts in network.
- ping All (): This is used to check connectivity between all nodes.
- Set Log Level: There are number of Log Level such as info, debug, and output. Info is recommended as it provides useful information.
- Dump Node Connections (): Dumps connections to/from a set of nodes.

Part 5 : Controlling default topology with DPCTL

1. Create a topology with 1 switch and 3 hosts. Using this command :

   ```
   mn --topo single,3 --mac --controller remote
   ```

   The option 'mac' set the mac address of each host according to node number and option 'remote' is to be used to connect switch to remote controller.

2. Try to ping between hosts. Explain what you see. (you can visualize the flow table of s1 using this command : `s1 dpctl dump-flows tcp:127.0.0.1:6634)`

3. There are two methods to add flow entries into flow table of switch, remote controller and 'dpctl' utility that works on port 6634. Dpctl is a data path controller that comes with OpenFlow reference distribution and is used to manage the flow table of switch by using 'dpctl' commands. Use the following commands to add flows :

```
S1 dpctl add-flow tcp:127.0.0.1:6634 in_port=2,actions=output:1

S1 dpctl add-flow tcp:127.0.0.1:6634 in_port=1,actions=output:2
```

4. Try to ping between hosts. Connectivity test should succeed.
5. Continue the exercise to install flow table for all hosts using MAC addresses as destination of the flows

```
S1 dpctl add-flow tcp:127.0.0.1:6634 dl_dst=0:0:0:0:0:1,actions=output:1

S1 dpctl add-flow tcp:127.0.0.1:6634 dl_dst=0:0:0:0:0:2,actions=output:2

S1 dpctl add-flow tcp:127.0.0.1:6634 dl_dst=0:0:0:0:0:3,actions=output:3
```

6. You can add the rule for packets forwarding in case of broadcast.

```
S1 dpctl add-flow tcp:127.0.0.1:6634 dl_dst=ff:ff:ff:ff:ff:ff,idle_timeout=1000,actions=flood
```