IOSR Journal of Engineering (IOSRJEN)
e-ISSN: 2250-3021, **p**-ISSN: 2278-8719
Vol. 3, Issue 8 (August. 2013), ||V4|| PP 32-41

Design and Implementation of Dynamic load balancer on OpenFlow enabled SDNs

Ragalatha P, Manoj Challa, Sundeep Kumar. K
[1.] M.Tech Computer Networks Computer Science CMR Institute of Technology, Bangalore, India
[2]. Associate Prof. Dept. of Computer Science CMR Institute of Technology, Bangalore, India
[3]. Head of Dept. of CMR Institute of Technology, Bangalore, India

Abstract: - To cope with heavy loads due to new technologies such as cloud computing and data-centers' the networking of devices on internet became more complex. To deal with complex demands the networks have to be smarter and able to control and manage the devices better hence the interest in software-defined networks. To effectively address the practical challenges of these networks the researchers have developed an open standard called OpenFlow. With OpenFlow firmware installed on routers and switches, users can use software to tap into flow tables and control a network's layout and traffic flow as fast as possible and flexibly. This software-based access allows inexpensively and easily to test new switching and routing protocols being developed from time to time. In this paper we present a dynamic load balancer that can enable application specific routing algorithms rather than network based static routing. The load balancer has been simulated on OpenFlow standard and it has been proved that load balancer is much faster and effective.

Keywords: - *Load balancer, Routing, SDN, OpenFlow standard*

I. INTRODUCTION

As the Internet continues to expand in size, the opportunities for innovation and challenges to networking researchers are increasing. It is estimated that roughly half of all network failures caused by the misconfiguration of networks and its devices. And the emerging new set of technologies and applications such as the cloud computing and massive datacenter's put heavy load on the networks and demand flexibility in their operations. To cope with these demands, networks have to be much smarter and able to control and manage themselves better. Thus, the growing interest in *software-defined networking* (SDN) based research [1].

Several networking related challenges such as robustness, ease of management, performance and cost of operations need to be handled flexibly and dynamically. The network administrators require a detailed and scalable control to maximize the total effectiveness of networking. Though SDNs alone promises administrators to help in this direction as stated, but its implementers doesn't provide such flexibility. For example, Cisco and Juniper provide proprietary implementations of the SDN technology [1] and they have their own well defined API and SDN functionality. However, it limits the ability to obtain and manage the traffic from networking devices supplied by multiple vendors and their interoperations. Therefore, there is a need for a standard for SDN technology and functionality. To address this practical challenge OpenFlow standard has been created.

OpenFlow is an open communication protocol used to control the network traffic flows of multiple networking devices such as switches from a centralized controller [1]. With OpenFlow controllers, administrators could define traffic flows and determine how packets are prioritized and forwarded through the networking devices over a network. Vendors and network operators begin to look towards this OpenFlow technology, as it promises to bring interoperability and better performance to the SDNs.

Load balancing in networks is a technique used to spread the heavy load across the multiple network links. Load balancing is achieved generally using highly specialized and thus expensive devices like load balancers that know a priory the information about the servers they are forwarding to. Load balancer improves the network performance by optimally using the available resources and helps in minimizing the latency and response time and maximizing throughput. They route packets based on the servers' current loads, location of content relative to the request, or some naïve rigid policies such as round-robin [6]. Since policy implementer and switch are coupled we are reduced to a single point of failure. Therefore, these kinds of load balancers are static in nature where the application specific routing is not a possibility.

Also, current load balancers have limited number of built-in algorithms that are also to be pre-configured at the router and likely tuned to a specific traffic pattern using current loads. In this paper, we explore how to offer an "Intelligent load balancing" service that is application specific so that the application user can choose among the available routing algorithms dynamically. This dynamic routing can be offered using the OpenFlow enabled networks due to its flexible OpenFlow API. Also this service should enable support for multiple load balancing algorithms as well as building of a flexible load balancer where actual balancing algorithms can be

dynamically changed based on changing traffic patterns. Load-balancer with an OpenFlow switch is less expensive than the commercial load-balancer. We have built such a dynamic load balancer using OpenFlow API and simulated the load on different types of networks.

In the next section, a detailed overview of OpenFlow standard is provided. Following it, a detailed architecture of dynamic loadbalancer using OpenFlow API is provided along with its implementation challenges and details. Lastly the conclusions are drawn.

II. OVERVIEW OF OPENFLOW STANDARD

In general the network devices have two parts: 1. the *control plane,* and 2. the *data plane*. In the control plane, the protocols that make majority of decisions regarding the traffic issues will be executed. In the data plane, the set of functions that forward packets based on the decisions taken in control plane will be executed. Whereas in, SDN network architecture, the control plane will be decoupled from the physical data plane. The control plane runs in software, generally on standard servers that are operated separately from the network devices such as switches. This separation allows the control plane to be implemented using a different distribution model than the data plane and hence can be programmatically managed and administered without disturbing the data flow at the device. Secondly, it allows the control plane development and runtime environment to be on a different computing platform than the conventionally low-powered management CPUs found on conventional switches. It gives the network administrators a more fine-grained control over the network traffic flows as they can get the data to the control plane and analyze it.

The key advantage of SDN are:

(1) Users can define network traffic flows and decide the routing paths in the network as per application specific demands.
(2) Programmatic and dynamic remote control of network hardware is possible.
(3) The use of standard, open hardware interfaces makes network devices free from a specific vendor.
(4) Ability to run new services and applications, with additional flexibility and control.
(5) Opportunities for development of advanced optimization and customization techniques and algorithms.

To achieve such flexibility demanded by current and future Internet applications, the SDN architecture requires a method for the control plane to communicate with the hardware data such as switch data. One such method is OpenFlow. Thus, OpenFlow provides the required support to enable the network innovations based on commercial switching hardware(s) by separating the intelligent control plane from the data path processing. The mapping between the switching hardware(s) and controller(s) could be either one-to-one or many-to-one devices. Because of the standardization of OpenFlow, vendors and network operators begin to look towards it. In March 2011, companies such as Cisco, Facebook, Yahoo, Google, and Microsoft formed the Open Networking Foundation (ONF) to promote OpenFlow technology [1] and the Software Defined Networking.

Characteristics of OpenFlow:

The OpenFlow network architecture consists of a set of elements. These are (1) networking devices such as routers, switches, (2) the controller - usually a standard computing server that controls the forwarding traffic, (3) a secure channel interconnecting the switch to the controller, and (4) the OpenFlow switching protocol for communication between the switches and the controller.

Typically an Ethernet switch runs both data plane and control plane in the same hardware. But OpenFlow separates the data and control plane functions as shown in figure 1. The data plane functionality is executed in the OpenFlow switch on the basis of a table called flow table, while the control plane functionality is handled by a separate computer based Controller. Flow tables describe how to move a packet from sender to destination efficiently. Though each vendor's flow table is different, a set of functions such as quality of service and traffic reporting are common to most switches. And the OpenFlow standardizes these common set of functions.

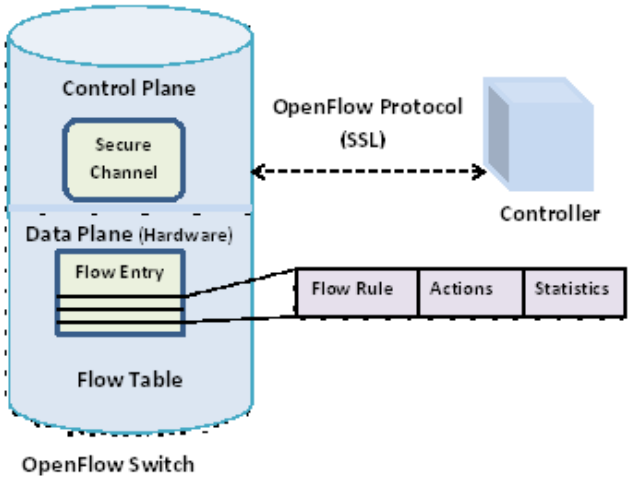

Figure 1: OpenFlow Network Architecture

OpenFlow Switches

OpenFlow switches are like standard hardware switches with flow tables performing the packet lookup and forwarding. The difference lies in how the flow rules are inserted and updated inside a flow table. A standard switch can have static rules inserted or be a learning switch where the switch inserts rules into flow table as it learns on which interface a machine is used. OpenFlow switch on the other hand, uses external controller to add rules into the flow table. Two types of OpenFlow switches are available for use. One is the hardware-based switch, which uses a proprietary operating system to implement the flow table and OpenFlow protocol. And the other is software-based switch that runs in UNIX/Linux based servers and implement the OpenFlow switch functions in software [2].

Controller

SDN uses the open-source controller that allows designers to write network control software in high-level programming languages such as Java and C#. Controller is responsible for adding or removing the new routing rules into the switch's flow table.

When a data packet arrives at a switch initially, the packet's destination is checked and forwarded according to a set of predefined forwarding rules. Packets going to same destination are routed along the same path and treated same manner further. The controller decides how packets of a new flow or of a mismatched flow are to be handled by the switch. When a new flow arrives at the switch, the packet gets redirected to the controller which decides whether the switch should drop the packet or forward it to a machine connected to the switch. The controller can also delete or modify existing flow entries in the switch. The controller executes modules that describes how new flows should be handled by the switches. Controller inserts a new rule into the switch's flow table using OpenFlow protocol.

Messages exchanged between switch and controller includes the information on sent packets, received packets, forwarding table modifications and statistics collection. The controller notifies switches to impose policies such as sending data by faster routes or having the fewest number of hops on network traffic flows for better performance.

Flow Table

A Flow table consists of flow entries used for directing an OpenFlow switch along the OpenFlow network for various incoming packets. When a packet from a client arrives at an OpenFlow switch, the packet header information is compared with flow table entries of the switch. Each flow entry consists of a set of flow rules, defined on basis of the packet header fields for packet matching, an action to be performed on the packets matching the flow rules, and the flow statistics, as shown in figure 2. A packet header includes port id, VLAN tag, Ethernet type, source and destination address, IP protocol type, source and destination address, UDP/TCP source and destination port.

The possible actions are:

(1) To send the packet to output ports,
(2) To forward the packet on an output port,
(3) Modify packet header fields, and
(4) To drop the packet

Flow statistics include number of packets and bytes arrived for each traffic flow. Time since the last packet matched also could be found to ease the task of removing inactive flows. A flow table entry can also have one or more action fields describing how matching packets are handled.

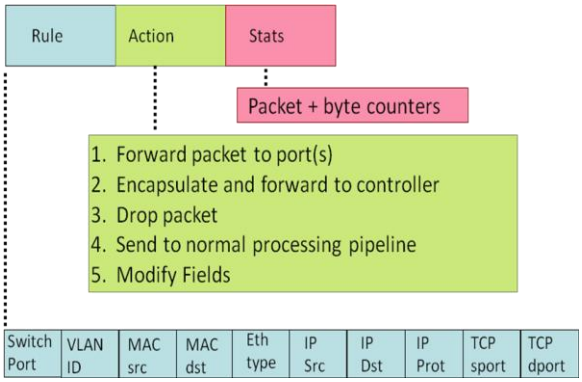

Figure 2: Flow Table entries

Each Flow entry has an expiration time after which the entry is deleted from the flow table. This expiration time is based on number of seconds a flow was idle and total amount of time the flow entry has been in the flow table. The controller can either chose a flow entry to exist permanently in the flow table or can set timers that delete flow entry when the timer expires.

Two flow tables are managed in the software based OpenFlow switch. The exact matching table and the linear table are the two tables, considered to manage for the OpenFlow forwarding rules. OpenFlow permits to set up rules with wildcards in the packet header fields to match packets to the flows. The rules for exact matching are stored in a hash table exploiting hashing function, and contains up to 131072 entries, while wild-cards are implemented using a linear search table allowing up to 100 entries.[2].

The Advantages of OpenFlow [1] are:

- **Performance and cost:** By removing the control processing load from switches, OpenFlow lets the switches to focus largely on moving traffic. Since the control function is independent of the hardware it controls, forwarding and routing processes are faster. By virtualizing the network management, OpenFlow enables the networks that are less expensive to build and run.
- **Implementing and testing new features:** Via the open API, OpenFlow lets administrators set up new features and switch functionality at full line rates using existing network architecture. These features work on multiple platforms so that users wouldn't have to implement them in each vendor's hardware. OpenFlow also enables administrators and researchers to write their own control software, which has been difficult in the past as major vendors' network devices lacked common APIs.
- **Security and management:** OpenFlow's centralized controller gives administrators a unified view of the network that facilitates better management, security, and other capabilities. By letting administrators view the overall traffic flow more clearly, OpenFlow makes spotting the intrusions and other problems easier.

The OpenFlow also has some concerns. They are:

- **Scalability:** Since OpenFlow centralizes network control functions into a single server controller there are concerns that the technology may not scale well to the mark.
- **Security:** It is feared that as OpenFlow places all management functionality in one virtual server, hacking the network will be easier. Henceforth, OpenFlow will fit well where we need less security.
- **Interoperability:** While basic OpenFlow work well, vendors may add proprietary extensions to the technology, eliminating the benefits of interoperability and make their versions incompatible with other implementations.

III. STATIC LOAD BALANCING

In static load balancing, network performance of available servers is determined at the beginning of execution of a service [4]. Depending upon their performance, the load is distributed among the servers in the beginning of service. The servers keep monitoring and calculating their allocated loads and report the results when there is a change in it to the controller. Then controller assigns a particular server to process the client request. Static load balancing helps to reduce the overall execution time by minimizing the communication delays. General disadvantage of all static load balancers is that the final selection of host is made when the

request is created and cannot be changed while in execution. The three static load balancing algorithms are as follows.

A. Round Robin Algorithm

In round robin, requests are divided evenly among all servers. A new request is assigned to a server in round robin fashion.

B. Randomized Algorithm

In randomized algorithm, requests are allocated to the servers randomly. Round Robin and Randomized algorithms work well when number of requests larger than number of servers. They can attain their best performances among all load balancing algorithms in special purpose applications and cannot expect to achieve good performance in general cases.

C. Load based Algorithm

In this algorithm, the controller selects the server with minimum load depending on the overall network load when a new request arrives. From network load state information the load balancing judgment is made. This information is updated each time the loads on the servers change.

IV. DYNAMIC LOAD BALANCING

In dynamic load balancing, the work load is calculated and distributed among the servers at runtime. The controller assigns new requests to the servers based on the load information collected. As shown in figure 3, a set of clients and servers are connected to a network. The controller connected to the network communicates via the OpenFlow protocol and has a set of defined load balancing algorithms.

Usually, a client resolves the hostname of a server to an IP address and sends its request to that IP on a known port number. In proposed design, the client sends the request to controller across the network without resolving the actual IP addresses of servers. Meaning, the first request from a host is sent on a virtual IP address of the server to the controller as IP addresses are not known when request is sent. The controller then resolves the IP address and sets this IP address for forwarding. Meanwhile, Servers register on the controller and report their current arbitrary loads as well as other attributes such as response time, total performance to simulate. When a request reaches the controller, the servers' performance and resources utilization are simulated under available different routing algorithms which are based on application specific request demands. Based on type of requests forwarded and results drawn from simulation and optimization, the controller forwards the request to chosen appropriate server using load balancing algorithms dynamically to process the request.

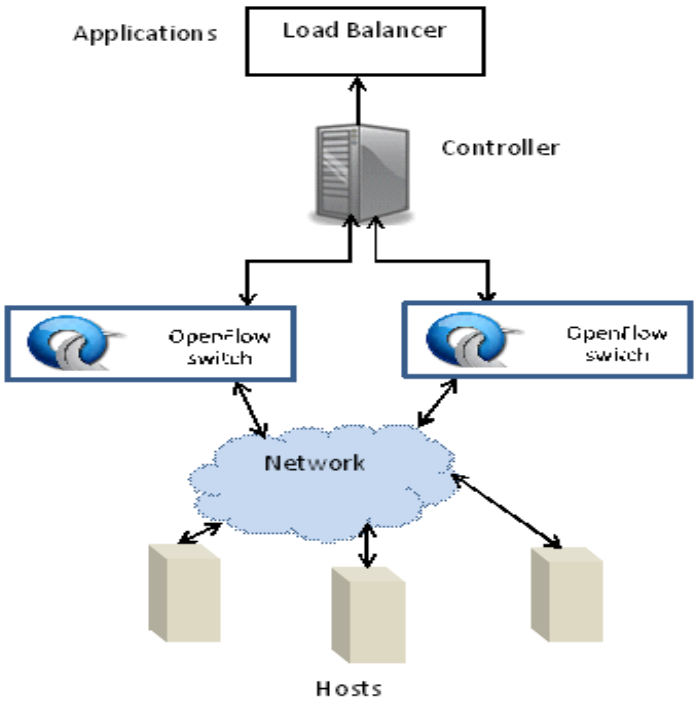

Figure 3: Load Balancing Architecture

V. IMPLEMENTATION

For implementing the dynamic utilization aware load balancer, the utilization of servers running applications and utilization of different network paths to servers is tracked continuously. The load balancing algorithm uses this tracking data to select the optimal combination of server and the path. The load balancing algorithm considered is based on the weighted least connections currently being serviced.

In least connections method, for selecting a node, only active connections are used in the calculations. These methods are relatively simple and best in environments where the servers have similar capabilities and capacities to avoid latency. For servers with varying capacities, weighted least connections methods are used instead.

Weighted least connections - Like the least connections methods, these weighted load balancing methods also select nodes based on number of active connections. However, they also base their selection decisions on server capacity or weight assigned to server. The criterion used by the proposed weighted load balancing algorithm to select appropriate server to serve a request is to ensure that the load is balanced across the network. The load on each server is computed as follows:

Number of active connections (transactions) currently being executed = n

Assigned weighted value of the server = w

Load on the server, **x = (Number of active transactions / weight) = n/w**

If servers are not handling any active transactions, then servers are selected in round robin fashion regardless of the weights assigned to them. First server in the series receives the current request when multiple servers have same load x. To test the load balancing performance, throughput across nodes, hosts and the switches are simulated and analyzed.

Definitions:

Floodlight OpenFlow Controller - Floodlight is an open source, Java based, Apache-licensed OpenFlow controller with rich extensible REST APIs [8]. Floodlight manages I/O from OpenFlow Switches, tracks end-host locations in the network and provides support for integrating with non-OpenFlow networks. Floodlight supports OpenFlow 1.0 specifications.

Mininet – Mininet is a network simulator used to create scalable SDNs. It is used for simulating the network topology and testing the traffic flows received from remote controller. Users create a network topology using miniedit.py script of Bob Lantz [7]. MiniEdit is a GUI-based network editor that allows users to drag and drop switches and hosts, wire them up, and create a live, usable network as in figure 4.

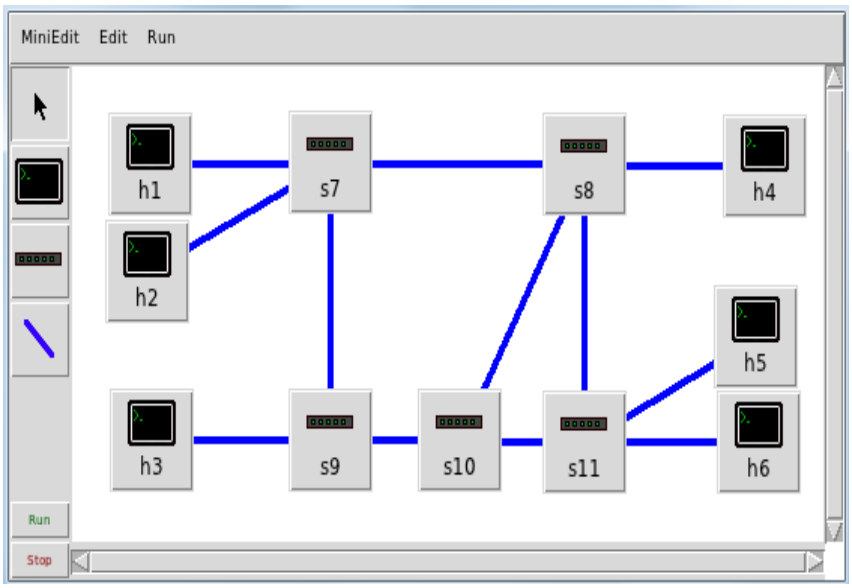

Figure 4: A 5 switch-6 host network created in Miniedit (where, h1-h6 are Hosts connected via links to s7-s11 are Switches)

We describe the Load balancing algorithm in figure 5:

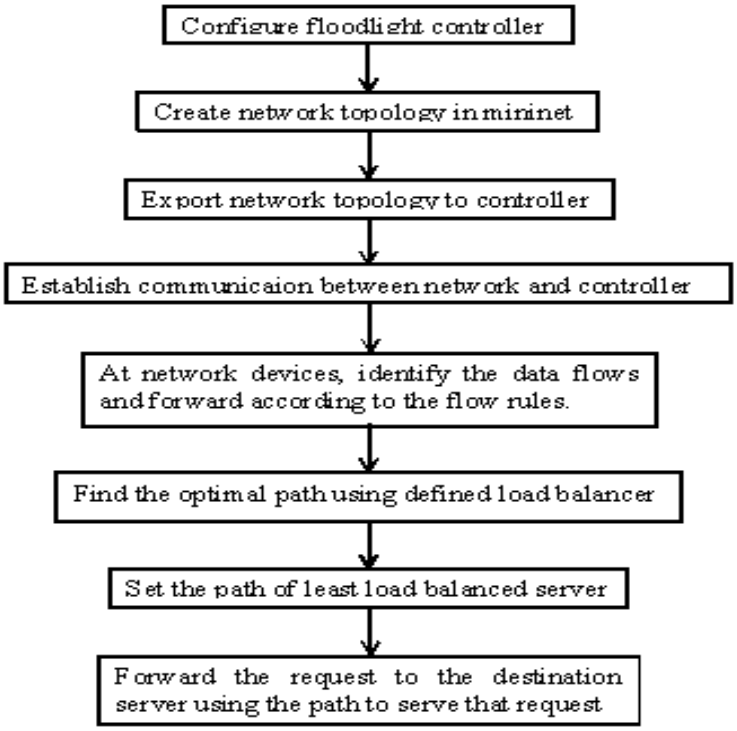

Figure 5: Load Balancing Alogithm

Further details of the load balancing algorithm: implemented using Mininet and Floodlight controller is described here in detail.

Input: Network topology with server weights

Output: Load balanced optimal path

Steps:

A. Configure Floodlight controller

1. At remote controller establish OpenFlow channel.

2. Instantiate controller //controller tcp: <ip_addr>: <tcp_port>

a. Assign IP address

b. Port number

c. Transport protocol

3. Start Floodlight controller

java -jar target/floodlight.jar

B. Create network topology

4. Create an enterprise network topology of switches and hosts through mininet. //using miniedit

5. Initialize and configure the hosts and switches assigning each a switch ID. sudo mn --custom ~/mininet/custom/<filename.py> --topo=custom

6. Configure the ports at switches and IP addresses of servers.

7. Store the topology for export into the controller.

8. Configure mininet for interacting with the remote controller.

C. Export network topology to controller

9. Import network topology from mininet

D. Establish communicaion between network and controller

10. Establish connection between switches and controller //Exchange Hello( ) and Echo( ) messages between switch and controller.sudo mn –custom ~/mininet/custom/<filename.py> --topo=custom –controller remote --ip=<controller IP> –port =<controller listening port>

E. At controller, identify the data flows and forward accordingly

11. For (every incoming packet_in) //switch performs lookup in its flow-table to find matching entry.

12. If (match found in lookup table) then,

13. Send the matched data and action set to next table.

14. Forward the packet on resolved output port in the

flow table; //Send packet_in to destination_server.

15. *End If*
16. *Else (not matched) then*
17. *Forward packet to controller; //packet_in from Switch to controller.*
18. *End For*

F. *Find the optimal path using defined load balancer*

19. *Identify and store all paths for every client-server combination //paths list*
20. *Initialize serverWeights //stores assigned weights of servers*
21. *Initialize serverLoads //stores server current loads typically no. of connections and duration of each connection.*
22. *For every incoming flow*
23. *Read flow entries to perform associated actions*
24. *For every new packet arriving at the controller*
25. *If (ip exists in packet_in) then*
26. *Identify least loaded server*

a. *compute each server current load Actual load of current server=No. of connections/ weight of server.*
b. *Sort all servers to find least loaded destination server*

27. *Increment the load at selected destination server*
28. *End If*
29. *End For*
30. *Choose the optimal path with least connections among all available paths to the destination server*

a. *Get all paths from source node to destination server from paths list*
b. *Calculate every path's total load based on connection load*
c. *Sort all paths to find least connection loaded final optimal path*

G. *Forward packet to the destination server usig the path*

31. *Update final path link cost and flow entries of all nodes //nodes= participating switches in chosen final path*
32. *Update port and IP of destination server for packet_in.*
33. *Client request is serviced by the least loaded server on the optimal path found*.

REST API:

REST API is the interface used to install flow rules using curl commands utilizing the features of Floodlight. This REST API is available at port 8080 of the controller.

REST API call instantiates a flow matching all fields and an associated action for matching rule in the switch's flow table. A Priority is also set so that any matching packet with a priority greater than specified priority can take precedence over the flow rule being added. When a flow rule is added on a switch, flow entries are as shown in figure 6.

Example: To add a flow, *curl*
-d '{"switch": "<DPID>", "name":"< flow name>", "cookie":"0", "priority":"32768", "active": "true", "actions":"output=2"}'
http://<controller-ip>:8080/wm/staticflowentrypusher/json

To retrieve DPIDs of connected switches to controller, *curl*
http://<controller-ip>:8080/wm/core/controller/switches/json

To retrieve flow statistics, *curl*
http://<controller-ip>:8080/wm/core/switch/all/flow/json

To delete a flow, *curl -X DELETE -d '{"name":"<flow name>"}'*
http://<controller-ip>:8080/wm/staticflowentrypusher/json

Other possible load balancing algorithms that can be used to implement using OpenFlow APIs apart from the utilization aware load balancer are:

1. *Application Aware:*

On Categorizing applications into different profiles based on their CPU and Network utilization, a request is assigned based on current state of active requests across applications and the profile of newly requested application. For example, it is possible to combine applications of high CPU intensity and low network intensity with those that of low CPU intensity and high network intensity in the same server.

2. *Energy Aware:* Extension of Application Aware load balancing with energy efficiency as a goal.
3. *Priority Driven:* Enables requests of specific applications to be given higher priority in selecting servers and paths so that they have the most guaranteed path possible, at the cost of other applications.
4. *Optmized load balancers:* Using optimisation algorithms such as genetic algorithm, the load can be optimised and path can be determined.

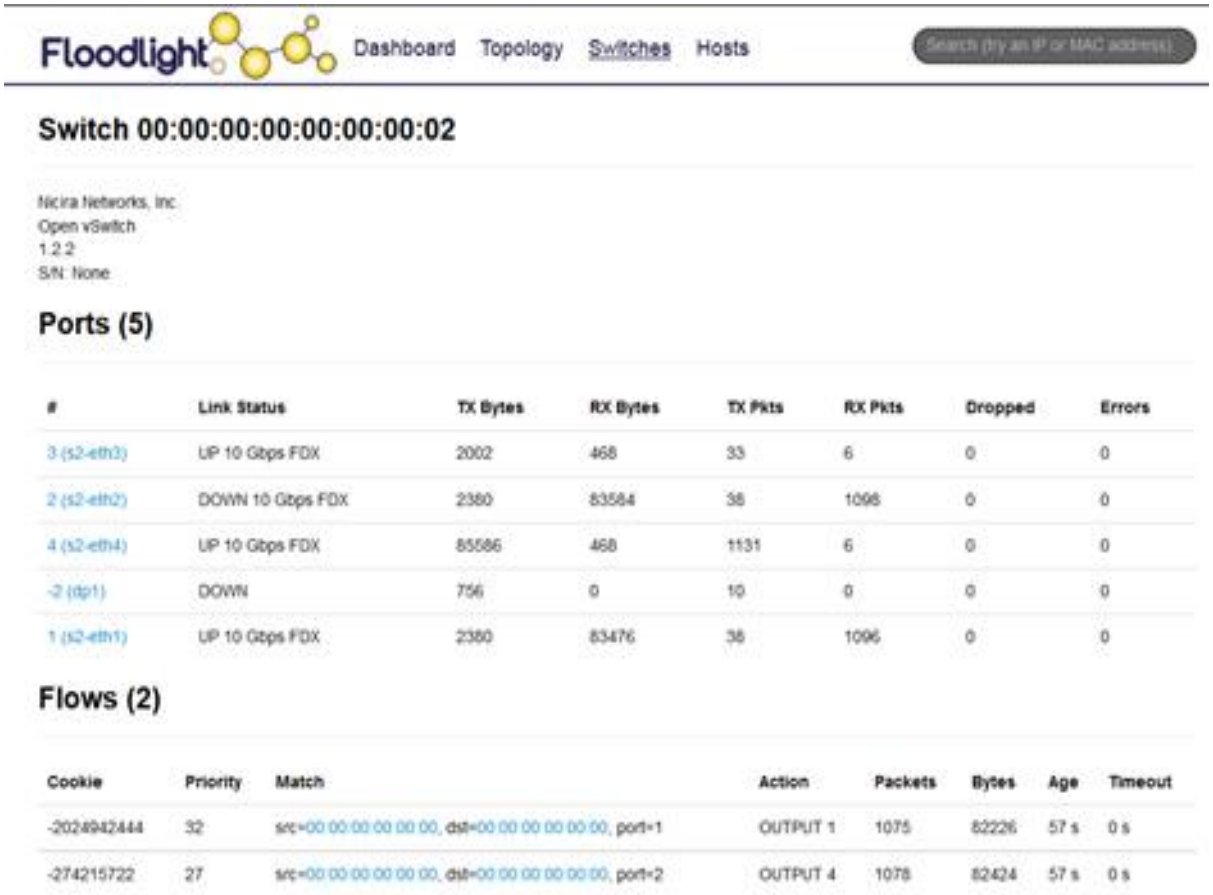

Figure 6: Flow entries to a particular switch

VI. CONCLUSION

OpenFlow allows networks to evolve and improve more quickly than they do at present. OpenFlow reduces the friction of implementing network changes for many organizations. In networks, where cost is a primary factor, it's likely to adopt OpenFlow as primary configuration tool for almost all network requirements. By choosing a load balancing algorithm dynamically, the total effectiveness of OpenFlow network is improved to larger extent. Though it was possible to get similar functionality from a commercial load-balancer using hardware switches. The dynamic selection of load balancing approach in OpenFlow network provides the required flexibility in implementation and routing. If OpenFlow can simplify the network architectures, and streamline network operations, it could be succeed but if OpenFlow demands new skills and requires complex network programming, it may not become popular.

REFERENCES

[1] Steven J. Vaughan-Nichols, "*OpenFlow: The Next Generation of the Network*?", IEEE computer , August 2011, PP 113-115.

[2] Andrea Bianco, Robert birke, Luca Giraudo, and Manuel Palacin, "*OpenFlow Switching: Data Plane Performance",* IEEE Communication Society, Proceedings of IEEE ICC July 2010.

[3] Richard Wang, Dana Butnariu and Jennifer Rexford, "*OpenFlow- Based Server Load Balancing Gone Wild",* December 2010.

[4] Sandeep Sharma, Sarabjit Singh and Meenakshi Sharma, *"Performance Analysis of Load Balancing Algorithms"*, World Academy of Science, Engineering and Technology 38 2008.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "*OpenFlow: Enabling Innovation in Campus Networks",* ACM SIGCOMM Computer Communications Review, 38(2):69–74,March 2008.
[6] Hardeep Uppal and Dane Brandon , "*OpenFlow Based Load Balancing",* University of Washington .
[7] Bob Lantz, "GitHub", Miniedit file, April, 2010 https://github.com/mininet/ mininet/blob/master/examples/miniedit.py
[8] The Floodlight REST API http://docs.projectfloodlight.org/display/floodlightcontroller/REST+API
[9] Brandon Heller, "*OpenFlow Switch Specification Version 1.1.0* ", 28 February 2011 http://www.OpenFlowswitch.org/documents/OpenFlow-spec-v1.1.0.pdf.
[10] Nick Feamster, "*Software-Defined Networking and the New Internet*", 28 September 2010 http://connectionmanagement.wordpress.com/2010/09/28/software-defined networking-and-the-new-internet/
[11] Open Networking Foundation http://en.wikipedia.org/wiki/Open_Networking_Foundation

Authors:
Ms. Ragalatha P is presently doing Master of Technology in Computer Networks and Engineering in CMR Institute of Technology, Bangalore, India. She obtained her Bachelor of Engineering degree in Computer Science and Engineering from RNS Institute of Technology, Bangalore, India in the year 2010.

Mr. Manoj Challa is pursuing Ph.D(CSE) in S.V.University, Tirupati, India. He completed his M.E (CSE) from Hindustan College of Engineering, Tamil Nadu in 2003. He is presently working as Associate Professor,CMR Institute of Technology, Bangalore. He presented nearly 18 papers in national and international conferences. His research areas include Artificial intelligence and computer networks.

Mr. Sundeep Kumar K received the M.Tech (IT) from Punjabi University in 2003, ME (CSE) from Anna University in 2009 and pursuing Ph. D (CSE) from JNTUA. He is with the department of Computer Science & Engineering and working as Head of Department of Computer Science, CMR Institute of Technology, Bangalore. He presented more than 10 papers in International and national Conferences. His research interests include OOMD, Software Engineering and Data Warehousing. He is a life member in ISTE.