

BANDWIDTH ON DEMAND **AS A NETWORK SERVICE**

SUBMITTED BY-

Athreya Shenoy Yelthimar (asy140130)

Nakul Dahiwade (nxd141030)

Ram narayan Madhav Peri (nrp140130)

Charan kumar edamalapati(cxe140230)

Udaya Bharatha Reddy Yella(uxy140230)

TABLE OF CONTENTS

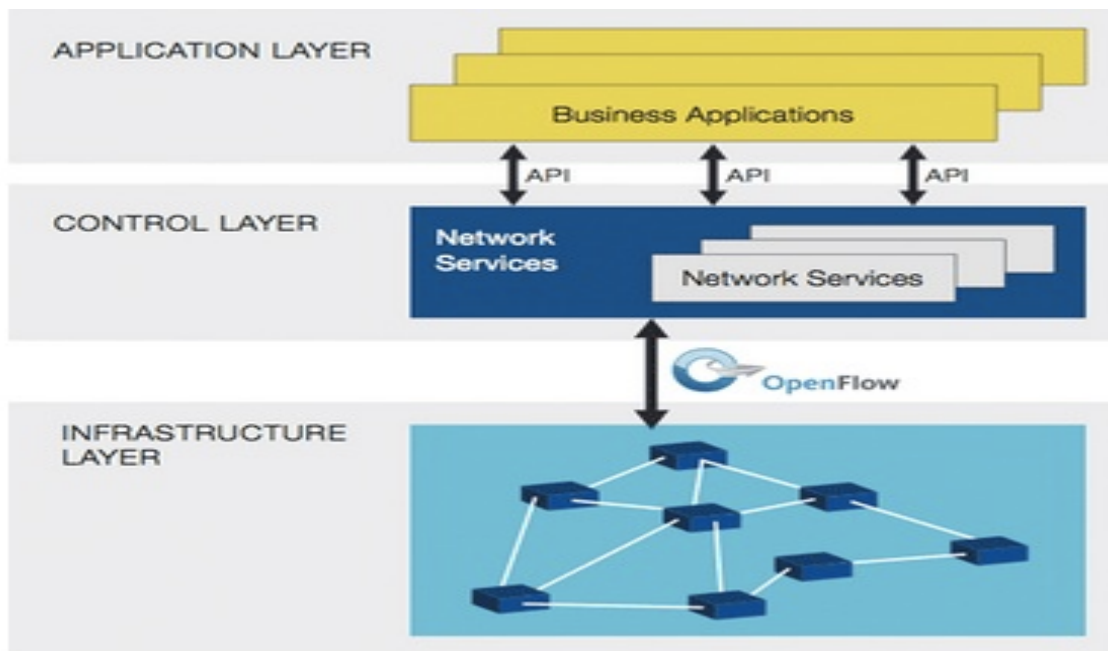
TITLE	PAGE NO.
INTRODUCTION TO SDN	3
INTRODUCTION	5
ARCHITECTURE	6
MAJOR COMPONENTS	7
FLOWCHART	10
WORKING	13
VALIDATION	15
PROBLEMS FACED	16
SCOPE	17
REFERENCES	17

INTRODUCTION TO SDN

Software Defined Networking is a new approach to designing, building, and managing networks that separates the network's control (brains) and forwarding (muscle) planes to better optimize each.

SDN providers offer a wide selection of competing architectures, but at its most simple, the SDN method centralizes control of the network by separating the control logic to off-device computer resources. All SDN models have some version of an SDN Controller, as well as southbound APIs and northbound APIs:

- **Controllers:** The “brains” of the network, SDN Controllers offer a centralized view of the overall network, and enable network administrators to dictate to the underlying systems (like switches and routers) how the forwarding plane should handle network traffic.
- **Southbound APIs:** SDN uses southbound APIs to relay information to the switches and routers “below.” OpenFlow, considered the first standard in SDN, was the original southbound API and remains as one of the most common protocols. Despite some considering OpenFlow and SDN to be one in the same, OpenFlow is merely one piece of the bigger SDN landscape.
- **Northbound APIs:** SDN uses northbound APIs to communicate with the applications and business logic “above.” These help network administrators to programmatically shape traffic and deploy services.



SOFTWARE DEFINED NETWORKING ARCHITECTURE

BENEFITS OF SDN:

Offering a centralized, programmable network that can dynamically provision so as to address the changing needs of businesses, SDN also provides the following benefits:

- **Reduce CapEx:** SDN potentially limits the need to purchase purpose-built, ASIC-based networking hardware, and instead supports pay-as-you-grow models.
- **Reduce OpEX:** SDN enables algorithmic control of the network of network elements (such as hardware or software switches / routers that are increasingly programmable, making it easier to design, deploy, manage, and scale networks. The ability to automate provisioning and orchestration optimizes service availability and reliability by reducing overall management time and the chance for human error.
- **Deliver Agility and Flexibility:** SDN helps organizations rapidly deploy new applications, services, and infrastructure to quickly meet changing business goals and objectives.
- **Enable Innovation:** SDN enables organizations to create new types of applications, services, and business models that can offer new revenue streams and more value from the network.

INTRODUCTION

Bandwidth-on-demand is a network connectivity service that allows users to request bandwidth at desired levels when and where they need it.

Connectivity services are a mostly static experience, meaning that once the endpoints and service attributes are defined, not much changes: someone needs a connection, they sign a contract, it gets turned on. And that usually takes quite a long time to establish, from a few days to few weeks.

However, the age of cloud networking means that static connectivity just isn't enough anymore. The demands put on our networks are changing. The expectation is that network capacity will adapt to our bandwidth needs and changing end points, instead of vice versa.

There are two types of Bandwidth-On-demand solutions:

- **Instant Bandwidth on demand:** Depends on the capacity of the network to support the requested bandwidth connection at the time of the request and for the desired duration of connectivity.
- **Scheduled Bandwidth on demand:** Allows you to reserve the bandwidth increase you need in advance.

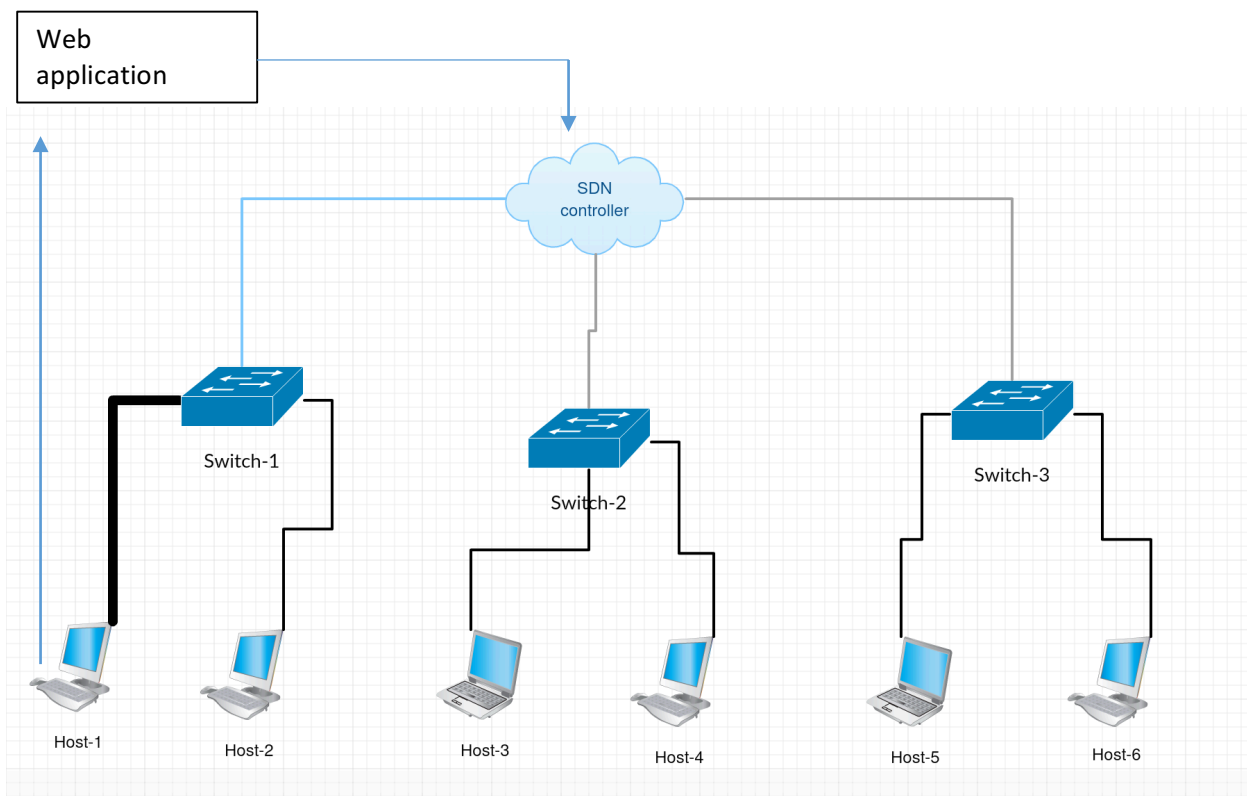
BANDWIDTH on DEMAND VS STATIC NETWORKS

Static Connectivity	Bandwidth on Demand
Network over-provisioning,	More accurate provisioning,
Slow turn up	Faster turn up
Multiple enterprise contracts	Consolidated SLAs
Eroding services revenue for	Increased market share,
Lack of service differentiation	Flexible and more appealing

ARCHITECTURE

Our Project focuses on Web-Application based service which can control/assign the bandwidth of a Virtual Network based on Mininet using OpenDaylight as the SDN Controller.

The Architecture of the Application involves a Web GUI(Graphical User Interface) where the User can enter the parameters(bandwidth) according the dynamic nature of the network which gets translated to the SDN Controller(OpenDaylight) via its Northbound API's which then translates the request into appropriate changes in the OVSDB Server(Database engine) via its Northbound plugin. It notifies the updates to the OpenVSwitch which then creates/manipulates/reconfigures ports, bridges, links in the network.



ODL Components Used

1. OVSDB
2. Restconf Api
3. Northbound Api

OVSDB

OVS Database is on mininet VM and its common to all the switches.

We can connect to this OVSDB in two ways:

- a. Active mode: ODL initially connects to OVSDB and after that it will listen to changes on OVSDB.
- b. Passive mode: OVSDB will listen for changes from ODL. We are making use of this mode.

In OVSDB we use following tables

Port Table:

This table has details of all the interface which connect to corresponding hosts and each interface is associated with Qos foreign key(UUID of QoS table). Each tuple is uniquely identified by UUID.

Qos Table:

This table contains Quality of Service (QoS) configuration for each Port that references it. Each row is identified by UUID and is mapped to several Queue in Queue table.

Queue Table

This table contains a configuration for a port output queue, used in configuring Quality of Service (QoS) features. We can set max_rate and min_rate using Other-config column. We used max_rate to set the maximum bandwidth which user request and min_rate to some predefined value.

Restconf Api

We Used this Api to set flows.

While setting the flow we define set queue in the action to the queue number in QoS Table.

et_queue is always zero because we have only one queue associated with each port.

Northbound Api

Northbound Apis are used to access OVSDB on mininet.

MAJOR COMPONENTS

OPENDAYLIGHT: OpenDaylight Project (ODL) is an open source SDN project aimed at enhancing software-defined networking (SDN) by offering a community-led and industry-supported framework. It is open to anyone, including end users and customers, and it provides a shared platform for those with SDN goals to work together to find new solutions.

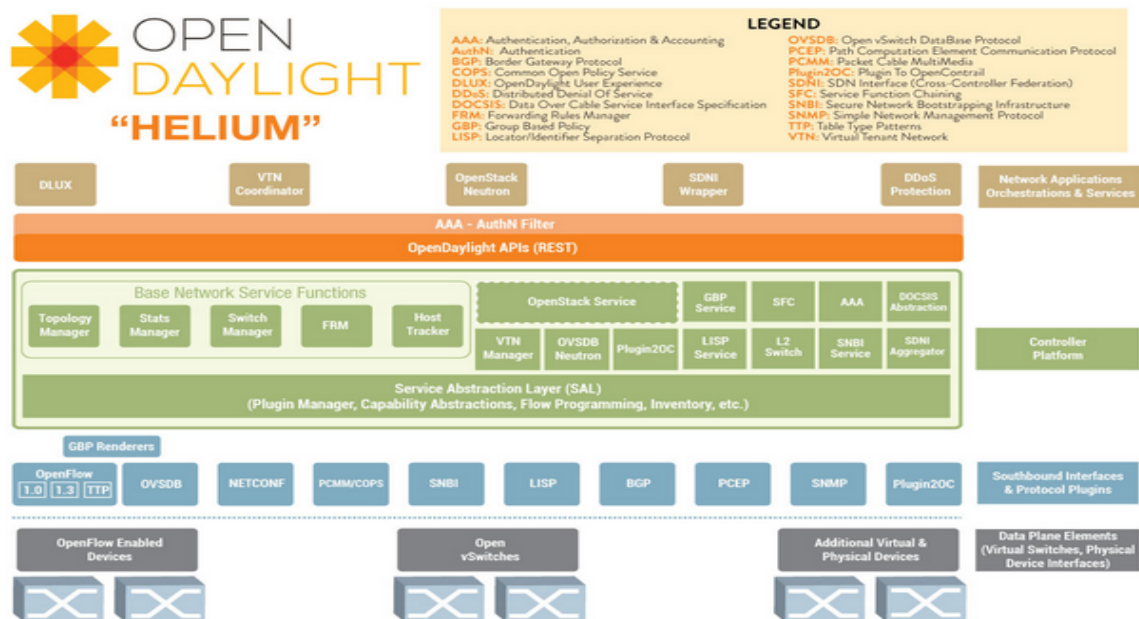
OpenDaylight includes support for the OpenFlow protocol, but can also support other open SDN standards.

The OpenFlow protocol, considered the first SDN standard, defines the open communications protocol that allows the SDN Controller to work with the forwarding plane and make changes to the network.

The OpenDaylight Controller is able to deploy in a variety of production network environments. It can support a modular controller framework, but can provide support for other SDN standards and upcoming protocols.

The OpenDaylight Controller exposes open northbound APIs, which are used by applications. These applications use the Controller to collect information about the network, run algorithms to conduct analytics, and then use the OpenDaylight Controller to create new rules throughout the network.

The OpenDaylight Controller is implemented solely in software, and is kept within its own Java Virtual Machine (JVM). This means it can be deployed on hardware and operating system platforms that support Java.



MININET: Mininet emulates a complete network of hosts, links, and switches on a single machine. To create a sample two-host, one-switch network.

Mininet is useful for interactive development, testing, and demos, especially those using OpenFlow and SDN. OpenFlow-based network controllers prototyped in Mininet can usually be transferred to hardware with minimal changes for full line-rate execution.

Mininet creates virtual networks using process-based virtualization and network namespaces - features that are available in recent Linux kernels. In Mininet, hosts are emulated as bash processes running in a network namespace, so any code that would normally run on a Linux server (like a web server or client program) should run just fine within a Mininet "Host". The Mininet "Host" will have its own private network interface and can only see its own processes. Switches in Mininet are software-based switches like Open vSwitch or the OpenFlow reference switch. Links are virtual ethernet pairs, which live in the Linux kernel and connect our emulated switches to emulated hosts (processes).

```
completed in 58.679 seconds
mininet@mininet-vm:~$ sudo mn --topo linear,4
*** 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> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

Creates a Virtual Network Instantaneously with the desired Network Topology

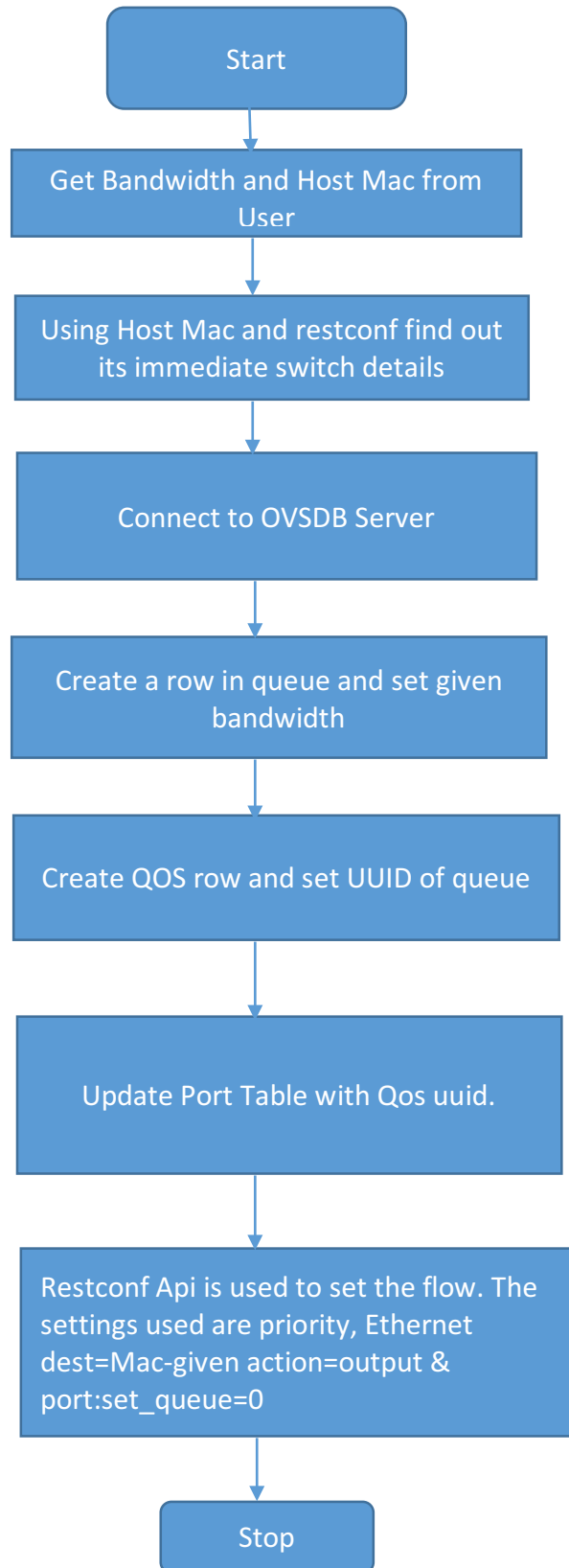
OVSDB: The ovsdb-server program provides RPC interfaces to one or more OpenvSwitch databases (OVSDBs). It supports JSON-RPC client connections over active or passive TCP/IP or Unix domain sockets.

Each OVSDB file may be specified on the command line as database. If none is specified, the default is /etc/openvswitch/conf.db. The database files must already have been created and initialized using, for example, ovsdb-tool create.

Features:

- Native JSON RPC v1 support. In fact, this is the only way you can talk to it.
- Persistent plain TCP socket connections. No HTTP overhead. You can optionally wrap your connections with SSL/TLS for super secret message sending.
- Journaled in-memory datastore.
- Atomic transactions.
- Relational data with automatic garbage collection of orphaned rows.

FLOW CHART



WORKING

The OVSDB Server has Three Tables namely Queue Table, QOS Table and Port Table. A row/record is created in the Queue Table according the configured Bandwidth, a uuid is generated for that row which is mapped/translated to the uuid of the QOS Table which is then mapped/translated to the uuid of the Port Table which OpenVSwitch refers to create/alter flows.

Port table															
uuid	bond_downdelay	bond_fake	iface	bond_mode	bond_updelay	external_ids	fake_bridge	interfaces	lacp	mac_name	other_config	qos			
	statistics	status	tag	trunks	vlan_mode										

956cd74f-6819-4ef4-9226-cd5e66333f8c	0			false		[]	0	{ }	false	[f1c1a73f-c5b0-4cfc-9f7a-b0ee90f0c8dd]	[]	[]	"s1"	{ }	[]
	{ }	{ }	[]	[]	[]										
e8995b7e-f616-414e-84bd-eafb04101828	0			false		[]	0	{ }	false	[3c7bf7e8-ac70-4846-9998-d321fce73790]	[]	[]	"s1-eth1"	{ }	3ae25588-fb25
-40f5-8d24-ad88beb29112	{ }					[]	[]	[]							
00d82a07-e39f-4969-97ad-efdb1b271425	0			false		[]	0	{ }	false	[a5808786-0965-4747-ae09-f0001f43a153]	[]	[]	"s1-eth2"	{ }	[]
	{ }	{ }	[]	[]	[]										

QoS table															
uuid	external_ids	other_config	queues	type											

58db965c-77cd-4eb0-a173-a4f0c779395d	{ }	{ }	{0=9ab0a9da-1e7f-432b-a0bd-48a5245be76b}	linux-htb											
bc69e751-b323-47f5-b890-a16a2644f753	{ }	{ }	{0=9ce46459-9932-4b6b-a4fb-523c2479744d}	linux-htb											
3461b5e0-666a-49bf-a9a7-846fc6bc2c26	{ }	{ }	{0=a13a7283-0375-4937-870f-8b4c7c785d9d}	linux-htb											
3ae25588-fb25-40f5-8d24-ad88beb29112	{ }	{ }	{0=b0d6ceda-c022-4323-a3cc-f17f9d3adbec}	linux-htb											
1680fac8-3b44-461b-ac70-97689a1104a2	{ }	{ }	{0=e844b361-4b33-44fe-b564-c5d3b624213e}	linux-htb											

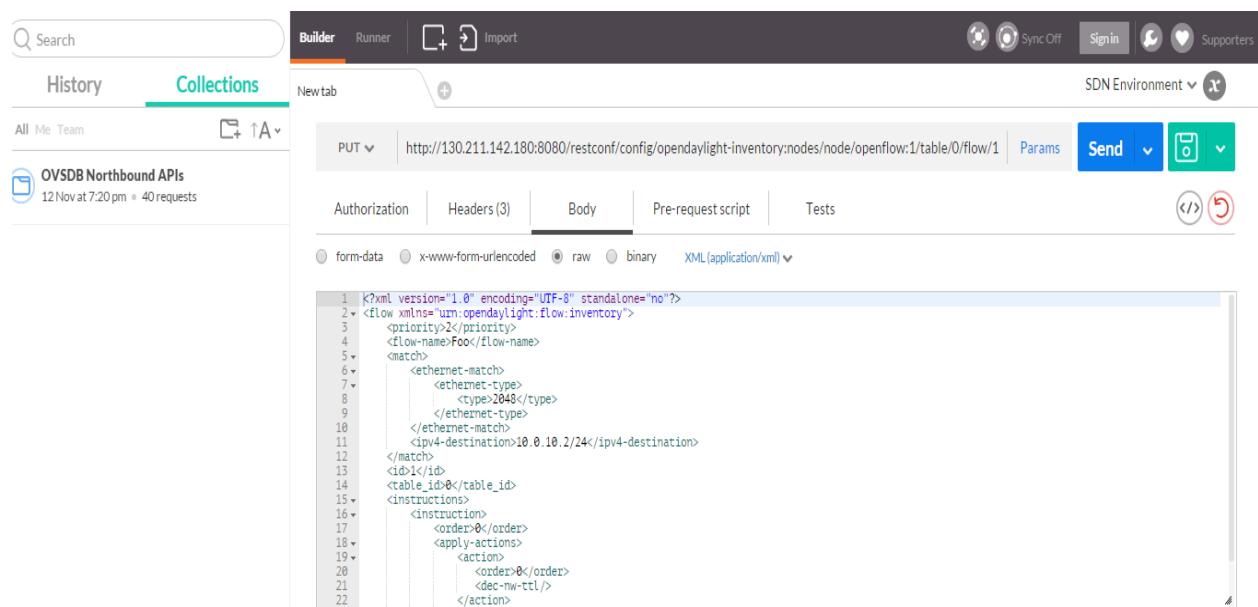
Queue table															
uuid	dscp	external_ids	other_config												

a13a7283-0375-4937-870f-8b4c7c785d9d	[]	{ }	{max-rate="20000000", min-rate="8000000"}												
b0d6ceda-c022-4323-a3cc-f17f9d3adbec	[]	{ }	{max-rate="20000000", min-rate="8000000"}												
9ab0a9da-1e7f-432b-a0bd-48a5245be76b	[]	{ }	{max-rate="30000000", min-rate="8000000"}												
e844b361-4b33-44fe-b564-c5d3b624213e	[]	{ }	{max-rate="32", min-rate="8000000"}												
9ce46459-9932-4b6b-a4fb-523c2479744d	[]	{ }	{max-rate="50000000", min-rate="8000000"}												

Three Tables in the OVSDB Server.

Generally there are two modes to connect to OVSDB(Open vSwitch Database Management Protocol) and ODL(Open Day Light)

- **Active** - In active connection OVSDB acts as client and ODL acts as server. We do not use active connection for connection.
- **Passive** - OVSDB server is listening and hence acts as server. And since ODL will connect to OVSDB server it will act as a client. There are three kinds of API's available in ODL: North Bound API's , Restconf , Netconf Among these three , only NorthBound OVSDB and restconf API's are used. NorthBound API's normally sit at the top of controller. It presents a network abstraction interface to the applications and management systems at the top of SDN stack. One of the main advantage of NorthBound API's is that it puts the application in control of the network. OVSDB NorthBound API's are used connect OVSDB to ODL. Restconf allows access to datastores locating in controller. There are two datastores which are called config and operational . It listens on port 8080 for HTTP requests. Resconf API is used mainly for two reasons: 1) To track the host 2) To set the flows.



Postman Fetching Flows from the restconf API

```

1 package com.sdn.app;
2
3 import java.io.FileReader;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18 public class Test {
19
20     public static void main(String[] args) {
21         String MAC="06:00:44:6a:83:55";
22         String Bandwidth="50000000";
23         String Controller_IP = "146.148.92.192";
24         String Mininet_IP = "10.240.0.3";
25         Host_location host_location = new Host_location(Controller_IP);
26         host_location.get_host_location();
27         // Get Switch details from host MAC
28         Switch s = host_location.host.get(MAC);
29
30         Set_Bandwidth_in_OVSDB ovsdb = new Set_Bandwidth_in_OVSDB(Controller_IP,Mininet_IP, Bandwidth, s.switch_id);
31         boolean status = ovsdb.Create_OVSDB_record();
32
33         Set_Flow flow = new Set_Flow(Controller_IP, MAC, 0, new Integer(s.port), s.id);
34         flow.Set();
35
36     }
37
38
39 }

```

VALIDATION

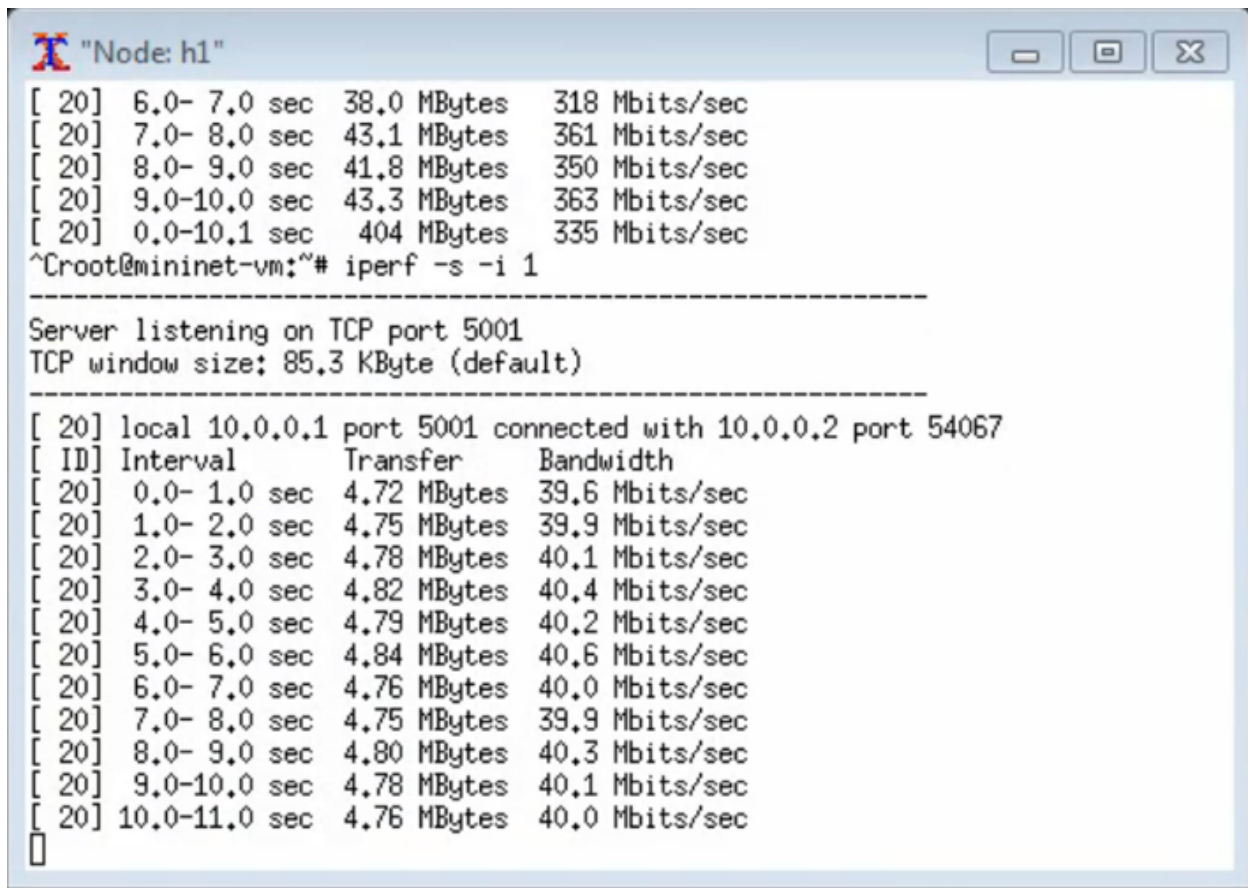
It is important to validate the results that we have acquired from the project we conducted. In order to do this, we are using a tool called IPERF. IPERF is a tool used to measure the bandwidth and the quality of the network link. IPERF is easy to install on any Linux/Windows system. One host is set as the server while the other must be set as the client. In our project the user who requests for the bandwidth acts as the server while any other random active host in the system can act as the client to test the bandwidth. In order to access the server and the client host's interface we are making use of Xterm. To test if the bandwidth requested for by the user has been properly set, the client host sends data and the rate at which the data is downloaded by the server host is measured. By default, the Iperf client connects to the Iperf server on the TCP port 5001 and the bandwidth

displayed by Iperf is the bandwidth from the client to the server. This way we can validate our project to be a success or a failure. Once we have access to the client and the server hosts using Xterm we need to execute the following command on either sides.

Server side: `#iperf -s`

Client side: `#iperf -c <server-IP>`

The results of the above validation (server side screen shot) are as show below (User set the bandwidth to 43mbps in web GUI):



```

"Node: h1"
[ 20] 6.0- 7.0 sec 38.0 MBytes 318 Mbits/sec
[ 20] 7.0- 8.0 sec 43.1 MBytes 361 Mbits/sec
[ 20] 8.0- 9.0 sec 41.8 MBytes 350 Mbits/sec
[ 20] 9.0-10.0 sec 43.3 MBytes 363 Mbits/sec
[ 20] 0.0-10.1 sec 404 MBytes 335 Mbits/sec
^Croot@mininet-vm:~# iperf -s -i 1
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 20] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 54067
[ ID] Interval      Transfer      Bandwidth
[ 20] 0.0- 1.0 sec  4.72 MBytes  39.6 Mbits/sec
[ 20] 1.0- 2.0 sec  4.75 MBytes  39.9 Mbits/sec
[ 20] 2.0- 3.0 sec  4.78 MBytes  40.1 Mbits/sec
[ 20] 3.0- 4.0 sec  4.82 MBytes  40.4 Mbits/sec
[ 20] 4.0- 5.0 sec  4.79 MBytes  40.2 Mbits/sec
[ 20] 5.0- 6.0 sec  4.84 MBytes  40.6 Mbits/sec
[ 20] 6.0- 7.0 sec  4.76 MBytes  40.0 Mbits/sec
[ 20] 7.0- 8.0 sec  4.75 MBytes  39.9 Mbits/sec
[ 20] 8.0- 9.0 sec  4.80 MBytes  40.3 Mbits/sec
[ 20] 9.0-10.0 sec  4.78 MBytes  40.1 Mbits/sec
[ 20] 10.0-11.0 sec 4.76 MBytes  40.0 Mbits/sec

```

PROBLEMS FACED

ODL caches the details of the host, if we restart the mininet for example 1st time we start mining with 2 hosts and then we restart it again with 3 hosts. Now when we use restconf to get the details of all the hosts we get 5 hosts instead of 3

OVSDB Northbound Apis doesn't work with lithium version and there is no documentation for OVSDB lithium. This is one of the reason why we used Helium instead of Lithium.

Northbound Api to set flow(flow programmer) doesn't work. So we used restconf to set flow.

ODL caches flows. For example once you have set the flow and then restart the ODL the flows will get updated automatically.

There is no proper documentation for how to set flows using restconf and there is no xml format to set queue in action. In this project we got the correct format after trail and error.

ODL is unstable. When you run this project with ODL it works fine for the first time and it may crash in the subsequent attempts. Work around for this is to delete the ODL and unzip downloaded ODL.

This project doesn't work in the college intranet since we are using mininet and ODL on different machines and for connecting mininet to ODL we have to configure mininet VM as bridged network which will get ip-address from the university dhcp server. However the university's dhcp server fails to assign IP.

SCOPE

The project can be extended/developed further to provide the following services/application:

- Client Based SDN or Bandwidth-On-Demand/QoS network services.
- Has the potential to provide Application driven SDN/BoD/QoS Service.
- Based on this system the Traditional Service Provider Client Billing system can change based on Bandwidth, Time, Application running Time, Service etc.

REFERENCES

<https://www.opendaylight.org/sites/opendaylight/files/bk-developers-guide-20141002.pdf>

<http://www.ciena.com/connect/blog/What-is-Bandwidth-on-Demand.html>

<http://www.technologyreview.com/news/409540/bandwidth-on-demand/>

<http://manpages.ubuntu.com/manpages/saucy/man1/ovsdb-server.1.html>

<https://github.com/mininet/mininet/wiki/Introduction-to-Mininet>

<https://www.sdxcentral.com/resources/sdn/what-the-definition-of-software-defined-networking-sdn/>