Using Custom Mininet Topology Configuring L2-Switch in Opendaylight

Guruprasad E
Assistant Professor
Department of Computer Science and Engineering
Siddaganga Institute of Technology, Tumakuru
guruyathi@gmail.com

Sindhu G
M. Tech Student
Computer Networks Engineering
Siddaganga Institute of Technology, Tumakuru
Sindhugowdar22@gmail.com

Abstract: Software Defined Networking is a new way of making traditional hardware based network management into software based controlling. To do this, require a emulator like Mininet, a freely available open source network simulator software for creating custom topologies of our wish in virtual environment and it also permits to create OpenFlow switches, hosts and an SDN controller in a virtual network all with a simple command on a Linux Kernal.and It is a better approach to start practicing about SDN and Open-Flow as well as test SDN controllers like OpenDaylight. In this paper, by creating custom topology in python script and use of OpenDaylight controller environment for controlling flows in openflow enabled switches and then configuring L2-Switch features and observe the out comes.

Index terms— *Software Defined Networking(SDN), Mininet, OpenFlow, OpenDaylight, L2-Switch.*

*****

I. INTRODUCTION

Software Defined Networking (SDN) is estimated as the forthcoming backbone for all kinds of computer networks. Though the idea is yet in earliest stages and the greater part of its improvements are going on at Research and Development labs. It is pointing that such a network comprises of an extensive number of switches and controllers and the network among them. The topologies of these extensive scale network is of extraordinary significance for SDN study.

Although there does not occur an identical and automatic device for creating such topologies. Due to large-scale topologies, it is not to execute SDNs for all intents and purposes for testing. Therefore researchers choose the utilization of a product testbed (emulator) like Mininet to concentrate the same. SDN programming emulators acknowledge topologies from client input and the whole topology must be determined physically It is not practical to input large scale topologies manually as there are chances of human error.

A. SDN OVERVIEW:

SDN is a way to deal with utilizing open proocols, for example, OpenFlow, to apply universally known programming control at the edges of the network to get switches and routers that ordinarily would utilize locked and proprietary firmware.

SDN is another networking concept which summaries the reasonable portion of computer networks to an central controller. The thought originated from the work initially done at Stanford University. It isolates the control plane which chooses where and how activity is sent from the basic gadgets (data plane) which essentially direct data flows, and offers programmable interfaces to regulate network traffic. Isolating the control plane from the data plane is the most describing properties of SDN. This isolation improves network management configuration since there is a necessity for administrators to state hardware parameters in a low level. Programmability is one of the critical element of SDN: the difficult control logic can be characterized by software programs, which is considerably easier to execute and keep up. The control plane is expelled from network devices and they turn into simple data forwarding components and Control functionality is put on a devoted element called SDN controller.

Forwarding choices are flow based. A flow could be characterized as a bundle stream between a source and destination that get a similar forwarding service. The network is programmable through programming applications running on the highest point of controller

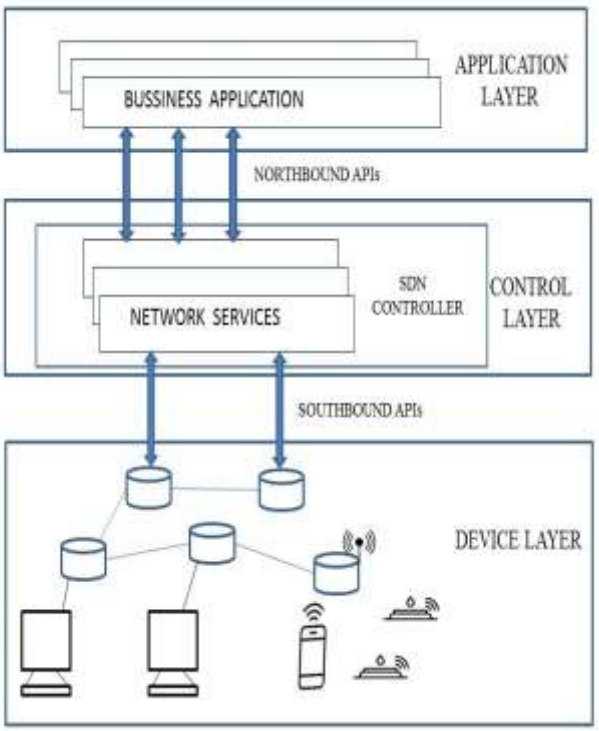

Figure1: Layered SDN Architecture

Figure shows the SDN architecture, which is being defined by the Open Networking Foundation (ONF). It basically consists of three layers:

- **Device layer**, which consists of the OpenFlow-enabled network elements, such as layer 2 switches and along with end devices like sensors and pc's etc.
- **Control layer**, which lets active provisioning of network services by means of an SDN controller.
- **Application layer**, which lets implement of network-aware applications on top of an SDN controller.

Northbound API presents a system reflection interface between applications that sit on the highest point of SDN stack and SDN controller.

Southbound APIs encourage productive system control and empower SDN controller to progressively roll out improvements in sending plane continuously. For instance, SDN controller can include or evacuate a section in sending table through southbound interface. OpenFlow is the most widely recognized southbound interface

B. OPENDAYLIGHT OVERVIEW:

OpenDaylight, an java-based open source platform for Software Defined Networking (SDN) that utilize open protocols to give unified, automatic control and manages network devices. In the same way as other SDN controllers, OpenDaylight provisions OpenFlow and other southbound APIs, (for example, Cisco OpFlex) and introduces network solutions as a major aspect of its platform. Much as your operating system gives an interface to the gadgets that contain your PC, OpenDaylight gives an interface that enables you to associate network gadgets rapidly and cleverly for optimal network performance.

Setting up your networking environment with OpenDaylight is not a solitary software installation. While your first sequential step is to install OpenDaylight, you install extra useful bundles as Karaf components to suit your particular needs.

The technologies have been using for OpenDaylight controller, with its inner core modules are MD-SAL and config subsystem then for Model-driven technologies are YANG, NETCONF and RESTCONF then for Project management tooling is Maven and Run-time environment are OSGi and Karaf finally Java environment are JMX, JConsole, etc

ODL supports for a extensive and rising collection of network protocols beyond OpenFlow, as well as SNMP, NETCONF, OVSDB, BGP, PCEP, LISP, etc and it Provision for creating new functionality consist of additional networking protocols and services.

C. MININET OVERVIEW:

Mininet a Linux kernel based network emulator which tracks pool of end-hosts, switches, routers and links, where these switches have to support OpenFlow protocol so as to test or implement SDN concepts. These virtualmininet's hosts, switches, links and controllers are created using software instead of real hardware devices. Mininet permits the user to simply design, modify, share and check SDN networks. Mininet will replicate SDN networks, will manage a controller for experiments. Mininet's VM comprise of some SDN controllers to imitate real world scenarios.

By a group of professors at Stanford University mininet was created and utilized as an apparatus for research work and for learning network technologies. Recently, Mininet is intended to make virtual software defined networks including an Open-Flow controller, multiple switches in a network which are Open-Flow enabled and various hosts associated with those switches. It has in-built functions that provisioning provisioning distinctive sorts of controllers and switches. Using Mininet's command line interface , can create SDN switches, hosts, controllers and links either by typing commands or by designing own custom topologies. SDN architecture divides the network control from forwarding functions by allowing the network control to become straightly programmable and the primary infrastructure to be isolated for applications and network services. The Open-Flow protocol is an initial element for developing SDN solutions and it is a standard protocol, used to convey a communication between controller, control plane and data plane respectively.

II. MININET CUSTOM TOPOLOGY

Mininet distributes a Python API to generate custom experiments, topologies, and node types: switch, controller, host, or other. A couple lines of Python are adequate to characterize a custom regression test that makes a network, executes commands on different nodes and showcases the outcomes. An example script:

```
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
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        h4 = self.addHost( 'h4' )
        s1 = self.addSwitch( 's1' )
        s2 = self.addSwitch( 's2' )

        # Add links
        self.addLink( h1, s1 )
        self.addLink( h2, s1 )
        self.addLink( s1, s2 )
        self.addLink( h3, s2 )
        self.addLink( h4, s2 )

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

Figure2: A Custom Topology python script with two switches and four hosts.

Creates a small network (4 hosts, 2 switches, 5 links) and pings one host from another when we invoke this script called mytopo present in the custom folder of mininet.

To make this network, Mininet simulates links, hosts, switches, and controllers. Mininet employs the lightweight virtualization procedure joined with the Linux OS.

The current Mininet distribution includes several example applications, including text-based scripts and graphical applications The hope is that the Mininet API will prove useful for system-level testing and experimentation, test network management, instructional materials, and applications that will surprise the authors.

III. CONFIGURING L2-SWITCH FEATURE IN OPENDAYLIGHT

The L2Switch development offers Layer2 switch functionality and it allows user to configure those features as per their requirements.

A. Components of the L2-Switch Project

Packet Handler: Decrypts the packets going to the controller and dispatches decoded packet notice, based on listed decoders.

- **Loop Remover**: Loops present in network will be expels and updates the comparing STP Status of network ports present in operational stock data store.
- **Arp Handler**: Handles the decrypted ARP packets, based on the configuration either by set up proactive flood flows or by transmitting packets back to network.
- **Address Tracker**: Find out the Addresses (MAC and IP) of devices in the network and follows them in operational inventory data store.
- **Host Tracker**: Follows the locations of hosts in the network and informs operational default topology tree in data store.
- **L2Switch Main**: Based on network traffic and address learned by address tracker, set up flows on each switch.

B. Configuring steps

Step1: Download the base distribution configuration files located in:
distribution/base/target/distributions-l2switch-base-0.1.0-SNAPSHOT osgipackage/opendaylight/configuration/initial

Step2: Find the Karaf distribution configuration files located in: distribution/karaf/target/assembly/etc/opendaylight/karaf
Here we are taken only Loop Remover (52-loopremover.xml) and in that showing outcome for changes did in graph refresh-delay.

A graph of the network is kept up, and gets refreshed as system components go up/down (that is to state, links go up/down and switches go up/down). After a network component going up/down, it holds up graph-refresh-delay seconds before re-processing the graph.

- A higher value has the benefit of doing less graph updates, at the possible cost of dropping some packets since the graph didn't update immediately.
- A lower value has the benefit of managing network topology changes faster, at the cost of doing more computation.

C. Observation

1. Graph-Refresh-Delay High

If we change install-lldp-flow to false then no flow will be installed on switches to send all lldp packets to the controller but our focus is on graph-refresh-delay so here we made this delay value high i,e. 1000000 ms thus it wll

update graph after this much of time if any changes made between this duration can not able to update fast.

```
<name>loop-remover-impl</name>
<is-install-lldp-flow>true</is-install-lldp-flow>
<lldp-flow-table-id>0</lldp-flow-table-id>
<lldp-flow-priority>100</lldp-flow-priority>
<lldp-flow-idle-timeout>0</lldp-flow-idle-timeout>
<lldp-flow-hard-timeout>0</lldp-flow-hard-timeout>
<graph-refresh-delay>1000000</graph-refresh-delay>
<topology-id>flow:1</topology-id>
```

Figure3: Graph made high value in xml file(52-loopremover.xml)

By pinging the host h1 and h2 and seeing the icmp echo reply time taken around 2ms at first ping and then time taking to ping will decrease.

```
mininet> h1 ping -c 5 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.57 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.488 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.315 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.075 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.047/0.499/1.570/0.559 ms
```

Figure4: pinging h1 to h2 in mininet_vm for checking icmp echo reply time taken to ping hosts if graph refresh delay made high.

2. Graph-Refresh-Delay Low

Here we made this delay value low i,e. 10 ms thus it wll update graph faster after this much of small time. if any changes made between this duration can able to update faster.

```
<name>loop-remover-impl</name>
<is-install-lldp-flow>true</is-install-lldp-flow>
<lldp-flow-table-id>0</lldp-flow-table-id>
<lldp-flow-priority>100</lldp-flow-priority>
<lldp-flow-idle-timeout>0</lldp-flow-idle-timeout>
<lldp-flow-hard-timeout>0</lldp-flow-hard-timeout>
<graph-refresh-delay>10</graph-refresh-delay>
<topology-id>flow:1</topology-id>
```

Figure5: Graph made low value in xml file(52-loopremover.xml)

By pinging the host h1 and h2 and seeing the icmp echo reply time taken around 0.5ms at first ping and then time taking to ping will decrease and by this we can come to conclusion that if any changes made with graph refresh delay low value no much time require for communication between devices.

```
mininet> h1 ping -c 5 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.461 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.046 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.056 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.058 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.046/0.134/0.461/0.163 ms
```

Figure6: pinging h1 to h2 in mininet_vm for checking icmp echo reply time taken to ping hosts if graph refresh delay made low.

IV. CONCLUSION

In this paper, we showed how to configure L2-Switch features of OpenDaylight in a xml file by creating Mininet custom topology and observed the out come of the same. By this we can came to conclusion that many things can be configure in OpenDaylight controller platform using Mininet so as to implement our own requirements easily.

The future work will be concentrating on all other remaining features of L2-Switch and extending this to OpenDaylight YangUi and developing own feature for L2-Switch.

REFERENCES

[1] Casimer DeCusatis, Aparicio Carranza and Jean Delgado-Caceres "Modeling Software Defined Networks using Mininet", ICCIST, 2016.

[2] Sunit Kumar Nandi "Topology generators for Software Defined Network testing", International Conference on Electrical, Electronics, and Optimization Techniques – 2016.

[3] Chaitra N. Shivayogimath and N.V. Uma Reddy "Modification of L3 Learning Switch Code for Firewall Functionality in POX Controller" International Journal of Research in Engineering and Technology, june 2015.

[4] Karamjeet Kaur, Japinder Singh and Navtej Singh Ghumman "Mininet as Software Defined Networking Testing Platform", International Conference on Communication, Computing & Systems, 2014.

[5] Kuldeep K. Sharma and Manu Sood "Mininet as a Container Based Emulator for Software Defined Networks", International Journal of Advanced Research in Computer Science and Software Engineering, 2014.

[6] Pooja and Manu Sood "SDN and Mininet: Some Basic Concepts", International Journal in Advanced Networking and Applications, 2015.

[7] Faris Keti and Shavan Askar "Emulation of Software Defined Networks Using Mininet in Different Simulation Environments", International Conference on Intelligent Systems, Modelling and Simulation, 2015 IEEE.