Abstract

Networks of today has emerged from series patches here and there to make it stable, scalable and manageable. Despite these regular improvements and upgrading, computer networks of today are struggling to meet the heterogeneous and ubiquitous society’s demand. The architectural inflexibility of computer networks gives researchers challenges in experimenting new ideas. The paradigm Software Defined Networking (SDN) and OpenFlow architecture, open ways for the implementation of a programmable network architecture to be applied gradually in production networks. In this paper, software switches in a virtual network environment replace physical switches. Displaying the SDN network in GUI, Floodlight controller was implemented as the network OS. Adapting its codes to develop components; packet forwarding/address learning, load balancer and firewall for optimizing network. This research achieved a state of the art network management from both a centralized and GUI controller.

Key Terms:  Software Defined Networks, Floodlight, OpenFlow, Mininet, controller, switch
1. **Introduction**

Continually states of network changes, and network operators must manually adjust network configurations in reaction to dynamic conditions of the network [1]. External tools are frequently used by network operators to manage their network, some end up building adhoc scripts to dynamically reconfigure devices on the network to handle events as they occur resulting in frequent changes and misconfigurations [2]. A new paradigm Software Defined Networking (SDN) has presented an alternative to manage such limitations. SDN is based on decoupling the control and data planes of a network, which implies that the switch is confined to a simple packet-forwarding device. A software-based controller now manages the rules followed by the switch to forward packets to destination [3]. Developed applications can now control the switches by running on top of a network operating system.

OpenFlow [4] [5] [6] [7] [8] was introduced to standardize the communication between switch (data plane) and the software-based controller (control plane) in SDN architecture. OpenFlow networks possess certain capabilities like centralized control of multiple switches from a single controller, analysis of traffic statistics using software, dynamic update of forwarding information and so much more [3].

There are several SDN controllers available, examples are NOX [9], Beacon [10], Maestro [11], Floodlight [12], Frenetic [13], Trema [14], and etc. The programming language used in coding a controller will be same a programmer will use to code applications for that controller.

Floodlight is an SDN-OpenFlow controller that comes with built-in applications [12]. A set of common functionalities is used by FloodLight Controller to probe and control an OpenFlow network, while applications on top of it achieve different features to solve various user needs over the network. Floodlight architecture, comprises of three inter-related components the Floodlight Controller, the applications built as Java modules compiled with Floodlight, and the applications built over the Floodlight REST API [15]. Java was used to develop network management modules and compiled them with floodlight controller module loading system.

Mininet is a system that provides large networks for rapid prototyping on a single computer. Using lightweight virtualization mechanism it creates scalable software-defined networks, such as processes and network namespaces. Users can create, interact with, customize and share prototypes faster using mininet.
2. **Research Design and Methodology (important for research papers)**

Codes for these functional components/modules were done in Java, based on Floodlight Controller and then implemented in Mininet simulated network.

The simulated network is designed to run on a PC running FloodLight Controller, 7 switches and 8 hosts. This architecture replicate data center architecture with 3 depths (The Core Switch (S1), the aggregate switches (S2 and S5) and The Edge Switches (S3, S4, S6 and S7)), with each edge switch connecting 2 hosts. This architecture gives much leeway for implementing and testing its applications to show SDN global visibility of the entire network.

![GUI display of network design.](image1)

Fig 1: GUI display of network design.

To create the network a guest machine was setup on VirtualBox “SDN with other controllers” running Ubuntu 14.04 64 bit Operating System, 2GB RAM with double processor, and setup two Network adapter interfaces one for “Network Address Translation” (NAT) to enable internet from the Host computer and the other for Host-only.

![VirtualBox running my SDN with Floodlight controllers](image2)

Fig 2: VirtualBox running my SDN with Floodlight controllers
Inside the guest machine, **Mininet** was setup to emulate the network and use as test bed, also **OpenFlow 1.3** (most recent version of OpenFlow as at the time of this research), **Open-V switch** (the OpenFlow enabled software switch), **Floodlight** branch 1.3 (most recent build of Floodlight Controller), **Eclipse**, among other utility applications that came with the Operating System which will be very essential, for example Terminals, xterm, Emacs, browser etc. was setup.

Setting up to run Floodlight, Virtual machine was powered up and a new workspace created in eclipse.

![Fig 3: Floodlight loaded on eclipse](image)

The code given in Table 1 below was entered in a Terminal to create and connect the network to Floodlight controller.

**Table 1: Network Topology**

```plaintext
sudo mn
controller=remote,ip=127.0.0.1,port=6653
mac
topo=tree,3
switch ovsk,protocols=OpenFlow13
```

(This command invokes Mininet and create the network as designed with a tree topology of 3 depths (core, aggregates and edge switches), each edge switches connecting to 2 (8 hosts in total) and instruct Mininet to use Open Vswitch with kernel, start the controller, connect the network through the OpenFlow interface on the software switch to the controller on the PC’s IP address and listening port then await further commands on Mininet mode to properly manage the network.)
Immediately the network on Mininet connects to floodlight controller running on eclipse, a handshake (HELLO PACKET sent and received) occurs to confirm the connection.

3. **Results and Discussion (Important for research paper)**

The switches start its self-study actions to learn the network paths as packets arrive its port. Then with the Address Learning Component, the switch installs all IP addresses learned into its flow table.

**a. Using the Pingall command:** issuing this command triggers all network hosts to send ping packets to each other to validate their connectivity and reachability. It uses (Internet Control Message Protocol) ICMP.

![Pingall command](image)

**Fig 4: Pingall command triggering packet forwarding between all hosts in the network**

All devices successfully pinged one another. And the address learning module populated the flow table with both the MAC and IP addresses to each destination.

![GUI display](image)

**Fig 5: GUI display of all network devices**
b. Using the iperf command:

![Graph showing bandwidth between hosts using Floodlight controller.]

Fig 6: Bandwidth between hosts using Floodlight controller.

The above graph shows that TCP bandwidth between hosts in a network that is using Floodlight controller within 5 sec:

- For Switch 3, h1 and h2: data was transferred at a rate of 9.67 Gigabits per second
- For switch 4, h3 and h4: the rate at which data was transferred 9.3 Gigabits per second
- For Switch 6, h5 and h6: data transfer rate was 9.39 Gigabits per second
- For Switch 7, h7 and h8: the data transfer rate was 10.0 Gigabits per second

From this test between hosts, it is concluded that Address learning component implemented to populate switch flow table with addresses associated to its port increases data transfer to destinations within the same switch.

c. Implementing firewall component:

```
$ curl http://localhost:8080/wm/firewall/module/enable/json
```

This command activates the firewall module compiled with the floodlight controller and blocks all communication between hosts on the network. To allow communication, rules has to be inserted in the firewall, so that any packet coming in will be checked with the rule before an ALLOW or DENY action will be implemented on the packet.
To implement the firewall using a bank network scenario with the server located on host 7 (h7), no host on the network should be able to send successful ping to h7 except the bank manager PC located in host 1 (h1).

Fig 7: Ping from hosts H1 to H7 confirming communication deny by firewall

Not to forget that the firewall has been implemented and activated already, hence, all communications are blocked, there firewall has to be configured to allow ping communications between hosts according to the organization policy, in this case, the ping communications between the bank manager’s PC and the bank server. To do this, the firewall is designed to accept rules through the REST API service, and below were commands to insert the rule:

```
curl -X POST -d '{"src-ip": "10.0.0.1/32", "dst-ip": "10.0.0.7/32", "dl-type":"ARP" }' http://localhost:8080/wm/firewall/rules/json

curl -X POST -d '{"src-ip": "10.0.0.7/32", "dst-ip": "10.0.0.1/32", "dl-type":"ARP" }' http://localhost:8080/wm/firewall/rules/json


```

The “POST” method, will insert the rules into the firewall rules file. Each incoming packet will be compared against this rules and if a match is found the rule’s action (allow /deny) is stored in the IRoutingDecision object and passed on to the packet forwarding object to push the flow entry if the decision is ALLOW, or drop flow if the decision is DENY, for this scenario it ALLOW the packet-in flow entry to be pushed.
d. Implementing LOAD BALANCER component

To commence with load balancing while firewall is running and apart from the rule we previously inserted, there is no other rule configured in the firewall, hence traffic flow will not be possible, the system has to terminate firewall.

Entering the command below into a terminal will disable firewall.

curl http://localhost:8080/wm/firewall/module/disable/json

The simple load balancer application module is implemented to ping, tcp and udp services. The ping service is used here.

Load balancing is simply intelligent routing to avoid traffic congestion as well as to properly utilize network resources. Load balancer works with set of criteria, the client requesting the service, the service, the pool (clusters of server where the services are located) and the virtual server.

The system through REST API query attached devices, to confirm Floodlight has learned where each exists. So also, routing service will be able to compute route for the load-balanced packets traversing the network.

The LoadBalancer module will load balance the ICMP packets between hosts 3(10.0.0.3) and host 4 (10.0.0.4) (BTW "port":"8" signifies ICMP packets internal to the LoadBalancer module) as shown below:

curl -X POST -d '{"id":"1","name":"vip1","protocol":"icmp","address":"10.0.0.100","port":"8"}' http://localhost:8080/quantum/v1.0/vips/
These commands creates a virtual server with IP address 10.0.0.100 and create a pool and add members (10.0.0.3 and 10.0.0.4) of the network to the pool as servers that will render the service needed by the requesting client. After the load balancer is configured, h1 pinged 10.0.0.100 (PS: 10.0.0.100 does not exist in our network, but is a proxy (virtual server) for the load-balanced hosts 10.0.0.3 and 10.0.0.4):

```
curl -X POST -d '{"id":1,"name":"pool1","protocol":"icmp","vip_id":"1"}' http://localhost:8080/quantum/v1.0/pools/
```

```
curl -X POST -d '{"id":1,"address":"10.0.0.3","port":8,"pool_id":"1"}' http://localhost:8080/quantum/v1.0/members/
```

```
curl -X POST -d '{"id":2,"address":"10.0.0.4","port":8,"pool_id":"1"}' http://localhost:8080/quantum/v1.0/members/
```

The initial delay is when the LoadBalancer module inserted the necessary flows for each switch along the way. The route taken is h1-->s3-->s2-->s4-->h3 (10.0.0.3). The flows inserted by the LoadBalancer module can be seen by requesting flows of each switch (outside mininet in another terminal window).

Then, h2 pinged 10.0.0.100. The LoadBalancer module redirect the ICMP packets to 10.0.0.4, so the route will be h1-->s3-->s2-->s4-->h4 (10.0.0.4) instead.
Fig 10: Ping command from host H2 to virtual server 10.0.0.100, Load Balancer directing the load to an available server (10.0.0.4).

From the result of the implementation, the messages exchange between switch and the OpenFlow protocol in SDN architecture is observed to clearly understand how it works and how to use it for control. It also showed that the real environment is not a must, to commence design and prototyping applications for networks, virtual networks can be useful for coding, testing and applying innovative applications.

5. Conclusion

Network configuration and management is quite a daunting task, the rigidity of the current network architecture is adding to the ordeal network operators faces. The paradigm, Software defined networking has presented an alternative. Seeing a lot of exploration from both academic and industry. You can tell by the amount of industrial buy-in SDN has seen by networking corporations like Cisco, HP, etc, developing SDN capable switches for programmable networking architecture.

This paper has sufficiently presented SDN and OpenFlow as a standard. Opened up the Mininet emulator and show how it helps researchers to develop and test their applications in a virtual scenario for gradual deployment into production networks. In this paper, the Controller as the main component of SDN was discussed. In the implementation scenario, the floodlight controller in eclipse was implemented and displayed the network in GUI. This is the first paper, which has the GUI network implementation. The result of the implementation was analyzed to show how communication between hosts.

SDN is still evolving, with so many areas still in need of research to be able to build a network we can full interact with and innovate on in the future. We intend to extend this work in the future to see how more networks can be connected to each other for routing component and observe the communication between hosts of different networks.
Reference


