

CS 6301 –Software defined Networks

SPLIT THE BANDWIDTH USING QOS POLICING ON OPEN DAYLIGHT

Mentor: Tim Hawks

Team Members:

Sriharsha Ganja –sxc151830

Raghunandan N. R.–rxn150230

Abhinav Kumar Parakh- akp140030

Arnav Sharma- axs143930

Panchami Rudrakshi-pgr150030

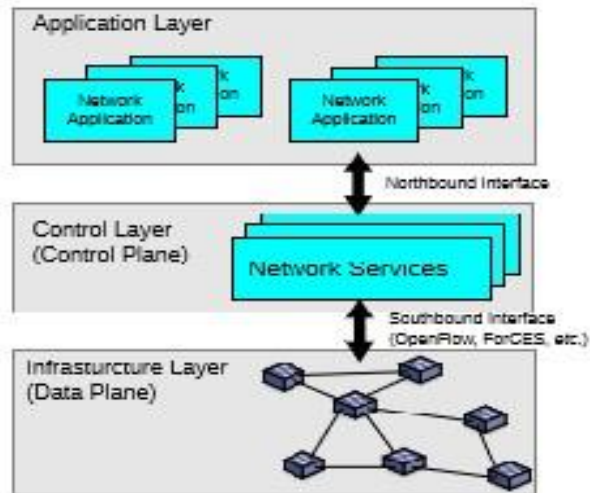
Mounica Reddy Bojja-mxb151730

Table of Contents

1. SDN Introduction.....	3
2. Problem Definition.....	4
3. Objective.....	4
4. Literature Survey.....	5
4.1 OpendayLight.....	5
4.2 Mininet.....	7
4.3 VirtualBox.....	9
5. Project Requirement Specification.....	11
5.1 Problem Analysis.....	11
5.2 Project Requirements.....	11
5.3 Project Functions.....	11
5.4 Operating Environment.....	12
5.5 System Requirement.....	13
6. System Design.....	14
6.1 Topology.....	14
6.2 Flowchart.....	14
7. Code.....	15
8. Results.....	20
9. Conclusion.....	21

1. SDN INTRODUCTION

Software-defined networking (SDN) is an approach to computer networking that allows network administrators to manage network services through abstraction of higher-level functionality. This is done by decoupling the system that makes decisions about where traffic is sent (the control plane) from the underlying systems that forward traffic to the selected destination (the data plane). The inventors and vendors of these systems claim that this simplifies networking.



SDN requires some method for the control plane to communicate with the data plane. One such mechanism, OpenFlow, is often misunderstood to be equivalent to SDN, but other mechanisms could also fit into the concept.

The above Architecture diagram shows logical Arrangement of components of a typical Software Defined Network containing three components(or layers), each layer plays significant role in the Accurate working of Network, Controller and Network applications deployed at each corresponding layer.

SDN consists of three layers:

- 1) Infrastructure Layer
- 2) Control layer
- 3) Application Layer

Infrastructure layer(data path) as shown above is the bottom layer , the layer where the actual network nodes are situated such as Switches, hosts etc. note here that the routers are not used here as the network devices(Switches) used here are dumb nodes with no intelligence in them apart from hardware and little software in it, the significant job the switches do here is forwarding the packets according to the controller directive.

Control layer is this layer which is heart of the SDN Network which takes actual decisions on how to deal with the packets at the data plane or Infrastructure layer based on the policy implementations by the network owner, here the actual controller (centralized Controller) is situated, it also acts as an interface between Application Layer at the top and Infrastructure layer at the bottom, The

centralized controller here maintains one central view over all the switches present in the network under its Control, There can be more than one Controller in a SDN which depends on the size of the network that is the no of nodes(switches and Hosts) in the network. Controller is able to see the abstract view of the underlying network and will be made available to the Business applications running on top of Controller.

Application layer where the actual Network applications run such as Business applications required specific to the particular business , The firewall application I have developed forms a business application hence is a part of this Application layer, this layer is also where the policies can be realized in the form of applications implementation using the APIs provided by the controller, here the controller am using is Open day light which allows to access the network a parameters in the form of APIs which in turn are mapped to underlying Network South bound protocol plugins via the controller so the controller also acts as an interface between south bound APIs and North bound Network Business Applications.

2. PROBLEM DEFINITION

- ⌘ Real time applications require stringent quality of service(QoS) guarantees.
- ⌘ Thus there is a need for the network programmers to design network protocols that can deliver performance guarantees.

Solution:

- ⌘ Firstly, we define a QoS Management and Orchestration architecture that allows us to manage the network in a modular way.
- ⌘ Secondly, we provide a seamless integration between the architecture and the standard SDN paradigm following the separation between the control and data planes.

3. OBJECTIVE:

- ⌘ Split the bandwidth according to the number of nodes connected.
- ⌘ The system has several virtual machines running , connected to the SDN controller.

Input to the controller: Sample of certain bandwidth.

- ⌘ **Process:** Controller senses the input and identifies the bandwidth. According to the number of nodes/switches connected to the controller, the bandwidth is split.

⌘ **Output:** Once the bandwidth reaches the nodes, it is tested using netperf.

4. LITERATURE SURVEY

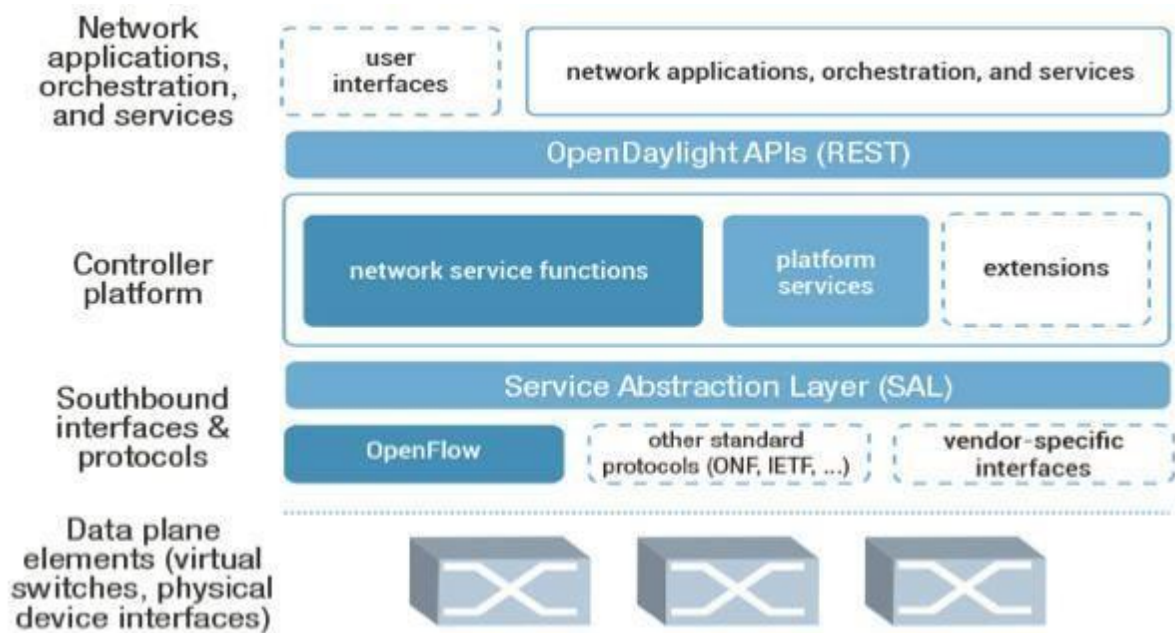
4.1 OPEN DAYLIGHT

Open daylight Software Controller is an open platform for network programmability to enable SDN and create a solid foundation for NFV for networks at any size and scale. It is a combination of various components such as controller, interfaces, protocol plugins, APIs and with some built-in applications in it. This controller also supports open networking foundations Open flow standard enabling a common open source framework and platform for SDN across the industry for customers, partners and developers, the software for this controller was developed in Java. On Feb. 2013 the Open daylight Project was started and the first version of it was released on Feb. 2014. Open daylight controller provides the APIs situated at the top layer of the architecture using which Applications in particular business applications can be developed as required. In the same way Firewall can be developed using these APIs Northbound APIs are REST APIs which can be accessed via JSON or XML objects. The above figure shows the arrangement of various layers in Open daylight controller platform, as we can see from the figure the network applications at the network application and orchestration services layer communicate with the switches at the data path by passing through the two layers below it those are controller platform and south bound interfaces, but the network application developers only need to know about the north bound REST APIs which are exported by the Controller platform layer (middle layer). As can be seen from the diagram the Openflow protocol is implemented at the layer3 that is as part of south protocol plugins, Open daylight provides provision for scalability for new network protocols to be developed and deployed at the layer 3.

4.1.1 On Generality of the Data Plane and Scalability of the Control Plane in SDN

This approach talks about the approaches to maintain the generality of the data plane and enhance the scalability of the control plane. The objective of this study is to focus on the problems of adding control functionalities to the data plane to fructify the quantifiably. This study is based on the following principles viz. principle 1 states that the control practicality should be reserved in the control plane, except for those that could promote the aging efficiency and adapt to the hardware and software requirement in the data path, so as to accord with the evolutionary trend of SDN. The principle 2 states that The control practicalities cannot change the basic process of the data plane, so as to ensure the generality of network devices in SDN. The last one that is principle states that Collecting statistics in the data plane cannot affect the accuracy and cogency, and cannot result in higher load in the control plane. This study focuses on eliminating redundant packet-in messages to the controller, although this concept is not related to this project but still it is covered under my literature survey, based on the above principles the experimental network was set up in the network laboratory and elimination of redundant packets at the control plane was added as new functionality at the data path. UDP flows were created and different flows were sent from one of the hosts at different speeds and CPU utilization of Open V-switch is measured and it is seen that in the software OpenFlow switch, CPU utilization of Open vSwitch without modifications maintained 12% around, and after

modifications, its CPU utilisation falls to about 4% showing that excess packets are eliminated thus reducing the overhead on the controller, the Cisco p-3290 series of switches were used for the experiment.



4.1.3 OpenFlow Tables

The OpenFlow specification describes a wide variety of topics. For instance, the protocol format that's used to communicate with an OpenFlow switch by a controller is defined – in much the same way as you'd see in an IETF draft. However, the agent that receives OpenFlow instructions must also operate a certain way, and the OpenFlow specification addresses these concepts as well. One such construct is OpenFlow Tables. You may have heard the term **pipeline** before. This is not an OpenFlow-centric term - the concept is often described in deep-dive sessions on internal switch architectures that explain the path a frame or packet takes through a switch. The myriad of systems used for internally processing a frame - such as regenerating a L2 header, identifying the next-hop destination, decrementing the TTL - all form a sort of “pipeline” that a frame travels through on its way out of the device.

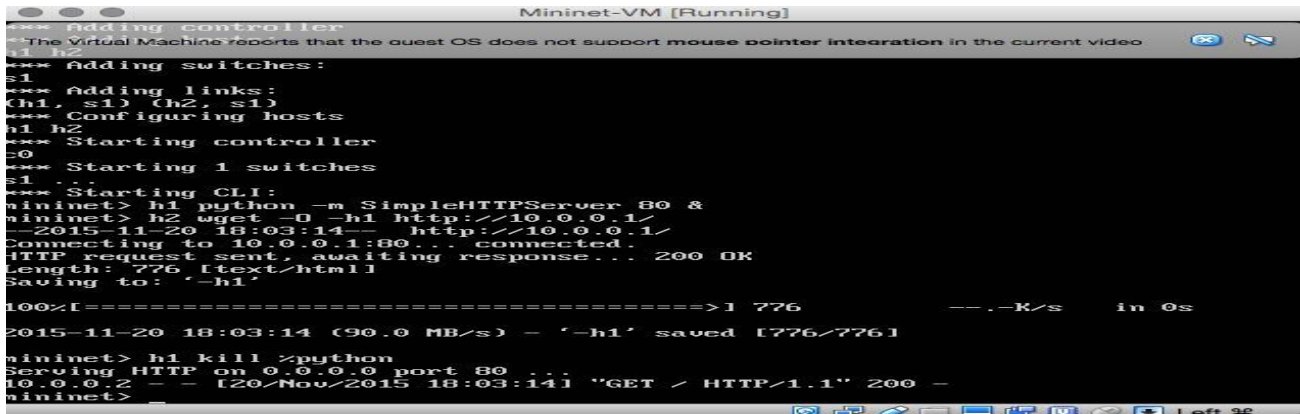
The OpenFlow pipeline is meant to emulate this process in a vendor-agnostic way. Just like a hardware pipeline in a switch would be composed of multiple “stops” where various tasks are performed, the OpenFlow pipeline is composed of multiple tables that accomplish similar purposes.

You could use one table for performing a port lookup, and another for making changes to the L3 header based off of its current information (NAT is a good example of this). Remember, OpenFlow is not a specific ASIC-level instruction-set, it is an abstraction that the switch creator must implement in a process that converts OpenFlow tables to hardware-specific instructions. It is up to the switch vendor to map the OpenFlow pipeline to specific hardware rules.

4.2 MININET

This study talks about the use of mininet for creation of software defined networks , its benefits and advantages of using emulation tool. this paper also about the use of mininet in combination with floodlight controller, this work also talks about other network simulators. There are three main constituents of a SDN viz. programmable nodes i.e switches, Controllers and the standard protocol as a medium and protocol for exchanging messages between them. this work talks about the commands usage in mininet using various commands , one such is the dpctl commands.

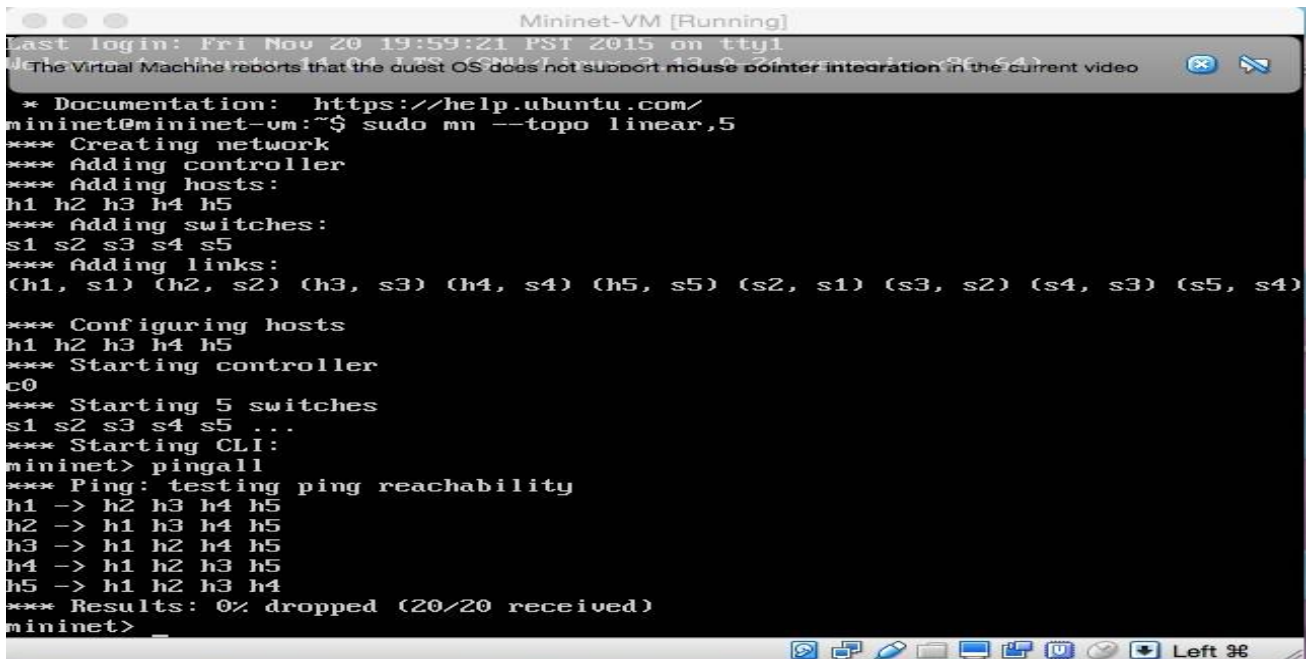
Step1: To run a simple web server and client in Mininet:



```
Mininet-VM [Running]
*** Adding controller
The Virtual Machine reports that the guest OS does not support mouse pointer integration in the current video
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet> h1 python -m SimpleHTTPServer 80 &
mininet> h2 wget -O - -h1 http://10.0.0.1/
--2015-11-20 18:03:14-- http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 776 [text/html]
Saving to: 'h1'
100%[=====] 776 --.-K/s in 0s
2015-11-20 18:03:14 (90.0 MB/s) - 'h1' saved [776/776]
mininet> h1 kill %python
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.2 - - [20/Nov/2015 18:03:14] "GET / HTTP/1.1" 200 -
mininet>
```

Step2: Creating built-in topologies:

Command used: `Sudomn -topo linear,5`

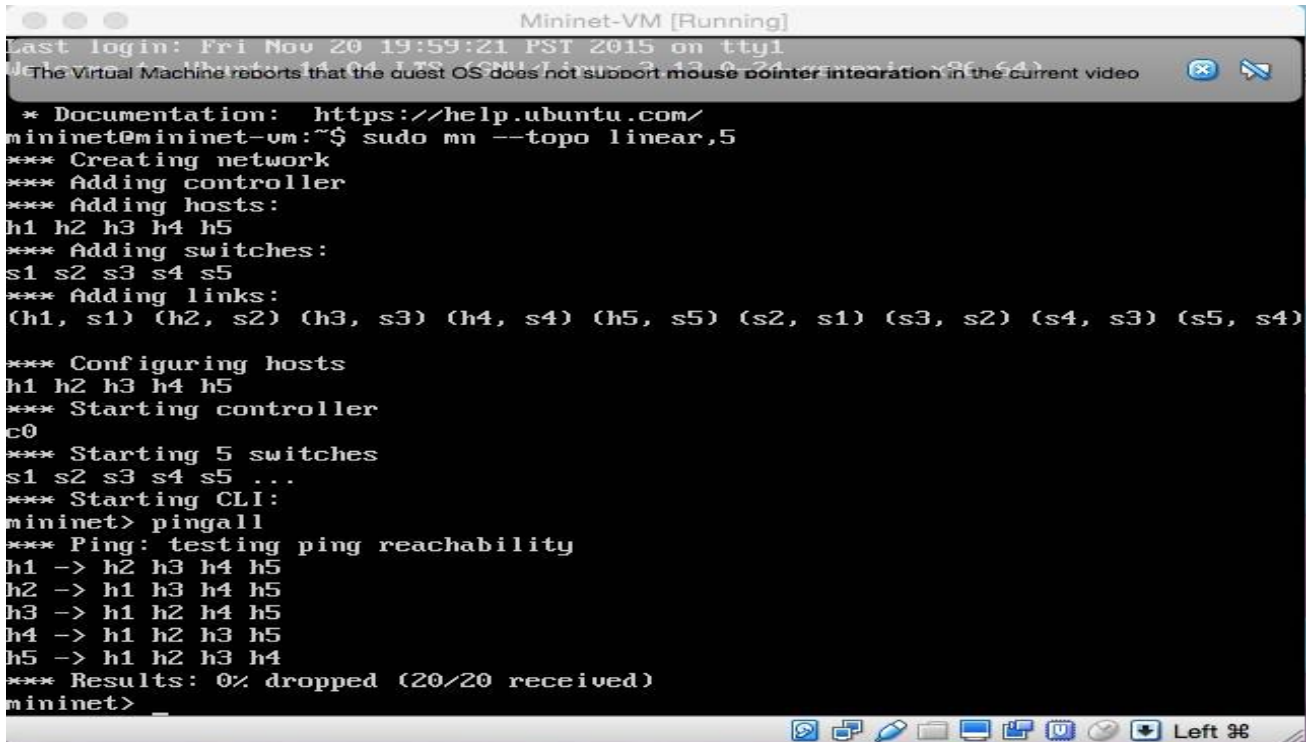


```
Mininet-VM [Running]
Last login: Fri Nov 20 19:59:21 PST 2015 on tty1
The Virtual Machine reports that the guest OS does not support mouse pointer integration in the current video
* Documentation: https://help.ubuntu.com/
mininet@mininet-vm:~$ sudo mn --topo linear,5
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (h5, s5) (s2, s1) (s3, s2) (s4, s3) (s5, s4)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet>
```

Step3: Creating a Tree topology

Commands used:

Sudomn topo linear,5



```
Mininet-VM [Running]
Last login: Fri Nov 20 19:59:21 PST 2015 on ttty1
The Virtual Machine reports that the guest OS does not support mouse pointer integration in the current video

* Documentation:  https://help.ubuntu.com/
mininet@mininet-vm:~$ sudo mn --topo linear,5
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (h5, s5) (s2, s1) (s3, s2) (s4, s3) (s5, s4)

*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet>
```


Step4: To create a topology and assign sequential mac address

Commands Used: `sudo mn --topo single,3 --mac --switch ovsk --controller remote`

```
**** Done
completed in 117.637 seconds
mininet@mininet-vm:~$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
**** Creating network
**** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
**** Adding hosts:
h1 h2 h3
**** Adding switches:
s1
**** Adding links:
(h1, s1) (h2, s1) (h3, s1)
**** Configuring hosts
h1 h2 h3
**** Starting controller
c0
**** Starting 1 switches
s1 ...
**** Starting CLI:
mininet> nodes
available nodes are:
c0 h1 h2 h3 s1
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=3906>
<Host h2: h2-eth0:10.0.0.2 pid=3909>
<Host h3: h3-eth0:10.0.0.3 pid=3911>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=3916>
<RemoteController c0: 127.0.0.1:6633 pid=3900>
mininet>
```

4.3 VIRTUAL BOX

Oracle VM VirtualBox (formerly Sun VirtualBox, Sun xVMVirtualBox and InnotekVirtualBox) is a hypervisor for x86 computers from Oracle Corporation. Developed initially by Innotek GmbH, it was acquired by Sun Microsystems in 2008 which was in turn acquired by Oracle in 2010. VirtualBox may be installed on a number of host operating systems, including: Linux, OS X, Windows, Solaris, and OpenSolaris. There are also ports to FreeBSD and Genode. It supports the creation and management of guest virtual machines running versions and derivations of Windows, Linux, BSD, OS/2, Solaris, Haiku, OSx86 and others, and limited virtualization of OS X guests on Apple hardware. For some guest operating systems, a "Guest Additions" package of device drivers and system applications is available which typically improves performance, especially of graphics.

DEVICE VIRTUALIZATION

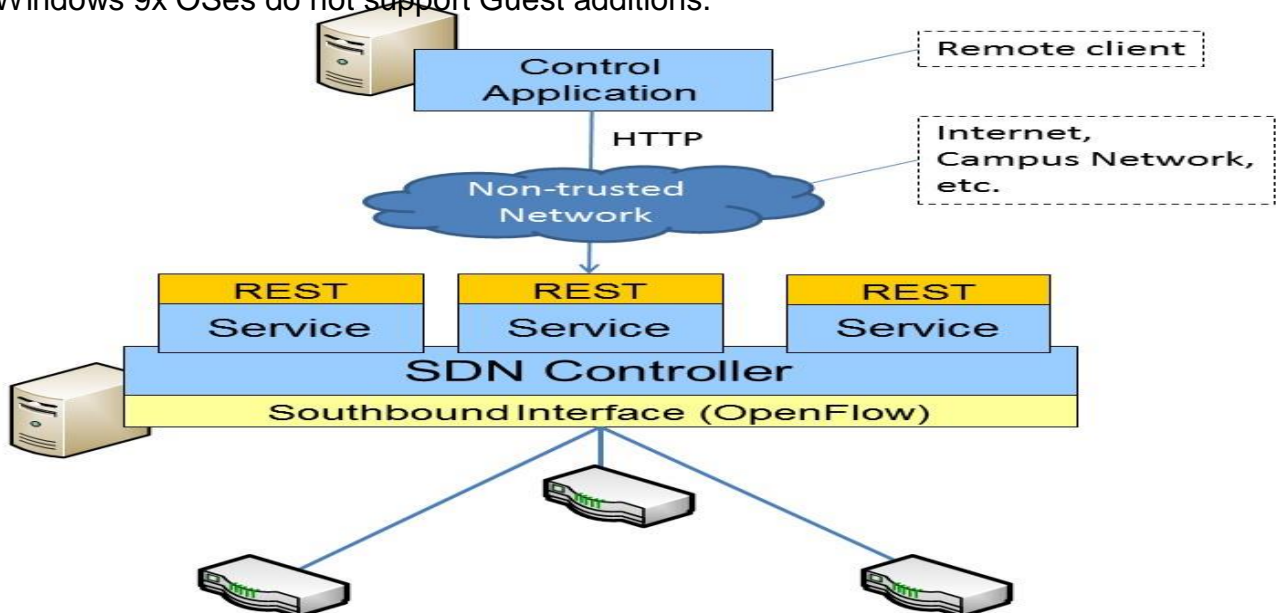
The system emulates hard disks in one of three disk image formats:

1. VDI: This format is the VirtualBox-specific *VirtualBox Disk Image* and stores data in files bearing ".vdi" filename extension.
2. VMDK: This open format is used by VMWare products such as VMWare Workstation and VMWare Player. It stores data in one or more files bearing ".vmdk" filename extensions. A single virtual hard disk may span several files.
3. VHD: This format is the native virtual disk format of Microsoft operating system, starting with Windows 7 and Windows Server 2008 R2. Data in this format are stored in a single file bearing ".vhd" filename extension.

A VirtualBox virtual machine can, therefore, use disks previously created in VMWare or Microsoft Virtual PC, as well as its own native format. VirtualBox can also connect to iSCSI targets and to raw partitions on the host, using either as virtual hard disks. VirtualBox emulates IDE (PIIX4 and ICH6 controllers), SCSI, SATA (ICH8M controller) and SAS controllers to which hard drives can be attached.

LIMITATIONS

- VirtualBox has a very low transfer rate from and to USB2 devices.^{[41][42]}
- Even though VirtualBox is an open source product some of its features are supplied only in a binary form under a commercial license (see The extension pack below)
- USB3 devices pass through is not supported by older guest OSes like Windows Vista and Windows XP due to the lack of drivers
- Windows 9x OSes do not support Guest additions.



5. PROJECT REQUIREMENTS SPECIFICATION

5.1 Project Analysis

This is the first phase of the project which includes identifying the problem(s) in the chosen domain and analysing the feasibility of solving the problem(s), Analysis also involves the tasks that go into induction, appraising and definition of a new or altered system. this forms a significant part of the design process that is the starting phase of the project. once the feasibility of solving the selected problem is done the next step involves identifying the requirements, this analysis phase also involves project requirements specification.

5.2 Project Requirements

- ⌘ Real time applications require stringent quality of service(QoS) guarantees.
- ⌘ Thus there is a need for the network programmers to design network protocols that can deliver performance guarantees.

Solution:

- ⌘ Firstly, we define a QoS Management and Orchestration architecture that allows us to manage the network in a modular way.
- ⌘ Secondly, we provide a seamless integration between the architecture and the standard SDN paradigm following the separation between the control and data planes.

Objective:

- ⌘ Split the bandwidth according to the number of nodes connected.
- ⌘ The system has several virtual machines running , connected to the SDN controller.
- ⌘ **Input to the controller:** Sample of certain bandwidth.
- ⌘ **Process:** Controller senses the input and identifies the bandwidth. According to the number of nodes/switches connected to the controller, the bandwidth is split.
- ⌘ **Output:** Once the bandwidth reaches the nodes, it is tested using netperf

5.3 Project Functions

The main deployment where this project can made is in medium to large scale organizations particularly organizations in the information technology field where there is need to manage the computer network in the organization efficiently and where network management forms a Centralized Signature based firewall over SDN Controller basic requirement for Mission critical applications deployment, According to the study and from my conclusion from my literature survey the applications developed using mininet emulator can be deployed in real network nodes

that support Openflow standard with minimal changes in the code. the firewall with a well defined User interface with more number of rules can be deployed in real networks if the underlying controller supports or provides access to various network parameters. The firewall can be deployed dynamically , automatically to any network topology with minimal changes in the code hence dynamical scaling and de-scaling of network nodes can be done. the network nodes are added the rules pro-actively that is before the network starts running hence this is called proactive Forwarding, but the flow rules can be updated by giving higher priority to the rule even while the network is running such is the flexibility and dynamicity of network management(particularly firewall rules management), In the future the no of firewall rules can be scaled once the support of various network parameters are supported such as all the header fields of a protocol such as TCP.

5.4 Operating Environment

It is the comprehensive set of tools, technologies required in order to carry out the project, so two types of operating environments need to be considered

1. Hardware Environment
2. Software Environment
3. Testing Environment

Hardware Environment is the underlying hardware needed in order for software environment to be installed, configured , tested and deployed, so in order to know for project to work the basic requirement is to specify the minimum hardware configuration in order to ensure the compatibility with Software Environment to set up in the next stage, ensuring the accurate and minimal hardware forms a basic preliminary requirement for setting up the Software environment. Software Environment is the very next stage in operating environment specification after Hardware environment specification, all the software needed to run the project must be Centralized Signature based firewall over SDN Controller specified and collected and has to be ensured in order to avoid compatibility issues later in the system design and project development stage otherwise it might lead to stasis in the project development.

Testing Environment is the optional third stage , if the project needs to be tested in the real environment this forms a significant stage of the project as it is needed to ensure the correct and desired working of the project and also to ensure the project requirements is met or not, but for this project there is non-availability of real testing environment hence it should be tested in the simulation environment which often forms as part of the project testing, project costs also play the role in setting up the testing environment, if the cost is significant compared to cost of project development then better to go for simulated test cases or simulation environment.

5.5 System Requirements Specification

A System requirements is the revealing of broad set of requirements needed for system design, building, constructing, testing and deploying the Project, all the system requirements specification(SRS) must be done during system design phase in order to confirm the feasibility of the project development. Different kinds of Requirement specification for system are shown below.

1. Hardware Requirements
2. Software Requirements

5.5.1 Hardware Requirements

CPU : Intel i5 or more,2.1 GHZ or more

Centralized Signature based firewall over SDN Controller

Memory : 4GB or more

Disk : 40GB or more

5.5.4 Software Requirements

Operating System: Windows 7 or more, Ubuntu 13 or more.

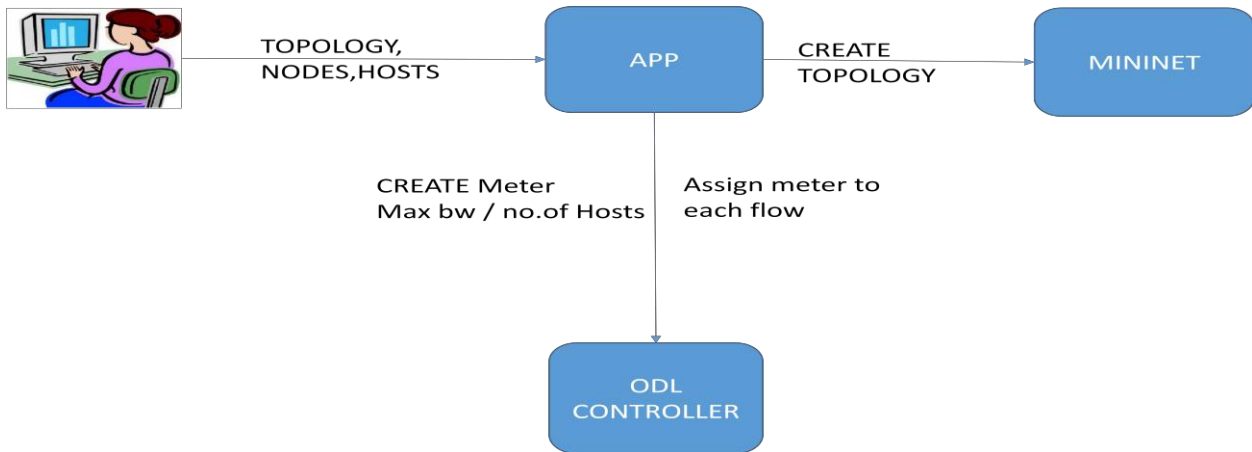
mininet SDN Emulator

Virtual box Hypervisor

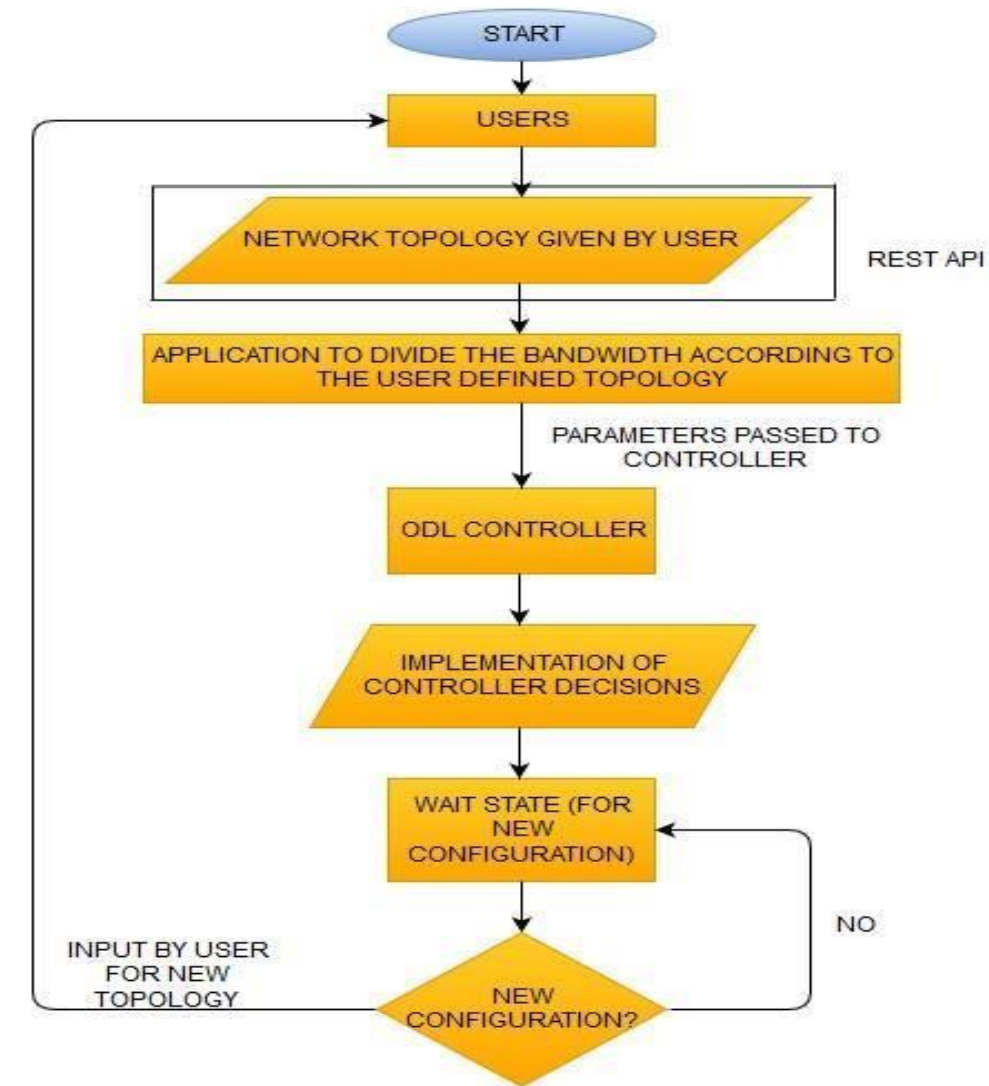
Opendaylight Controller

6 SYSTEM DESIGN

6.1 Topology



6.2 Flowchar



7.SOURCE CODE

HOW TO SET UP KARAF!

Start the karaf distribution and install the following features:

```
feature:install odl-restconf odl-l2switch-switch odl-mdsal-apidocs  
odl-adsal-all odl-adsal-northbound odl-dlux-core
```

After that connect mininet (sudo mn --topo single,3 --mac
--controller 'remote,ip=192.168.56.1,port=6633' --switch ovsk) and do a
pingall, everything works as expected. You can see the configured Flows
via `ovs-ofctl` on my mininet.

To get the statistics use a REST call like the one below:

GET

<http://localhost:8181/restconf/operational/.opendaylight-inventory:nodes/node/openflow:1/table/0>

To push for a Static Flow,

Chrome with PostMan:

Content-Type: application/xml

Accept: application/xml

Authentication

PUT

<http://localhost:8181/restconf/config/.opendaylight-inventory:nodes/node/openflow:1/table/0/flow/1>

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
<flow xmlns="urn:opendaylight:flow:inventory">
```

```
<priority>2</priority>
```

```
<flow-name>Foo</flow-name>
```

```
<match>
```

```
<ethernet-match>
```

```
<ethernet-type>
```

```
<type>2048</type>
```

```
</ethernet-type>
```

```
</ethernet-match>
```

```
<ipv4-destination>10.0.10.2/24</ipv4-destination>
```

```
</match>
```

```
<id>1</id>
```

```
<table_id>0</table_id>
```

```
<instructions>
```

```

<instruction>
<order>0</order>
<apply-actions>
<action>
<order>0</order>
<dec-nw-ttl/>
</action>
</apply-actions>
</instruction>
</instructions>
</flow>

```

To create an meter with the band type drop which limits the bandwidth to 50000kbps you can use the following REST call:

Address (change "ip" and "openflow:1" according to your settings):

PUT http://ip:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/meter/1

XML:

```

<meter
xmlns="urn:opendaylight:flow:inventory">
<meter-id>1</meter-id>
<container-name>mymeter</container-name>
<meter-name>mymeter</meter-name>
<flags>meter-kbps</flags>
<meter-band-headers>
<meter-band-header>
<band-id>0</band-id>
<band-rate>50000</band-rate>
<meter-band-types>
<flags>ofpmbt-drop</flags>
</meter-band-types>
<band-burst-size>0</band-burst-size>
<drop-rate>50000</drop-rate>
<drop-burst-size>0</drop-burst-size>
</meter-band-header>
</meter-band-headers>
</meter>

```


To assign a meter with id 1 to a flow (which should be limited to band of the meter) with id 10.

```
PUT http://ip:8181/restconf/config/opendaylight-  
inventory:nodes/node/openflow:1/table/100/flow/10  
<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<flow xmlns="urn:opendaylight:flow:inventory">  
<priority>2</priority>  
<hard-timeout>0</hard-timeout>  
<idle-timeout>0</idle-timeout>  
<flow-name>dl_dst-tb7-p2p4</flow-name>  
<cookie_mask>255</cookie_mask>  
<cookie>7</cookie>  
<match>  
<ethernet-match>  
<ethernet-destination>  
<address>00:1b:21:8b:83:93</address>  
</ethernet-destination>  
</ethernet-match>  
</match>  
<id>10</id>  
<table_id>100</table_id>  
<instructions>  
<instruction>  
<order>0</order>  
<meter>  
<meter-id>1</meter-id>  
</meter>  
</instruction>  
<instruction>  
<order>1</order>  
<apply-actions>  
<action>  
<order>0</order>  
<output-action>  
<output-node-connector>openflow:1:31</output-node-connector>  
<max-length>60</max-length>  
</output-action>
```

</action>
</apply-actions>
</instruction>
</instructions>
</flow>

XML USED FOR PUT AND GET METHODS:

PUT:

Accept: application/xml
Origin: chrome-extension://hgmloofddfdnphfgcellkdfbfjeloo
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/43.0.2357.130 Safari/537.36
CSP: active
Authorization: Basic YWRtaW46YWRtaW4=
Content-Type: application/xml
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Cookie: _sm_au_c=iVVMPPFsMt00NJv6P0d

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flowxmlns="urn:opendaylight:flow:inventory">
<strict>false</strict>
<instructions>
<instruction>
<order>0</order>
<go-to-table>
<table_id>1</table_id>
</go-to-table>
</instruction>
</instructions>
<table_id>0</table_id>
<id>256</id>
<cookie_mask>10</cookie_mask>
<installHw>false</installHw>
<match>
<ipv4-source>192.168.11.0/24</ipv4-source>
<ipv4-destination>192.168.11.0/24</ipv4-destination>
<ip-match>
```

```
<ip-protocol>4</ip-protocol>
</ip-match>
<in-port>0</in-port>
</match>
<hard-timeout>1800</hard-timeout>
<cookie>10</cookie>
<idle-timeout>1800</idle-timeout>
<flow-name>flow-instruction-go-to-table</flow-name>
<priority>2</priority>
<barrier>false</barrier>
</flow>
```

GET:

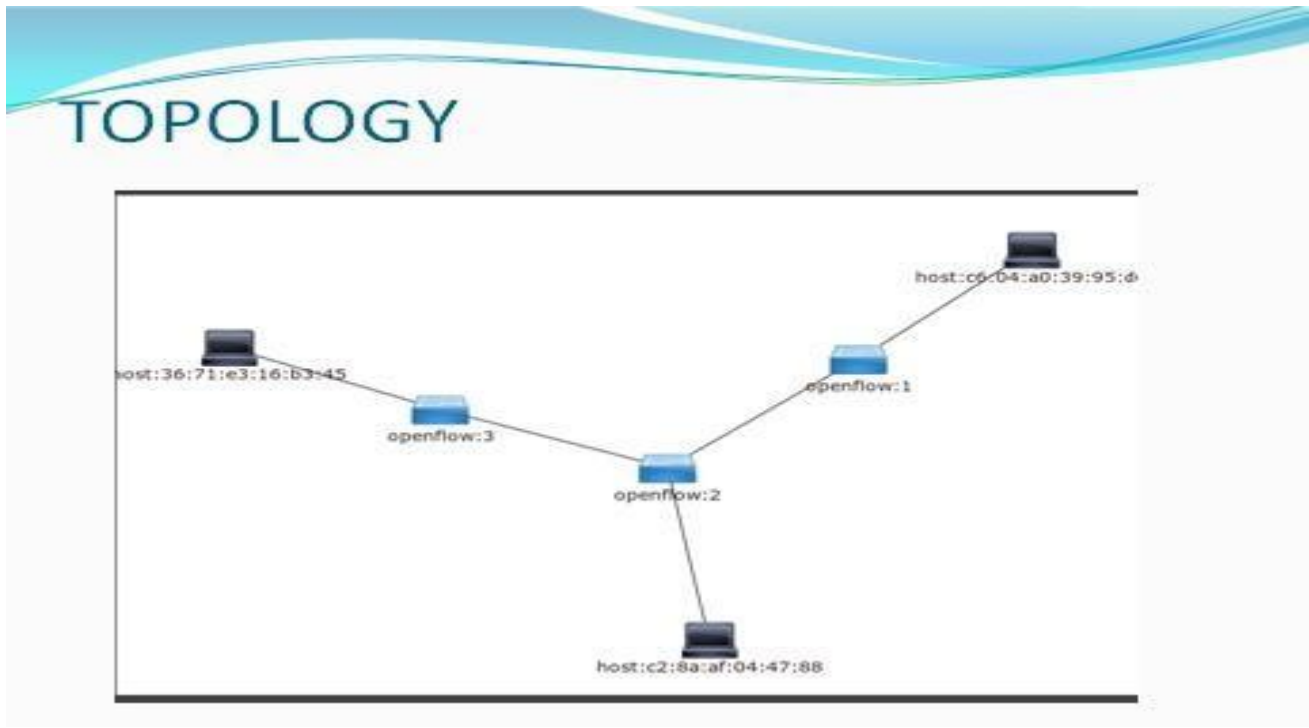
```
<flowxmlns="urn:opendaylight:flow:inventory">
<id>256</id>
<priority>2</priority>
<hard-timeout>1800</hard-timeout>
<instructions>
<instruction>
<order>0</order>
<go-to-table>
<table_id>1</table_id>
</go-to-table>
</instruction>
</instructions>
<match>
<in-port>0</in-port>
<ipv4-source>192.168.11.0/24</ipv4-source>
<ipv4-destination>192.168.11.0/24</ipv4-destination>
<ip-match>
<ip-protocol>4</ip-protocol>
</ip-match>
</match>
<flow-name>flow-instruction-go-to-table</flow-name>
<installHw>false</installHw>
<cookie_mask>10</cookie_mask>
<table_id>0</table_id>
<idle-timeout>1800</idle-timeout>
```

```

<barrier>false</barrier>
<strict>false</strict>
<cookie>10</cookie>
</flow>

```

8. RESULTS



Here we create meter-table, which is shown below. The meter-table triggers a variety of performance-related actions on a flow. Meter table consists of meter entries defining per flow meters. Per flow meters openflow to implement various simple Qos operations.

```

PUT http://localhost:8181/restconf/config/.opendaylight-inventory:nodes/node/openflow:2/flow-node-inventory:meter
Authorization:
Headers (3):
Body:
form-data x-www-form-urlencoded raw binary JSON (application/json)
{
  "flow-node-inventory:meter": [
    {
      "meter-id": 1,
      "flags": "meter-kbps",
      "meter-band-headers": {
        "meter-band-header": [
          {
            "band-id": 0,
            "band-burst-size": 0,
            "drop-rate": 50000,
            "drop-burst-size": 0,
            "meter-band-types": {
              "flags": "oipmbt-drop"
            },
            "band-rate": 50000
          }
        ]
      },
      "meter-name": "mymeter"
    }
  ]
}
Status: 200 OK Time: 24 ms

```

Here we are assigning the meter bandwidth to the flow

[illegible]

9. CONCLUSION

- ⌘ The purpose of this project was to create Qos based bandwidth split.
 - ⌘ We were able to develop a SDN application to get the input of the number of nodes and assign bandwidth accordingly.
 - ⌘ We were able to create the meter table to measure the performance related issues End
- Fragment