SDN-based Network Management Solution

Roop Kumar Sriramulu, Sachin Siddanuru Prabhulingappa, Suhas Janardhan, Swapnasheel Sonkamble
Computer Engineering, San Jose State University
San Jose, California, United States
roopkumar.sriramulu@sjsu.edu, sachin.siddanuruprabhulingappa@sjsu.edu, suhas.janardhan@sjsu.edu, swapnasheeldattu.sonkamble@sjsu.edu

Abstract—**In this project, we have developed a Network management application to monitor & control an enterprise network comprising of OpenFlow Enabled switches and SNMP enabled devices. We are running this application on utilizing the REST interface spunned by the controller centralized Software Defined Networking controller. The SDN Controller we are using is an open-source controller named OpenDaylight(ODL). We have used the SNMP and OpenFlow Southbound Plugins to communicate to the networking devices. For testing the application, we have simulated a network in GNS3 with a combination of legacy devices and OpenFlow switches**.

***Index Terms* — SDN, SNMP, NMS, REST, ODL, GNS3, OpenFlow, Southbound plugin.**

I. INTRODUCTION

Software Defined Networking is the emerging technology that gives hope to overcome the drawbacks of the current network infrastructures. With this technology, the networks control logic (control plane) is separated from the logic used to forward the traffic to the underlying devices (data plane). This separation makes the underlying switches and routers as simple forwarding devices and the control logic is implemented as a centralized controller which simplifies the configuration and evolution. By means of a well-defined programming interface between the switches and the SDN controller, the separation of the control plane and the data plane can be realized.

Management of network is a very important part of the network, which includes operation, maintenance, and provisioning of network devices. The network has been growing rapidly. Hence there will be new security problems, efficiency related issues at large datacenters or enterprises. Therefore, a good Network Management System (NMS) is required to ensure the accurate network view and optimize the network.

In this paper, we have proposed to use the SDN OpenDaylight controller to manage the network using the SNMP plugins and other plugins given by the controller. We retrieve the network statistics, link statistics and the information with respect to the underlying devices such as switches and routers to control and improve the performance of the network.

II. MOTIVATION

Network management system (NMS) is very important to ensure the efficient utilization and accurate working of the network devices in a network. Traditional NMS usually uses only SNMP as management protocol and has had great success. However, with emerging rapid growth of the network which exposes the network devices to various new threats and inefficient utilization of the network devices. Software-Defined Network (SDN) is a new technology which separates the data plane from the control plane as opposed to the distributed control of current networks. This centralized control plane concept can simplify the complexity of network management. Although SDN has many advantages, it still lacks good network management solution and is difficult to realize full deployment, which makes network operators to completely replace their legacy network management solution. As a result, it is necessary to combine traditional network management solution with SDN.

III. OPENDAYLIGHT CONTROLLER

Opendaylight controller is a highly modular, available, scalable, extensible and multi-protocol controller infrastructure which is used for SDN framework deployment on multi-vendor networks. It is a Java Virtual Machine which can run on any hardware and OS which supports Java JVM 1.7+. Model-driven service abstraction layer provides the required abstractions to support the south bound protocols through plugins. Applications provide Northbound APIs through:

OpenDaylight provides open SDN platform and offers flexibility for enabling the delivery of automated service which includes the following:

- Model-Driven Service abstraction layer(MD-SAL): This uses YANG models as an industry standard to map underlying devices to network applications to readily support technologies and hardware in the existing network.
- Modular, plug-in southbound interface approach where standard network management is supported. Along with these other devices and OpenFlow are also supported.
- Inter-based Northbound interface which abstracts the underlying infrastructure details and SDN capabilities are exposed to diverse network

applications.

- Multi-tenancy is provided in a multi-service environment by Network virtualization along flexible policy mechanisms.

It is used for those applications which run in same address s pace as controller.

A. *OSGi services:*

OSGi is a JAVA framework which provides modular and versioning and extensible nature of controller and life-cycle management for OSGi modules and services. It helps in loading plugins at run time into the controller.

B. *Bidirectional RESTful web services*

Applications which are not in the same address space as con troller uses REST services**.**

Below is the framework for the open daylight controller

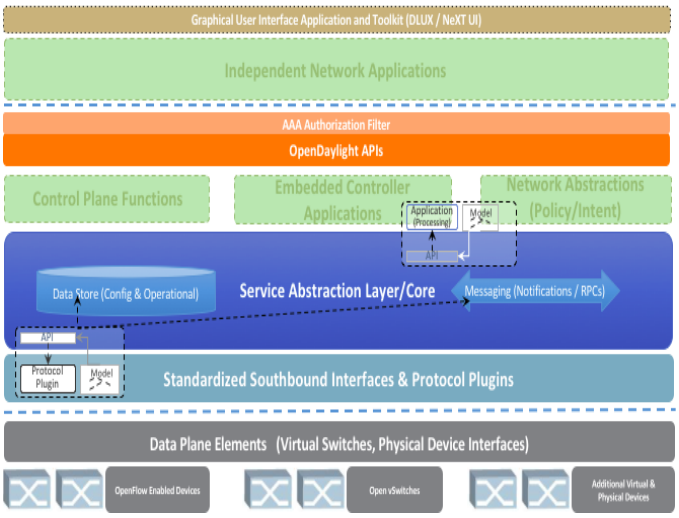

Fig. 1. Framework for OpenDaylight Controller

There are many protocols supported in the southbound which are dynamically linked to SAL(Service Abstraction Layer). SAL provides requested service irrespective of the underlying protocol used between network device and controller.

Below are the details of main components of the OpenDaylight controller.

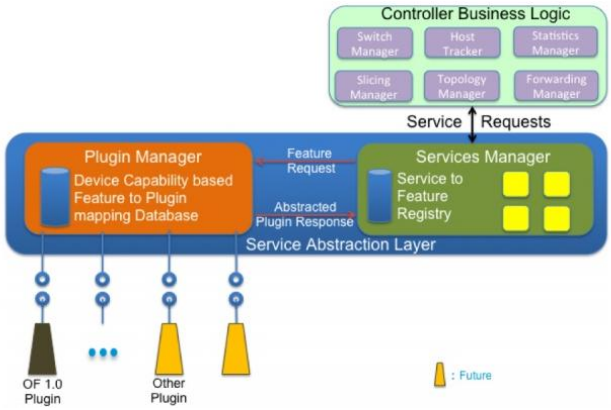

Fig. 2. Components of OpenDaylight Controller

A. *Service Abstraction Layer:*

SAL helps in separating northbound applications and southbound protocol plugins.

SAL acts as central part of the modular design of the controller which supports multiple southbound protocols. Topology manager uses services like Device Discovery provided by SAL to build the topology and device capabilities. Plugins provide features which help in building the services.

Control business logic, as shown above, sends services requests to SAL, service manager maps them to appropriate plugin through plugin manager and interacts with the network device using the most appropriate plugins.

B. *Model-Driven Service Abstraction Layer:*

A new architecture was built for Service Abstraction Layer which is model driven(MD-SAL).

Plugins of the controller can be either service/data consumer or service/data providers. Provider produces there services and data through its APIs. The consumer consumes the services provided by providers.

For example, OpenFlow plugin is used to add, modify and delete the flows from switches. Forwarding rule manager who is one of the consumers of OpenFlow plugin services provides a high level of flow programming services to clients.

This provides an opportunity to combine both northbound and southbound APIs. YANG has used a modeling language for data and service abstractions. It provides:

- Modeling single system components.
- Defining relationships and semantic elements.
- Modeling the functionality provided by controller components and structure of XML data.

C. *Karaf:*

Applications and components can be deployed onto lightweight container provided by OSGi run time component called Apache Karaf.

Some of the features of Karaf are Dynamic Configuration, Hot Deployment, Provisioning, Native OS integration, Logging system, Remote access, Managing instances.

Once the OpenDaylight controller package is downloaded it comes with all the features and one has to install the features manually. Karaf helps in installing the features.

IV. PLUGINS

If you are using *Word*, use either the Microsoft Equation Editor or the *MathType* add-on (http://www.mathtype.com) for equations in your paper (Insert | Object | Create New | Microsoft Equation *or* MathType Equation). “Float over text” should *not* be selected.

A. *OpenFlow Plugin*

OpenFlow interface enables interaction between forwarding and control layers of SDN architecture. It helps in OpenFlow implementation. It supports OpenFlow 1.0 and OpenFlow 1.3x. It is based on Model Driven Service Abstraction Layer

OpenFlow 1.3 plugin supports functionality like error handling, Message handling, State management, Mapping

function, connection handling, Event handling and propagation to upper layers.

OpenFlow plugin functionality can be divided into four areas as shown below:

- *Device Manager*: Low-level interactions with the switch is provided by the Device manager. It helps in tracking and performing response dispatch to the requesting entity.
- *Connection Manager*: Determines switch features and identity and provides early negotiation.
- *Statistics Manager*: It helps in maintaining synchronization between counters and on-switch.
- *RPC manager*: It helps in routing the MD-SAL requests towards the device.

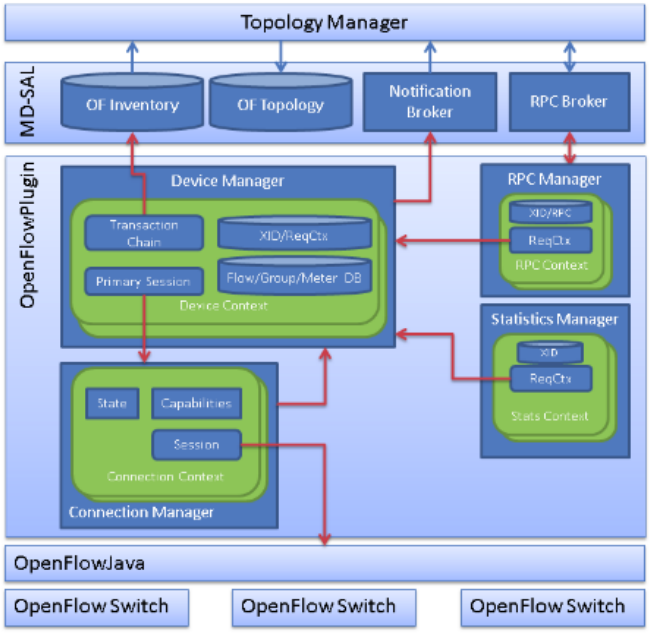

Fig. 3. OpenFlow Plugin Functionality

B. *SNMP Plugin:*

SNMP is a simple network management protocol used for network management. It is used to gather information from switches, routers, printers etc. This plugin enables controller and application to interact with devices using SNMP. The southbound plugin helps application which acts as SNMP manager to interact with devices that act as an agent. SNMP plugin includes:

SNMPv1, SNMPv2, SNMPv3 support, simple API invoked as RPC, APIs which enables applications to perform following functions:

GetRequest, SetRequest, GetNextRequest, GetBulkRequest, Trap support, InformRequest.

SNMP uses MIB to interact with the network device. The addition of MIB includes getting .my and the .oid files corresponding to the MIB. MIB file is converted into YANG model before it is used with SNMP plugin. A tool called "smidump" helps in converting MIBs into YANG models.

SNMP plugin is installed by executing below command:
feature:install odl-SNMP-plugin

There two northbound APIs which we have used are SNMP GET and SNMP SET. They have the following fields Ip-address, oid, get-type, community. The http request URI using these northbound APIs are of the following format

http://localhost:8181/restconf/operation/snmp:snmp-get
http://localhost:8181/restconf/operation/snmp:snmp-set

The input formats with the URI to request the information using snmp get and to set the information using the snmp set are shown below in the code snippets.

C. *RestConf Plugin:*

In the OpendDaylight controller, Restconf is packaged into Karaf bundles/features. Once we download the ODL package, one has to install it manually.

```
url = 'http://localhost:8181/restconf/operations/snmp:snmp-get'
Param = {
        "input":
                {
                    "ip-address": str(ip),
                    "oid" : str(oid),
                    "get-type" : get,
                    "community" : community
                }
        }
resp, content = h.request(
    uri = url,
    method = 'POST',
    headers={'Content-Type':'application/json'},
    body=json.dumps(Param)
    )
```

Fig. 4. http request using snmp-get API call

```
url = 'http://localhost:8181/restconf/operations/snmp:snmp-set'
Param = {
        "input":
                {
                    "ip-address": str(ip),
                    "oid": str(oid),
                    "community": community,
                "value": value
                    }
        }
resp, content = h.request(
    uri = url,
    method = 'POST',
    headers={'Content-Type':'application/json'},
    body=json.dumps(Param)
    )
```

Fig. 5. http request using snmp-set API call

/restconf

The API resource contains the access points and states for the RESTCONF features. It can be accessed through URI-"/restconf". Restconf listens on port 8181 for HTTP requests. Restconf enables datastores in the controller. There are two datastores:

- *config* - contains data inserted via controller
- *operational* - contains other data

/restconf/operational

This represents the operational datastore. It contains statistics data and operational data resources that will be retrieved by a client. Client cannot create or delete it.

Request and response data can be in JSON or XML format. The media type of request data (input) is set via "Content-

Type" field in HTTP header. The media type of response data (output) is set in the "Accept" field in HTTP header.

V. APPLICATION METHODOLOGY

This Application is used to Monitor the network and get real-time data of the devices. The application is written using python making use of the following libraries,

httplib2, json, sys, os, crypt, getpass, OrderedDict

When the application is run, it authenticates the user and presents the operations to selected,

Get Topology, Get Link Statistics, Get System Statistics, Use SNMP-GET, Use SNMP-SET, Real-Time-Monitoring, Exit.

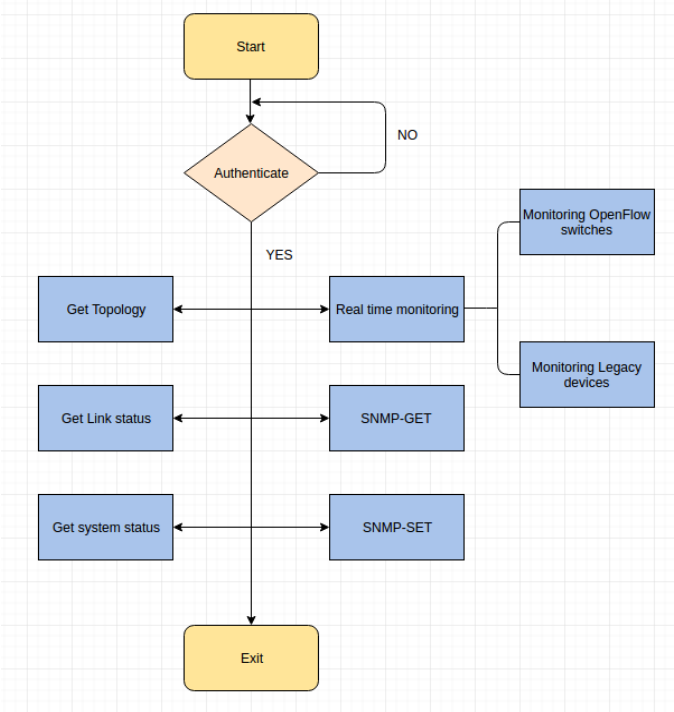

Fig. 6. Flow of Application

A. *Get Topology, Get Link Statistics:*

Plugins Used - RESTCONF, OpenFlow.

These operations will call the topology(), linkStats() methods which sends an HTTP request to the RESTCONF interface and fetches the data from the data-store of the OpenDaylight controller and sends a response with JSON data. This JSON content is parsed to get the topology information which in turn is presented to the user.

B. *Get System Statistics:*

Plugins Used - RESTCONF, SNMP.

This operation will get the IP address from the user and fetches the system statistics of the corresponding host in the network. This method which sends an HTTP request to the RESTCONF interface. An SNMP-GET request is sent to the corresponding device to extract data. This data is stored in the data-store of the ODL controller. An HTTP response is sent to the application with JSON data. This JSON content is parsed to get the System Statistics information like System Description, CPU usage, Memory Usage, Disk Usage.

C. *Use SNMP-GET:*

Plugins Used - RESTCONF, SNMP.

This operation will get the IP address, OID's, Get type and Community from the user and fetches the system statistics of the corresponding host in the network. This method sends an HTTP POST request to the RESTCONF interface. An SNMP-GET request is sent to the corresponding device to extract data. This data is stored in the data-store of the ODL controller. An HTTP response is sent to the application with JSON data. This JSON content is parsed to present the user with values for the requested OID.

D. *Use SNMP-SET*:

Plugins Used - RESTCONF, SNMP.

This operation will get the IP address, OID's, Community and value to be set for the corresponding host in the network. This method sends an HTTP POST request to the RESTCONF interface. An SNMP-SET request is sent to the corresponding device to change the data of the corresponding OID.

E. *Real-Time-Monitoring:*

In our application, we are doing real-time-monitoring for both legacy devices and OpenFlow devices.

- *Monitoring Legacy devices:*

This operation lets us know about communication and statistics information of legacy devices. There are two main functions called- snmp() and trap(). First one uses "tcpdump" and listens to port 161 to get all the communication data. Second one listens to port 162 and uses "tcpdump" to dump all the trap messages. These two methods by default stores the data in the form of pcap file which can be used for analyzing the packets.

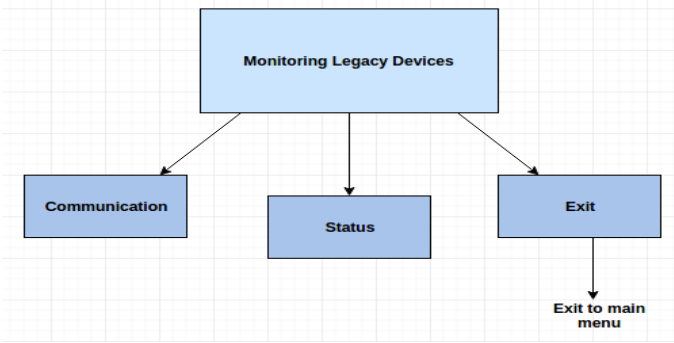

Fig. 7. Monitoring Legacy Devices

- *Monitoring OpenFlow devices:*

This operation happens by default when the application is run. This helps in monitoring OpenFlow switches. On running the application it gets the topology details which uses the method- topo(). It then executes the method checkTopo() to get the details of a number of nodes and links present in the topology. This method is run in forever loop to get the link and node details. trackLinksUp() and trackNodesUp() are used to find out any newly added links and nodes respectively. trackLinksDown() and trackNodesDown() are used to find out any deleted added links and nodes respectively.

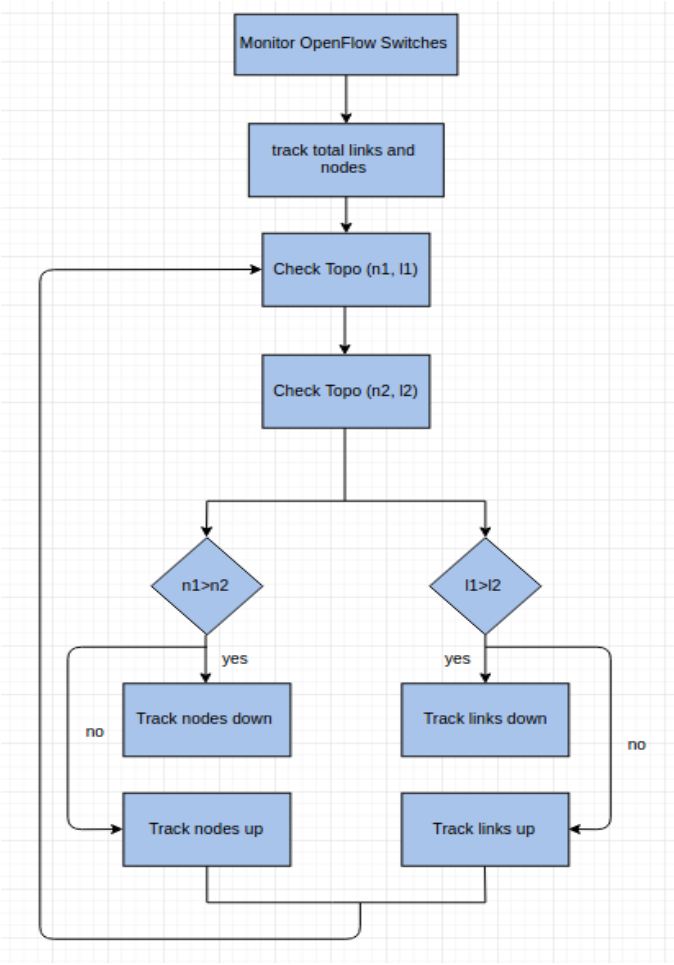

Fig. 8. Monitoring OpenFlow Devices

VI. Experiment and Results

For testing the design and implementation of our management system, we made use of OpenDaylights SNMP plugin.

Our infrastructure for applications consists of an Ubuntu VM running the OpenDayLight controller, hubs for connecting different nodes to the controller, one legacy Cisco c3600 router, two OpenVswitch management Docker containers, and five Ubuntu Docker containers. We make use of the internet Docker container to enable DHCP IP addressing scheme in the system and updating additional repositories along with SNMP daemons in the Ubuntu Dockers. The topology is as shown in the figure where OpenVswitch's eth0 interface acts as the management interface responsible for communicating with the controller via the hub.

Fig. 9. Topology Created

A. *Topology management*

- Our system can monitor and track information and changes from the devices in the topology. We tested our system in two scenarios - With GNS3 topology – We tested the system using GNS3 simulation software to create a testbed which could show us a topology view of the system in the controller using the 'odl-dlux' feature. The application runs on the testbed can retrieve device information like Link statistics, Node IDs, Port statistics, and system statuses like CPU, memory, and disk utilization. We can also set functionalities using the SNMP OIDs.

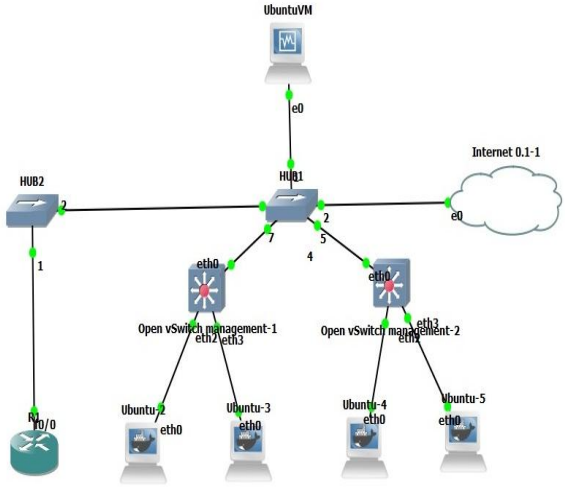

Fig. 9. Monitoring OpenFlow Devices

B. *Real-time monitoring*

- With the application, we can monitor real-time SNMP trap messages from the configured legacy devices. Using Mininet for simulation, we created a single switch two hosts topology and tracked real-time link status changes by toggling link up/ down status.

VII. Results

The screenshot below shows the output when the user requests for Topology and Link statistics data fetched from the controller data-store.

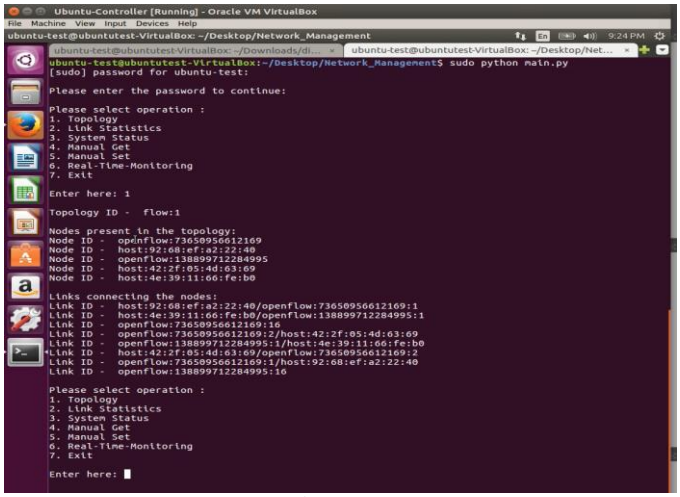

Fig. 10. User Requests for Topology and Link statistics

The screenshot below shows the output when the user requests for the system status of the host specified.

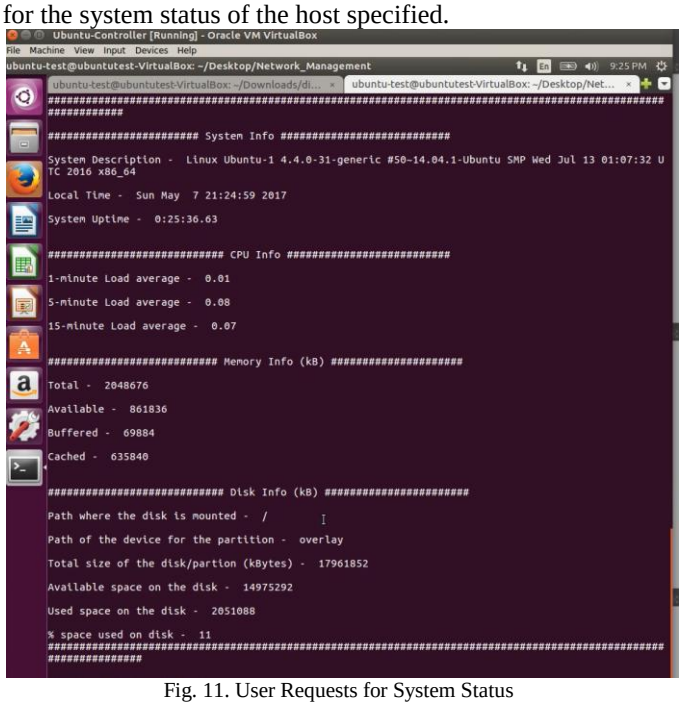

Fig. 11. User Requests for System Status

The screenshot below shows the output when the user wants to use SNMP GET functionality to get the host's data manually.

Fig. 12. SNMP GET output

The screenshot below shows the output when the user wants to use SNMP SET functionality to set any status value of the host's manually.

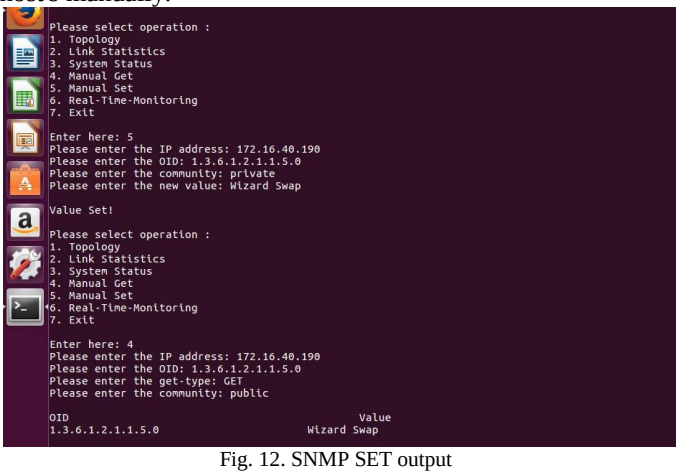

Fig. 12. SNMP SET output

The below screenshot shows the output of testing the real-time-monitoring monitoring functionality of Openflow enabled devices with Mininet topology.

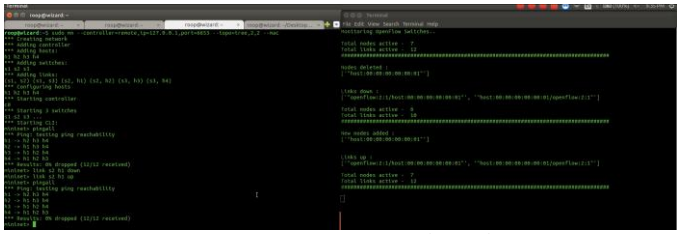

Fig. 13. Real Time Monitoring Output of Openflow devices

The below screenshot shows the output of testing the real-time-monitoring monitoring functionality of SNMP enabled devices with the GNS3 topology created.

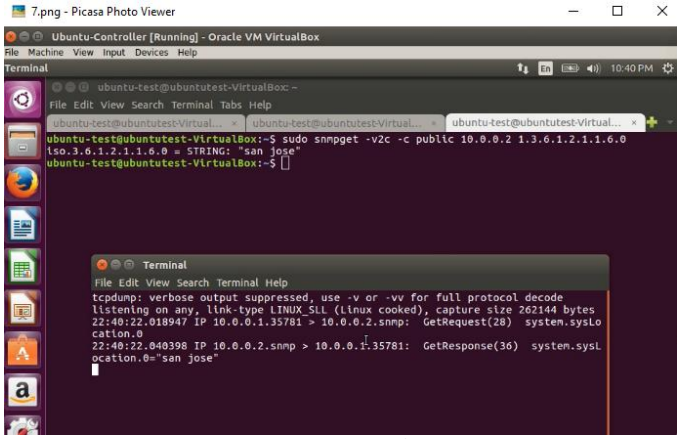

Fig. 14. Real Time Monitoring output of SNMP enabled devices

The screenshot below shows the output when the interface of the legacy device (R1) in the GNS3 topology is turned off and on.

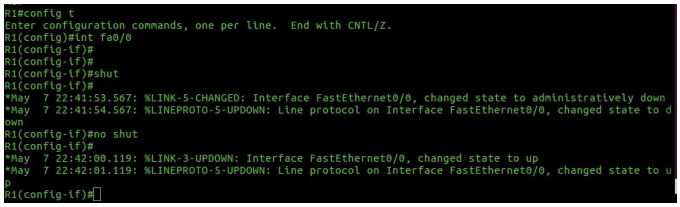

Fig. 15. Output when Legacy device is turned-off

The screenshot below shows the output in the real-time-monitoring functionality of the application when the process from the above screenshot is done.

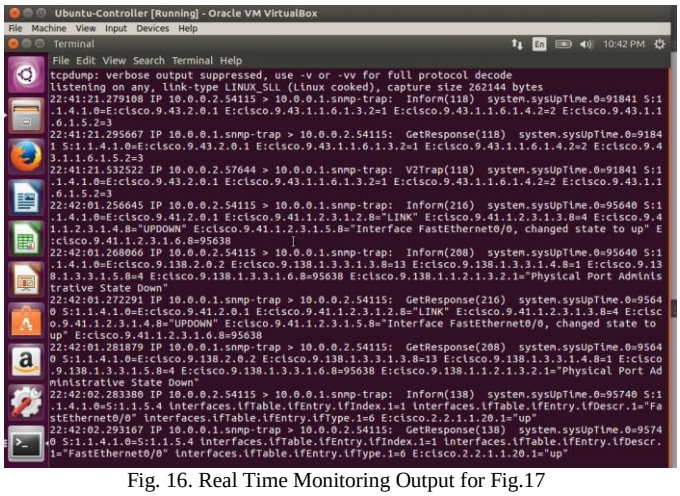

Fig. 16. Real Time Monitoring Output for Fig.17

VIII. CONCLUSION AND FUTURE WORK

This paper presents a use case for monitoring the network with the OpenDaylight controller using SNMP and OpenFlow southbound plugins. In the future, by including a functionality for adding and deleting flows in our application, we can manage the network infrastructure using an SDN controller. We can also extend this functionality to track realtime unsolicited protocol requests and train the system to learn possible attacks on the infrastructure. This can be compared to a predefined threshold and new flow rules and updates can be added to the nodes by the controller.

IX. APPENDIX

Controller Code Analysis:

In the application we have developed to fetch the information from the underlying devices or switches, we have used the http requests is nothing but the URI that contains the Restful APIs. We have used the following requests

- http://localhost:8181/restconf/operational/network-topology:network-topology
- http://127.0.0.1:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:(switch) /node-connector/openflow:(switch):link
- http://127.0.0.1:8181/restconf/operations/snmp:snmp-get
- http://127.0.0.1:8181/restconf/operations/snmp:snmp-set

The Restconf protocol supports OPTIONS, GET, PUT, POST and DELETE operations. We have used the POST method in our application to retrieve the data based on the module and rpc names we provide in the API which is of the format as given below.

POST /restconf/operations/<moduleName>:<rpcName>

This API invokes the RPC. The RPC is a request-reply message pair, when the consumer triggers the request, RPC sends the request to the provider. This also replies with the reply from the provider in future.
Here moduleName is the name of the module and rpcName is the name of RPC in this module.
The data sent to the RPC must have the argument input.
The result retrieved with the call of this API will have the root element output.

SNMP Get Method:

When the snmpGet method is called from any program, the SNMP pugin calls the built in snmpGet method. This method logs the information about the type, OID, ip address and the community from the input for which you need to see the information from the SNMP agent into the operational datastore provided by the Restconf. This method creates an object of AsyncGetHandler and calls the getRpcResponse method of that object.

The http request used here is
http://127.0.0.1:8181/restconf/operations/snmp:snmp-get

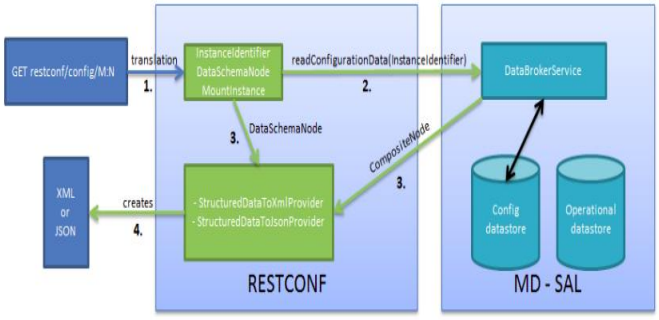

Fig. 6. SnmpGet in Action

The getRpcResponse method calls sendRequest method which in turns calls the snmp.send method with proper arguments to fetch the required information.

SNMP Set Method:

When the snmpSet method is called from any program, the SNMP pugin calls the built in snmpSet method. This method logs the information about the type, OID, ip address and the community from the input for which you need to see the information from the SNMP agent. This method creates an object of AsyncSetHandler unlike the snmpSet method creates an object of AsyncGetHandler and calls the getRpcResponse method of that object.

The http request used here is
http://127.0.0.1:8181/restconf/operations/snmp:snmp-get

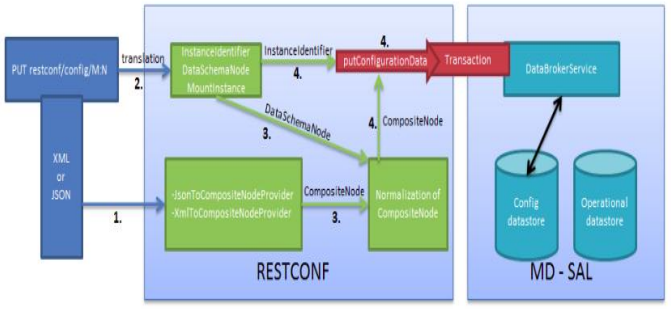

Fig. 7. SnmpSet in Action

The getRpcResponse method calls sendSnmpSet method which in turns calls the snmp4j.send method with proper arguments to set the required information.

X. REFERENCES

[1] http://www.networkmanagementsoftware.com/snmp-tutorial-part-2- rounding-out- the-basic/
[2] https://www.manageengine.com/products/oputils/enable-snmp-cisco-router.html
[3] http://mininet.org/walkthrough/
[4] http://mininet.org/api/annotated.html
[5] http://www.nojitter.com/post/240164599/monitoring-a-software-defined-network-part-1
[6] http://docwiki.cisco.com/wiki/Simple_Network_Management_Protocol
[7] https://wiki.opendaylight.org/view/SNMP_Plugin:Getting_Started
[8] https://linux.die.net/man/8/snmptrapd

[9] https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-an-snmp-daemon-and-client-on-ubuntu-14-04
[10] http://www.alvestrand.no/objectid/1.3.6.1.2.1.html
[11] https://support.atera.com/hc/en-us/articles/220109447-How-To-Monitor-Linux-Servers-Using-SNMP
[12] http://www.cisco.com/c/en/us/support/docs/ip/simple-network-management-protocol-snmp/15215-collect-cpu-util-snmp.html
[13] https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#creating
[14] https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin::End_to_End_Inventory
[15] http://docs.opendaylight.org/en/stable-boron/user-guide/
[16] Virtualized network infrastructure using OpenFlow
[17] Hideyuki Shimonishi; Shuji Ishii 2010 IEEE/IFIP *Network Operations and Management Symposium Workshops*, Year: 2010
[18] Yongyue Zhang; Xiangyang Gong; Yannan Hu; Wendong Wang; Xirong Que, 2015, *SDNMP: Enabling SDN management using traditional NMS*, IEEE International Conference on Communication Workshop (ICCW), Year: 2015