



Organized by

Dept. of Computer Science, Garden City University, Bangalore-560049, India

Implementation of SDN using Open Day Light Controller

Saleh Asadollahi, Dr. Bhargavi Goswami, Dr. Atul M Gonsai

Department of Computer Science, Saurashtra University, Rajkot, India

Department of Computer Science, Garden City University, Bangalore, India

Department of Computer Science, Saurashtra University, Rajkot, India

ABSTRACT—Software Defined Network is the most researcher's choice of research domain to make the internet an architecture independent structure which will bring the biggest leap in the domain of networks. The growth of network has gone mature and slow due to restricted architecture of traditional network which needs the changes from fundamental designing. Since 2010, till date, the modeling has been proposed by top research oriented universities worldwide. Now what is needed is to bring the imagination to reality by implementing the proposed ideas that brings solution as per researcher's expectation. This paper is the effort to help the researchers to implement the SDN Infrastructure such that further analysis and improvement can be looked into by research community. In this paper, we have shown step by step procedure to implement ODL – OpenDayLight Controller of SDN and development and implementation of desired Scenario.

KEYWORDS: SDN, OpenDayLight, Mininet, Swithe, Controller

I. INTRODUCTION

It is been 45 years that internet is used to interconnect network devices and share information. Ethernet protocol and devices are most well-worked, but circumstances changed, number of end devices that are connected to internet is more than one per person. Again, new technologies such as IoT provide connectivity and remote manageability to almost all devices via internet and thus, cause traffic increase rapidly. In such a situation, control of congestion in one side and managing Data center to store user's data in other side become a momentous subject of research. Configuring huge and diverse network equipment such as routers and switches, using low-level command and pre defending protocol to build up network topology among lack of innovation make researchers eager to work on that subject. Software Define Networking (SDN)[1][2] is a new approach in area of networking that appears to change the architecture of network devices such as switches by simplifying the intricate node structure. It provides one centralized controller (software based) among the ability of programmability and simple forwarded devices instead of multipart and messy devices. Different network society comes up with number of controller among different features and purposes. But, out of numerous available controllers with verity we opted for Open Daylight [3].

In this paper second part talks about problems of traditional network, third part describes SDN architecture and follow by next part explains the architecture of Open Daylight controller. Last part provides detailed step by step implementation of a SDN scenario on ODL controller with Mininet followed by conclusion and references.

II. ARCHITECTURE AND PROBLEM OF TRADITIONAL NETWORKING

In current networks, end devices that are responsible to route and forward packet through the network such as routers and switches have to perform three different activities. First: Data plan (process the transit traffic according to decision made by control plan), second: control plan (figures out and handle what's going on around, depend to its type and configuration) and third: management plan (converse with administrator).

Distributed and transport protocols are running inside of network protocol is the base of transiting traffic through IP network. Once the packet reaches to data plan through port, according to the header information and forwarding table,

An International Conference on Recent Trends in IT Innovations - Tec'afe 2017

Organized by

Dept. of Computer Science, Garden City University, Bangalore-560049, India

data plan will transmit it to proper port or taking specific decision such as drop packet or send it to control plan in case of lack of information. Control plan or brain of device configures interface, IP subnet and routing protocol. Further, it collects and keep the information in queue, build route tables and calculate Spanning-Tree Protocol and send a copy of table to data plan, which it can further deal with arrival of packet without involving control plan later. Administrators configure the Control plan through the management plan using numerous CLI commands.

Complicated architecture of traditional devices, where data plane and control plan are vertically tightly coupled to each other causes increased complexity, difficulty to manage architecture, shortage of entire view of network, distress of dynamic change, persistence of faults, reduction of the flexibility and hampering of the innovation, etc [4]. Figure 1 illustrates the traditional switch devices.

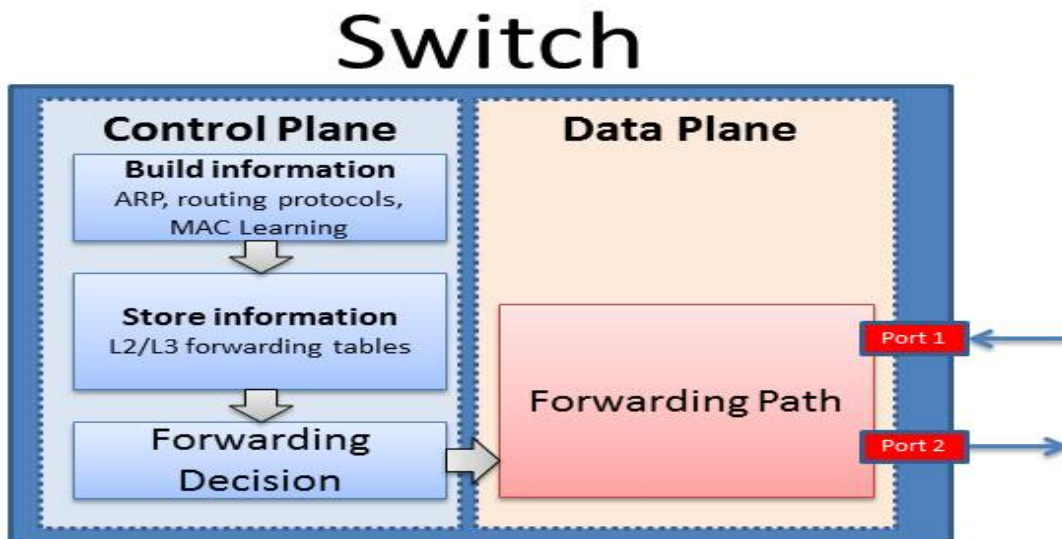


Fig. 1. Traditional network device architecture

III. ARCHITECTURE OF SOFTWARE DEFINED NETWORKING

As the matter of fact, traditional network is feeble to face requirement of today's enterprise network and demanding end users. Software Defined Networking (SDN) protrudes ahead in area of networking as solution for problems with traditional network. SDN is transforming networking architecture and providing unprecedented feature such as, programmability, flexibility, stability and automation by provide programmable controller.

Idea behind of SDN is to simplify the complex network devices by decoupling control plan from data plan, where end devices become simple forwarding devices without complexity. This is possible by putting in the network intelligence (control plan) and logically centralized controllers separate. Figure 2 illustrate the SDN architecture. [5]

Organized by

Dept. of Computer Science, Garden City University, Bangalore-560049, India

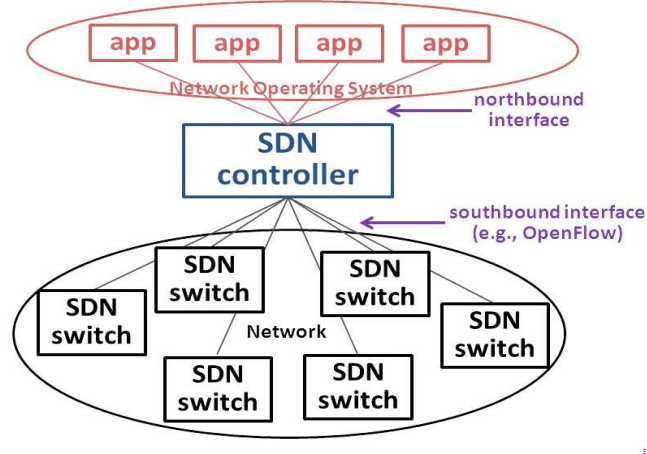


Fig. 2. Software Defined Networking high

The brain (control plan) resides in between application and data plan. It exploits the Application Interfaces (APIs) to control and deal over the data and management plan, the most well-know example of this southbound API is Open Flow [6],[7] was developed by ONF (Open Network Foundation) [8]. Southbound interface provides communication between specific network components with low-level components such as end nodes. It enables the controller to interact with switches and other nodes to identify network topology, determine network flows and implement request sent to it via northbound interfaces.

Contradictory to earlier scenario, northbound interface provides communication among the higher-level components. The services and applications running over the network set up communication via northbound interface through controller. API between controller and application provide a platform that allow to business application operate without being tied to the details of their implementation.

SDN provide an abstraction for upper layer where, researcher and developer need not to interfere themselves with underlying infrastructures of network element. It is dynamic network architecture that protects existing investments while future-proofing the network. SDN also makes it possible to manage the entire network through intelligent orchestration and provisioning systems.

There are various number of Controllers with different platform are flourishing in recent years. For example NOX[9], POX[10], Floodlight[11], ONOS[12], Ryu[13] and so on with different platform and architecture, open source and commercial, different vendors and based on different language. In this paper we especially chose Open Daylight and implemented scenario over that. Following section describes the ODL architecture.

IV. OPENDAYLIGHT CONTROLLER ARCHITECTURE

OpenDayLight is an open source and java based SDN controller which is managed by Linux Foundation [14] and supported by more than 40 companies such as IBM,Cisco, Juniper, VMWare and number of other major networking vendors. It is known as most well-documented controller .It can be extended on any hardware and operating system platform that supports Java. ODL is based on OSGI (the Open Services Gateway Initiative) architecture [15]. OSGI is known as the Dynamic Module System for Java, defines a standard for modular application development.

ODL contain several component and projects and include of three layers. In top layer business and network logic applications reside that use the controller to gather network intelligence, run algorithms to control and monitor network behavior. The core is the framework layer where controller exists. It stead north-bound and southbound APIs and unlimber the application requirement services from the network devices. Controller provides an abstraction which allows to developer to focus on the development of application functionality rather than involving with writing device-

An International Conference on Recent Trends in IT Innovations - Tec'afe 2017

Organized by

Dept. of Computer Science, Garden City University, Bangalore-560049, India

specific drivers. And the bottom layer consists of physical and forwarded devices. Figure 3 show architecture mainly comprises three layers. As figure 3 shows, middle platform include of basic network service function which we will refer to some of them in this paper during implementation.

Topology manager is a service for learning the network layout and provide the service to those applications that are looking for network view. Switch manager alongside Topology management are responsible for storing node discovered on the physical layer. Forwarding along Topology manager and switch manager provide services for registering and preserve the network flow state.

ODL supports multiple protocols in southbound interface such as OpenFlow 1.0, OpenFlow 1.3, BGP-LS, LISP, SNMP[16], etc. ODL is created with an objective of reducing vendor; locking therefore it supports protocols other than Open Flow. The southbound interface is capable of supporting multiple protocols such as OpenFlow and BGP-LS as separate plug-in. The SAL determines how to fulfill the requested service irrespective of the underlying protocol used between the controller and the network devices.

V. IMPLIMENTATION OF SCENERIO VIA ODL

OpenDaylight is a very large platform, with lots of plugins and features. Because of this, it may create some confusion and initial hype of being complex to the newcomers. Thus, let's keep things simple and learn step by step procedure of implementation.

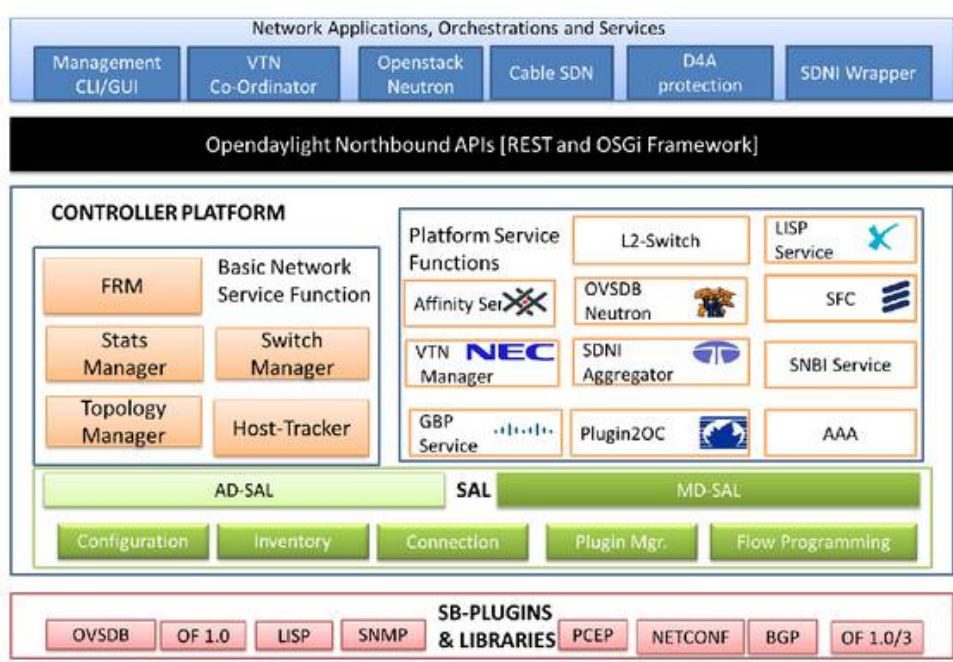


Fig. 3. ODL detailed architecture

Step 1: Installation overview

The easiest way to download and install ODL-Beryllium is through its Karaf distribution. We can obtain the Karaf pre-built package either as a .zip file or as a .tar file —same content— from OpenDaylight’s downloads page. i.e <https://www.opendaylight.org/downloads>



An International Conference on Recent Trends in IT Innovations - Tec'afe 2017

Organized by

Dept. of Computer Science, Garden City University, Bangalore-560049, India

We already have our new Ubuntu server running, with a NAT access to the Internet. In my case I'm running *test@ubuntu*, with *eth0* as network interface, and an IP address of 192.168.42.137 via DHCP. Let's manually configure this interface to have a **static ip**:

Assigning IP to both interface public and private for Ubuntu

Interface.

eth0 → Nat, public

eth1 → Private

This command can be used to assign the IP to specific interface through DHCP service.

```
test@ubuntu:~$ sudo dhclient interface
```

Step 2: OpenJDK project is an open-source implementation of the Java SE Platform. This is the default version of Java that is provided from a supported Ubuntu repository. Currently, there are two versions available, *openjdk-6* and *openjdk-7*

The next step is to install the Java Runtime Environment (JRE). ODL-Be runs on Java 7 or higher JVM versions.

```
test@ubuntu:~$ sudo apt-get install openjdk-7-jre
```

Step 3: At last, we need to define the *JAVA_HOME* environment variable —Karaf will need it. By default Java is located in *"/usr/lib/jvm/java-<version>"* directory. To persist the *JAVA_HOME* variable we must edit our ".profile" file and append a *JAVA_HOME* definition:

```
test@ubuntu:~$ nano .profile
```

add below command to end of this file.

```
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-amd64 → for 64bit OS
```

```
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-i386 → for 32bit OS
```

Step 4: Download the OpenDaylight Beryllium

```
test@USERVER:~$ wget \
```

```
https://nexus.opendaylight.org/content/groups/public/org.opendaylight/integration/distribution-karaf/0.4.0-Beryllium/distribution-karaf-0.4.0-Beryllium.tar.gz
```

Step 5: unzip

Unpack the tarball on same *<HOME>* directory —or wherever we want *ODL-Be* to stay:

```
test@ubuntu:~$ tar -xvzf distribution-karaf-0.4.0-Beryllium.tar.gz
```

Step 6: Run ODL controller

First run OpenDaylight and start the Karaf container in regular mode, by doing

```
test@ubuntu:~$ ./distribution-karaf-0.4.0-Beryllium/bin/karaf
```

Figure 4 shows that OpenDayLight is now running and is ready to deploy all the features we want in our scenario. So let's install these features on *ODL-Be*. The figure gives confirmation that successfully OpenDayLight has been

An International Conference on Recent Trends in IT Innovations - Tec'afe 2017

Organized by

Dept. of Computer Science, Garden City University, Bangalore-560049, India

implemented and we are ready to implement the scenario of our own choice over ODL. There are multiple features of ODL that we need to look into because it provides wide variety of choice.

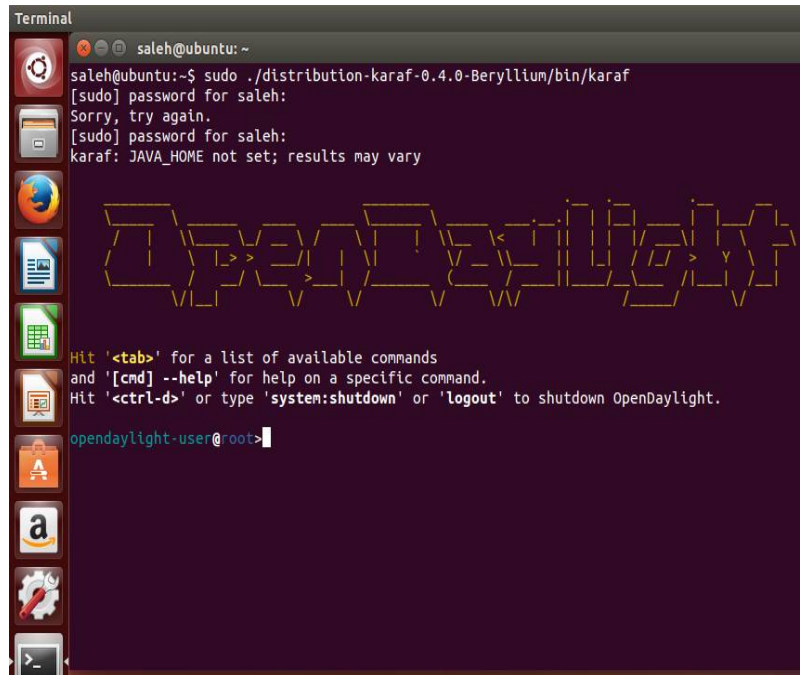


Fig. 4. Running ODL controller

The features we're going to use in our simple example:

- The **OpenFlow southbound** plugin to talk OpenFlow protocol —version 1.3.x— to our OpenFlow-enabled switches —we'll take care of these switches later. Other *southbound* plugins are available too, but we won't use them in this example.
- The Service Abstraction Layer (**SAL**), which is the core layer, will act to us as an abstraction layer for all the *southbound* plugins —only OpenFlow in our case.
- The basic-network-functions bundle, which covers things such as handling packets and flows, being able to discover switches and managing them, tracking hosts, discover addresses and so on, what's been named the **L2switch** bundle.
- And finally we'll install the **DLUX** bundle, which is a web-based GUI application, in order to graphically see the topology of our network, its switches, hosts, flows... This application has dependencies on the REST *northbound* API among others.

```
opendaylight-user@root>feature:install odl-l2switch-switch  
opendaylight-user@root>feature:install odl-dlux-all
```

Step 7: Open ODL through Firefox

ODL provide web GUI facility. You may to insert following address to run it.

<http://192.168.42.137:8181/index.html#/topology>

192.168.42.137 (eht0) is the IP of Linux public interface which ODL could run on it.



An International Conference on Recent Trends in IT Innovations - Tec'afe 2017

Organized by

Dept. of Computer Science, Garden City University, Bangalore-560049, India

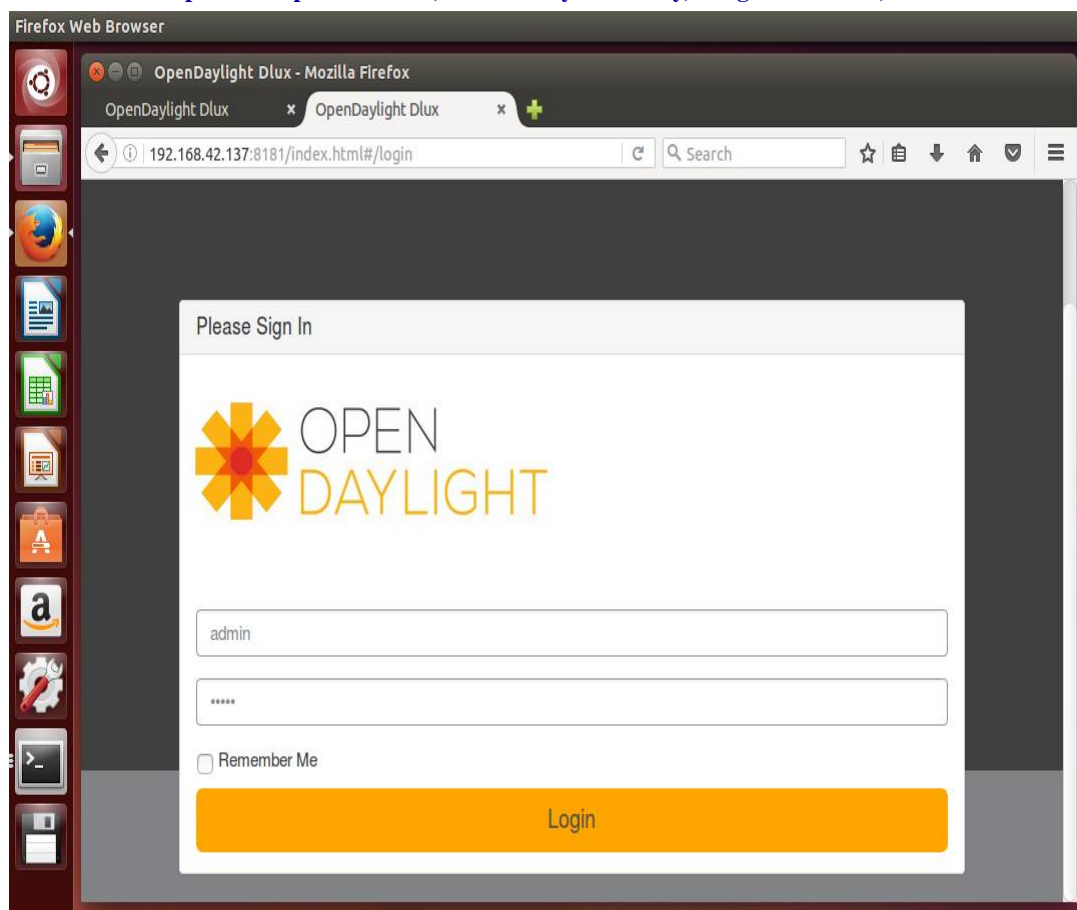


Fig. 5. Using ODL web GUI option

Enter Admin as username and password to could login as shown in Fig. 5. And now we are going to implement our scenario with the help of Mininet.

Step 8: The next step is to set up a simple OpenFlow-enabled **network** and link it to our brand new *ODL-Be* controller. An easy way to do it is by using OpenvSwitch, which is a free and open-source OpenFlow-enabled virtual switch that we can easily install in our machines. But an even easier way is by using Mininet, a fantastic tool that allows deploying into a single machine an entire virtual network of interconnected OpenvSwitch switches and virtual hosts.

So after download **Mininet VM image** from its web page, changed the network adapter setting to 'NAT', and started it up. The default login and password is mininet. Inside Mininet VM, to set up a network topology of four connected OpenFlow1.3-enabled switches —with a single host on each switch— and link it to our *ODL-Be* SDN controller, we'll run the command and check Fig 6:

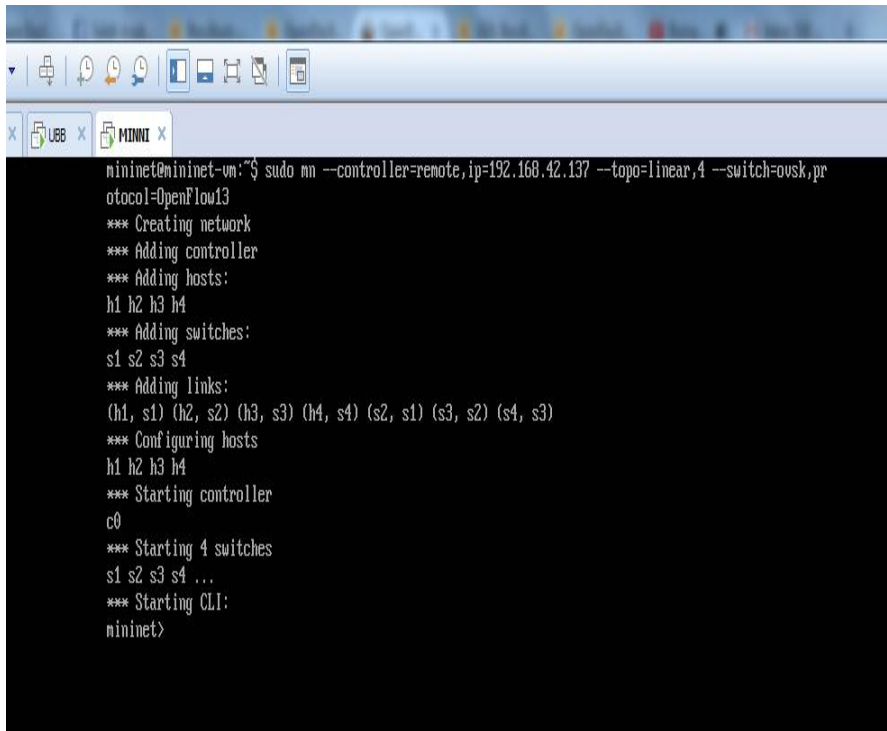
MININET

```
sudo mn --controller=remote,ip=192.168.42.137 --topo=linear,4 --switch=ovsk,protocols=OpenFlow13
```

An International Conference on Recent Trends in IT Innovations - Tec'afe 2017

Organized by

Dept. of Computer Science, Garden City University, Bangalore-560049, India



```
nininet@mininet-vm:~$ sudo mn --controller=remote,ip=192.168.42.137 --topo=linear,4 --switch=ovsk,pr
otocol=OpenFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet>
```

Fig. 6. Running the network scenario over Mininet

We proceed to log in and we'll indeed see our SDN network topology as shown in Fig. 7.

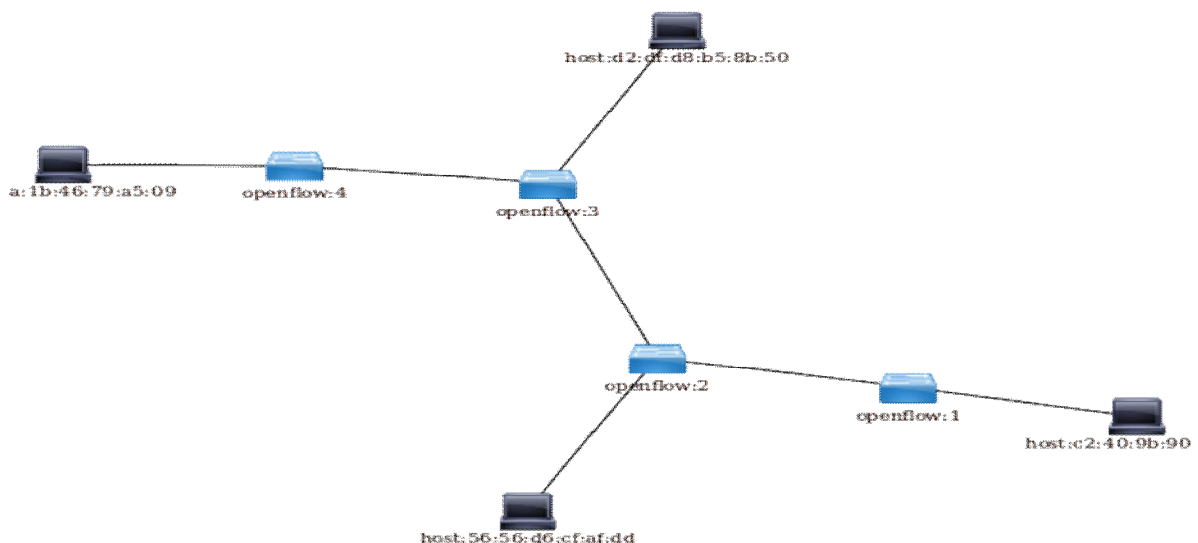


Fig. 7. Observing implemented scenario in ODL web GUI option



An International Conference on Recent Trends in IT Innovations - Tec'afe 2017

Organized by

Dept. of Computer Science, Garden City University, Bangalore-560049, India

Finally we get it running by getting the screen shown in Figure 8. You may celebrate for successful implementation of controller of ODL since its ready to implement any scenario of user's choice.

Important information: ODL is Linux operation system platform. Beryllium-SR4 is a version of ODL released in 26 October 2016 with verity of feature like Authentication, BGP, BMP, DIDM, Centinel , L2 Switch, NetIDE, Time Series Data Repository (TSDR) and etc.

```
View VM Tabs Help [Icons] Home x [Close]
To direct input to this virtual machine.

s1 s2 s3 s4 ...
*** Starting CLI:
mininet> h1 ping 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=0.948 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.331 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.488 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.357 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.362 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.329 ms
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5003ms
rtt min/avg/max/mdev = 0.329/0.469/0.948/0.221 ms
mininet> h1 ping h4
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=0.893 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.293 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=1.16 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.336 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.336 ms
^C
--- 10.0.0.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4002ms
rtt min/avg/max/mdev = 0.293/0.605/1.168/0.358 ms
mininet> h3 ping 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=0.960 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.522 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.357 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.363 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 0.357/0.550/0.960/0.246 ms

mininet>
```

Fig. 8. Connectivity all the host

VI. CONCLUSION

Open Day Light (ODL) is one of the stellar framework for implementation of SDN that has been addressed in detail in this paper. Through this paper we have provided the detailed step by step implementation of SDN based architecture on which if successfully implemented as per the steps stated in section V, researchers can develop their desired topological scenario from simple to complex and make further performance evaluation. This paper is an effort to practically bring out the solution to address the issues of researcher who have thorough concept of SDN but, is not able to implement it practically, for them, this paper will push them to leap ahead and step towards implementation of experimentation further. This paper will open doors for the researchers who are willing to work upon security, scalability, efficiency, congestion, speed, configuration, protocol and such many domains by implementing their experiments on provided framework of ODL as stated in this paper.

REFERENCES

- [1] N. Mckeown, "How SDN will Shape Networking," October 2011. [Online]. Available: <http://www.youtube.com/watch?v=c9-K5OqYgA>
- [2] S. Schenker, "The Future of Networking, and the Past of Protocols," October 2011. [Online]. Available: <http://www.youtube.com/watch?v=YHeyuD89n1Y>
- [3] Zuhran Khan Khattak , Muhammad Awais , Adnan Iqbal "Performance Evaluation of OpenDaylight SDNController," in Proceeding of the 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), 16-19 Dec. 2014. DOI: 10.1109/PADSW.2014.7097868
- [4] T. Benson, A. Akella, and D. Maltz, "Unraveling the complexity of network management," in Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, ser. NSDI'09, Berkeley, CA, USA, 2009, pp. 335–348.
- [5] M.K. Shin, K.H. Nam, H. J. Kim, "Software-Defined Networking (SDN): A Reference Architecture and Open APIs," In Proceedings of International Conference on ICT Convergence (ICTC), pp.360–361, Oct. 2012.
- [6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, Mar. 2008.
- [7] ONF, "Open networking foundation," 2014. [Online]. Available: <https://www.opennetworking.org>



ISSN(Online) : 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering
An ISO 3297: 2007 Certified Organization *Vol.5, Special Issue 2, April 2017*

An International Conference on Recent Trends in IT Innovations - Tec'afe 2017

Organized by

Dept. of Computer Science, Garden City University, Bangalore-560049, India

- [8] ONF, "OpenFlow management and configuration protocol (OF-Config 1.1.1)," March 2014. [Online]. Available: <https://www.opennetworking.org/images/stories/downloads/sdnresources/onf-specifications/openflow-config/of-config-1-1-1.pdf>
- [9] NOX detailed implementation, available online: <http://www.noxrepo.org>
- [10] POX detailed implementation, available online: <http://www.noxrepo.org>
- [11] Floodlight detailed implementation, available online: <http://floodlight.openflowhub.org>
- [12] Ola Salman, Imad H. Elhadj, Ayman Kayssi, Ali Chehab "SDN Controllers: A Comparative Study". Mediterranean Electrotechnical Conference MELECON 2016, Limassol, Cyprus, 2016.
- [13] Ryu detailed implementation, available online: http://ryu.readthedocs.io/en/latest/getting_started.html#what-s-ryu
- [14] Linux Foundation, "Open platform for NFV," <https://www.opnfv.org>, Sep 2014
- [15] *OSGi Core Release 5*, OSGi Alliance, San Ramon, CA, USA, Mar. 2012.
- [16] D. Harrington, R. Presuhn, and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks," Internet Engineering Task Force, dec 2002. [Online]. Available: <http://www.ietf.org/rfc/rfc3411.txt>