

CS3543 Lab Assignment for Feb 19th (Deadline: 23:59 on February 25th (TUE), 2020)

Member 1: SAI HARSHA KOTTAPALLI (CS17BTECH11036)

Member 2: SAGAR JAIN (CS17BTECH11034)

# General Information

1. This assignment is a pair assignment. The same mark will be offered to the pair of students regardless of individual contributions.
2. The assignment is customized for Ubuntu + KVM environment. It is highly recommended for non-Ubuntu users to enable dual boot on your laptop computer and install Ubuntu. If you would like to work on another operating system and virtualization platform, you need to interpret the Ubuntu/KVM terminology to another environment's terminology.
3. Each pair can create a locally copy of this question file, give the answer to the local copy, and submit in a form of PDF file.
4. Only one submission is good enough as far as the student name and ID are properly mentioned.
5. Do not send any private comment to separately mention the buddy.

### Question 1.

Install Mininet as a VM and SDN Controller of your choice. Form a custom network topology using Mininet Hosts and Open vSwitch (OVS) given in Figure 1. Manually configure IP address to each Mininet Host. It is highly recommended to give IP addresses in the same prefix to avoid routing in the network. Activate OVSes as stand-alone learning switches without SDN Controller. Launch Mininet using “-x” options so that you will get terminal access to Mininet Hosts and Open vSwitches.

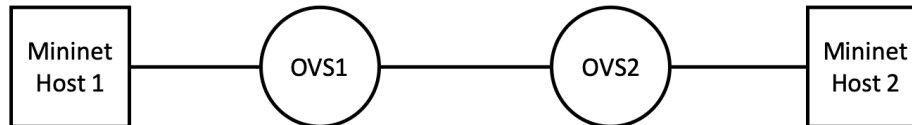


Figure 1.

#### 1.1. Give the screen capture of Mininet Configuration for the custom topology creation.

```
mininet@mininet-vm:~$ sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --topo mytopo -x --controller=none
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s3 s4
*** Adding links:
(h1, s3) (s3, s4) (s4, h2)
*** Configuring hosts
h1 h2
*** Running terms on :0
*** Starting controller

*** Starting 2 switches
s3 s4 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

```

from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost = self.addHost( 'h1', ip='10.0.0.1/24' )
        rightHost = self.addHost( 'h2', ip='10.0.0.4/24' )
        leftSwitch = self.addSwitch( 's3', failMode='standalone' )
        rightSwitch = self.addSwitch( 's4', failMode='standalone' )

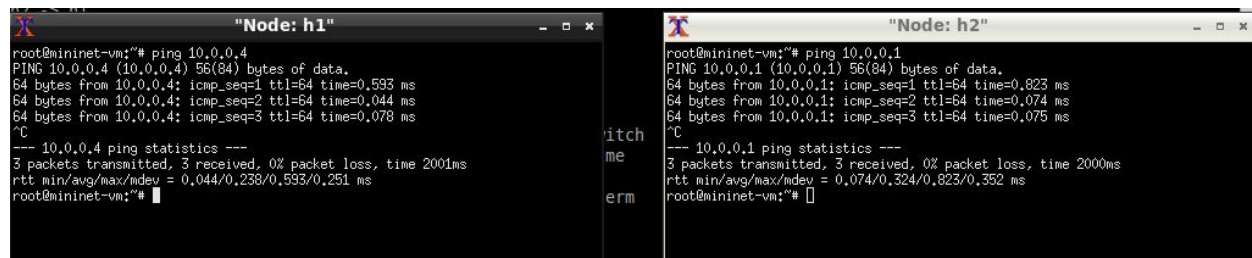
        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )

topos = { 'mytopo': ( lambda: MyTopo() ) }

```

1.2. Run ping and iperf between Mininet Hosts, and insert the screen captures of the results.

### ping



The image shows two terminal windows side-by-side. The left window is titled "Node: h1" and shows the output of a ping command from h1 to 10.0.0.4. The right window is titled "Node: h2" and shows the output of a ping command from h2 to 10.0.0.1. Both windows show successful ping results with 0% packet loss and round-trip times around 0.25ms.

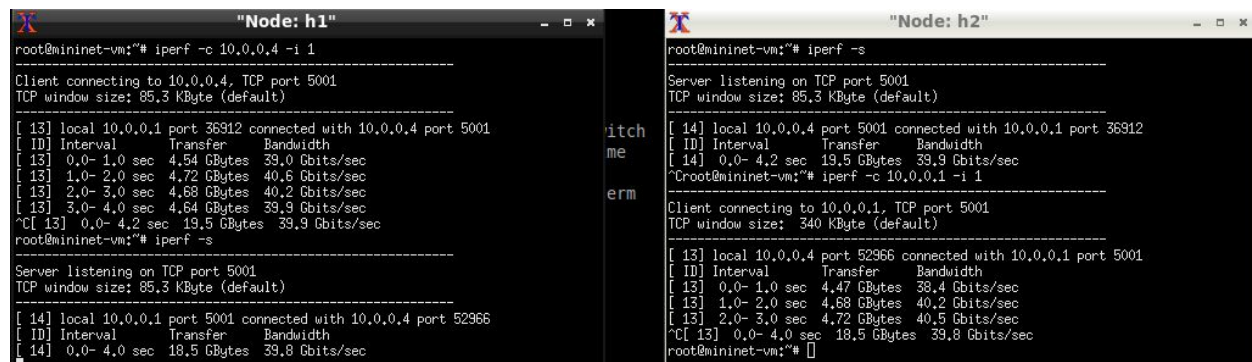
```

root@mininet-vm:~# ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data:
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.593 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.044 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.078 ms
^C
--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.044/0.238/0.593/0.251 ms
root@mininet-vm:~#

root@mininet-vm:~# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data:
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.823 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.074 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.075 ms
^C
--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.074/0.324/0.823/0.352 ms
root@mininet-vm:~#

```

### iperf



The image shows two terminal windows side-by-side. The left window is titled "Node: h1" and shows the output of an iperf client command from h1 to 10.0.0.4. The right window is titled "Node: h2" and shows the output of an iperf server command on h2, followed by an iperf client command from h2 to 10.0.0.1. Both windows show successful iperf results with bandwidths around 39.9 Gbits/sec.

```

root@mininet-vm:~# iperf -c 10.0.0.4 -i 1

Client connecting to 10.0.0.4, TCP port 5001
TCP window size: 85.3 KByte (default)

[ 13] local 10.0.0.1 port 36912 connected with 10.0.0.4 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0- 1.0 sec  4.54 GBytes 39.0 Gbits/sec
[ 13] 1.0- 2.0 sec  4.72 GBytes 40.6 Gbits/sec
[ 13] 2.0- 3.0 sec  4.68 GBytes 40.2 Gbits/sec
[ 13] 3.0- 4.0 sec  4.64 GBytes 39.9 Gbits/sec
^C[ 13] 0.0- 4.2 sec 19.5 GBytes 39.9 Gbits/sec
root@mininet-vm:~# iperf -s

Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)

[ 14] local 10.0.0.1 port 5001 connected with 10.0.0.4 port 52966
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0- 1.0 sec  4.47 GBytes 38.4 Gbits/sec
[ 13] 1.0- 2.0 sec  4.68 GBytes 40.2 Gbits/sec
[ 13] 2.0- 3.0 sec  4.72 GBytes 40.5 Gbits/sec
^C[ 13] 0.0- 4.0 sec 18.5 GBytes 39.8 Gbits/sec
root@mininet-vm:~#

```

## Question 2.

Form a custom network topology using Mininet Hosts, Open vSwitch (OVS) and SDN controller of your choice given in Figure 2. Activate OVSeS as SDN Switches that need instruction by SDN Controller. Activate SDN controller using the standard Learning Switch code, which is available as a sample code bundled with most SDN Controllers. Refer to the tutorials of Mininet and each SDN Controller if needed.

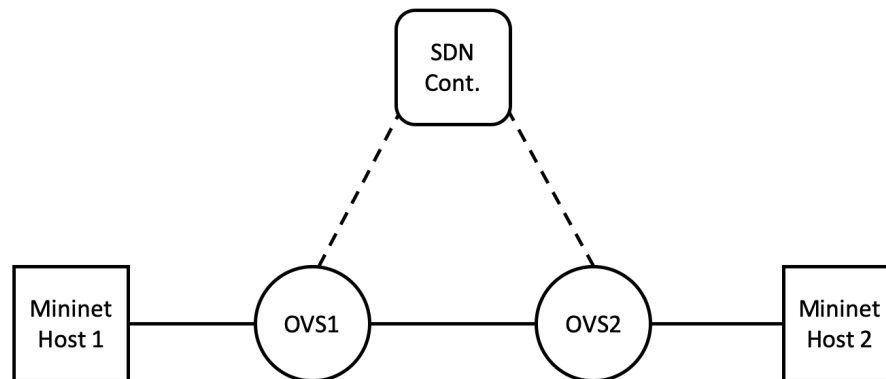


Figure 2.

2.1. Which SDN controller did you use? Give its name and version.

**ryu 4.34**

2.2. Run ping and iperf between Mininet Hosts, and insert the screen captures of the results.

```
root@mininet-vm:~# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data:
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=3.35 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=1.75 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.076 ms
^C
--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.076/1.727/3.353/1.338 ms
root@mininet-vm:~#

root@mininet-vm:~# ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data:
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=2.30 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=1.27 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.072 ms
^C
--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 0.072/1.215/2.300/0.910 ms
root@mininet-vm:~#

root@mininet-vm:~# iperf -c 10.0.0.1
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)

[ 15] local 10.0.0.4 port 47830 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 15] 0.0-10.0 sec  48.0 GBytes  41.2 Gbits/sec
root@mininet-vm:~# iperf -s

Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)

[ 16] local 10.0.0.4 port 5001 connected with 10.0.0.1 port 52248
^CWaiting for server threads to complete. Interrupt again to force quit.
[ ID] Interval      Transfer      Bandwidth
[ 16] 0.0-10.0 sec  50.5 GBytes  43.4 Gbits/sec
root@mininet-vm:~#

root@mininet-vm:~# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)

[ 15] local 10.0.0.1 port 5001 connected with 10.0.0.4 port 47830
^CWaiting for server threads to complete. Interrupt again to force quit.
[ ID] Interval      Transfer      Bandwidth
[ 15] 0.0-10.0 sec  48.0 GBytes  41.2 Gbits/sec
root@mininet-vm:~# iperf -c 10.0.0.4

Client connecting to 10.0.0.4, TCP port 5001
TCP window size: 85.3 KByte (default)

[ 16] local 10.0.0.1 port 52248 connected with 10.0.0.4 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 16] 0.0-10.0 sec  50.5 GBytes  43.4 Gbits/sec
root@mininet-vm:~#
```

2.2. Execute the command to dump the flow table of OVS1 to CLI and insert the screenshot.

```

mininet> dpctl dump-flows
*** s3 -----
NXST FLOW reply (xid=0x4):
 cookie=0x0, duration=18.927s, table=0, n_packets=2, n_bytes=140, idle_age=13, in_port=1,dl_src=92:23:92:98:ed:7b,dl_dst=8a:4
e:2f:e5:33:1e actions=output:2
 cookie=0x0, duration=18.928s, table=0, n_packets=3, n_bytes=238, idle_age=13, in_port=2,dl_src=8a:4e:2f:e5:33:1e,dl_dst=92:2
3:92:98:ed:7b actions=output:1
*** s4 -----
NXST FLOW reply (xid=0x4):
 cookie=0x0, duration=18.939s, table=0, n_packets=2, n_bytes=140, idle_age=13, in_port=1,dl_src=92:23:92:98:ed:7b,dl_dst=8a:4
e:2f:e5:33:1e actions=output:2
 cookie=0x0, duration=18.94s, table=0, n_packets=3, n_bytes=238, idle_age=13, in_port=2,dl_src=8a:4e:2f:e5:33:1e,dl_dst=92:23
:92:98:ed:7b actions=output:1
mininet>

```

2.3. Explain the flow rule about the communication between Mininet Hosts 1 and 2. Give the human understandable interpretation of these flow rules.

When h1 ping h2 and vice versa, first the switch receives the packet. Since, the switch doesn't know what to do with (forward to where) it asks the SDN controller for the details. The SDN controller then tells where to forward the packet and that entry is stored into the flow table of the switch. The flow table has two such rules for forwarding packets to either direction. Each rule has details such as Cookie (*an opaque data value that is set by the controller*), acts as an identifier to flow entry which can map back to internal state associated with it, need not be unique), duration (time since, it has been added), table (table id where it is stored), n\_packets (num of packets sent), n\_bytes (number of bytes sent), idle\_age (seconds not in use), in\_port (port number), dl\_src (source), dl\_dst (destination), actions (action to be taken that is usually the place to forward when such input packet is received).

Question 3.

Form a custom network topology using Mininet Hosts, Open vSwitch (OVS) and SDN controller of your choice given in Figure 3. Activate OVSes as SDN Switches that need instruction by SDN Controller. Activate SDN controller using the standard Learning Switch code, which is available as a sample code bundled with most SDN Controllers. Refer to the tutorials of Mininet and each SDN Controller if needed.

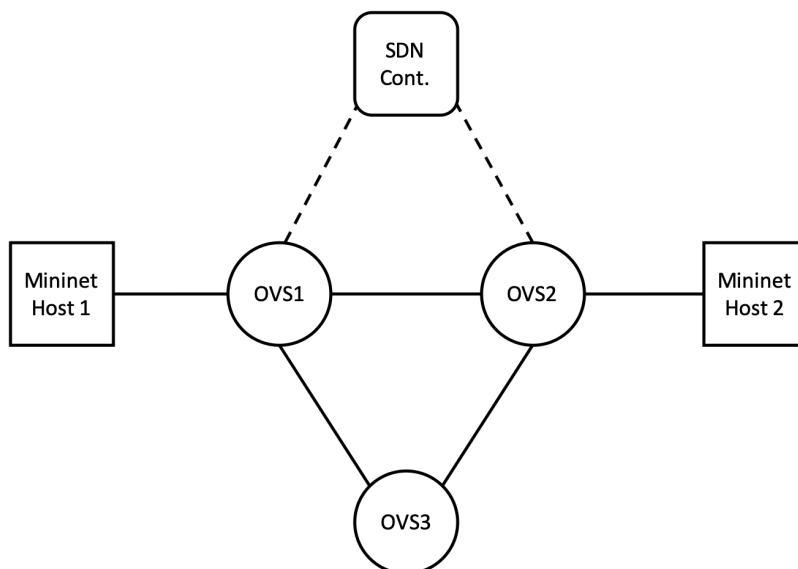


Figure 3.



3.1. When you enable the SDN controller and let Mininet Hosts 1 and 2 communicate, you will see the phenomenon called Broadcast Storm. Insert the screen capture of the console of SDN Controller or OVS that indicate Broadcast Storm.

```
packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 2
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 2
packet in 3 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 2
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 3
packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 3
packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 3
packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 3
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 3
packet in 3 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
packet in 3 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
packet in 3 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
packet in 3 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 2
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 2
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 2
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 2
packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 3
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 3
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 3
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 3
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 3
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 2
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
EVENT ofp_event->SimpleSwitch EventOFPPacketIn
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 2
packet in 3 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 2
packet in 3 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 2
packet in 3 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 2
packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 2
```

```

mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable
From 10.0.0.1 icmp_seq=6 Destination Host Unreachable
From 10.0.0.1 icmp_seq=7 Destination Host Unreachable
From 10.0.0.1 icmp_seq=8 Destination Host Unreachable
From 10.0.0.1 icmp_seq=9 Destination Host Unreachable
From 10.0.0.1 icmp_seq=10 Destination Host Unreachable
From 10.0.0.1 icmp_seq=11 Destination Host Unreachable
From 10.0.0.1 icmp_seq=12 Destination Host Unreachable
^C
--- 10.0.0.2 ping statistics ---
13 packets transmitted, 0 received, +12 errors, 100% packet loss, time 12085ms
pipe 3
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 5 links
.....
*** Stopping 3 switches
s1 s2 s3
*** Stopping 2 hosts
h1 h2
*** Done
completed in 38.218 seconds
mininet@mininet-vm:/home$ _

```

3.2. Implement “My STP (Spanning Tree Protocol)” so that you can avoid Broadcast Storm in the network. Demonstrate “My STP” is working properly to TAs.

3.3. As part of My STP implementation, you need to get the network topology information. Insert the screen capture of the code where you get the topology information using the API of SDN controller. You must not import the configuration of Custom Topology as part of the implementation.

```
def TopoInfo(self):
    for item, value in self.net.items():
        if value.__class__.__name__ in CONTROLLERS_TYPES:
            self.Nodes.append({'name': item, 'widget': None, 'type': value.__class__.__name__, 'ip': value.ip, 'port': value.port, 'color': self.Controller_color})
        elif value.__class__.__name__ in SWITCHES_TYPES:
            self.Nodes.append({'name': item, 'widget': None, 'type': value.__class__.__name__, 'dpid': value.dpid, 'color': None, 'controllers': []})
        elif value.__class__.__name__ in HOSTS_TYPES:
            if self.appPrefs['displayHosts'] == 1:
                self.Nodes.append({'name': item, 'widget': None, 'type': value.__class__.__name__, 'ip': value.IP(), 'color': None})
            else:
                continue
        else:
            self.Nodes.append(
                {'name': item, 'widget': None, 'type': value.__class__.__name__, 'color': None})

    #Gather interface info for all interfaces of a node
    for intf in value.intfList():
        intf2 = str(intf.link).replace(intf.name, '').replace('<->', '')
        if intf2 != 'None':
            self.intfData.append({'node': item, 'type': value.__class__.__name__, 'interface': intf.name, 'mac': intf.mac, 'ip': intf.ip,
                                'link': intf2, 'TXP': 0, 'RXP': 0, 'TXB': 0, 'RXB': 0})

    #To find and save the controller that each switch is connected to. Needed because there can be more than one controller.
    for switch in self.Nodes:
        if switch['type'] in SWITCHES_TYPES:
            try:
                switch_info = check_output(["ovs-vsctl", "get-controller", switch['name']])
            except:
                switch_info = '-1'
            first_controller = None
            for controller in self.Nodes:
                if controller['type'] in CONTROLLERS_TYPES:
                    controller_info = str(controller['ip']) + ':' + str(controller['port'])
                    if controller_info in switch_info:
                        switch['controllers'].append(controller['name'])
                    if first_controller == None:
                        first_controller = controller['name']
            # TODO: Assign user switch properly to the correct controller
            # Currently just assigning the first controller to the switch. Will work if there is only one controller.
            if switch['controllers'] == []:
                switch['controllers'].append(first_controller)
```

source: <https://github.com/fabiobento/stp-mininet-ryu-tutorial/blob/master/MiniNAM.py>

This code was used from the source for obtaining the topology information, the STP implementation was also inspired from here.

# Additional Instructions

1. STP must not be enabled on Open vSwitches

Done!!