OpenFlow: Load Balancing in enterprise networks using Floodlight Controller

Srinivas Govindraj, Arunkumar Jayaraman, Nitin Khanna, Kaushik Ravi Prakash
srinivas.govindraj@colorado.edu,
arunkumar.jayaraman@colorado.edu, nitin.khanna@colorado.edu,
kaushik.p.rakash@colorado.edu

A capstone paper submitted as partial fulfillment of the requirements for the degree of Masters in Interdisciplinary Telecommunications at the University of Colorado, Boulder, May 4, 2012. Project directed by Professor Dirk Grunwald.

1 Introduction

OpenFlow is a switching protocol that is based on the concept of Software Defined Networks (SDN). A switch is a computer network device that facilitates communication between connected hosts in a Local Area Network (LAN). The typical architecture of a generic switch comprises of a control plane and a forwarding plane. The control plane is where the decision to forward a set of packets along a certain port is taken while the forwarding plane does the forwarding of data based on the above decision. Traditionally, switches combine both the planes in the same device and this leads to a non-scalable switching solution. SDN seeks to change this, by dividing the control and forwarding planes into separate entities and running them on separate devices. By doing this, SDN allows network administrators to have control over the whole network from a central location as well as be able to divide the network resources into various sections for specific use. This also allows the network administrator to have control over the whole network from a central location as well as be able to divide the network resources into various sections for specific use.

The OpenFlow specification is composed of two components namely, the OpenFlow controller and the OpenFlow switch. An OpenFlow controller is a software package that takes the forwarding decisions by analyzing the first packet of every flow based on rules defined by the administrator. An OpenFlow switch can be a generic computer running one of the many software based switching packages available in the market or it may be a dedicated hardware running OpenFlow as a protocol with both controller and forwarding plane sitting in the same device. The difference between the two approaches is that, while in the former, the controller of the switch is a separate device, in the latter, the same device holds the controller, but that controller is capable of controlling other OpenFlow switches as well. A third possibility is a set of OpenFlow based devices running the switching package and passing flow information to a distributed cluster of computers. This is a more realistic view of an OpenFlow based switching network because it has often been said that, more the number of flows that pass through a network, the larger the computing power of the controller hardware needs to be. Thus, a cluster of hardware running the controller applications seems like the most scalable way to run OpenFlow in a large production environment. Because of OpenFlow’s advantages, it is being touted as the next big way to solve the various issues that plague large networks [1].
2 Research Question

The primary research question of this paper is to determine whether the OpenFlow protocol can support load balancing of traffic meant for production network environments. The secondary research question is to identify whether it is beneficial for enterprise networks to move to a centralized OpenFlow switching environment as opposed to a distributed switching network that is currently used by all networks. The authors will compare the load balancing metrics of OpenFlow with respect to source MAC address and ingress port traffic classification. From this research, the authors will also try to determine whether the OpenFlow controller and the OpenFlow switches are a good substitute for legacy load-balancers used in production networks like campus networks.

3 Definitions

CBench – A tool designed for benchmarking OpenFlow controllers. The benchmarking criterion is based on the amount of flows per second that can be sent out by the controller, under varying test environments.

Floodlight - An OpenFlow controller used in this project to co-ordinate the flow inputs and the model. ‘Floodlight’ is an open source, Java based, Apache-licensed OpenFlow Controller, developed by David Erickson and a community of developers.

Iperf - An open source performance measuring tool used to test the bandwidth, throughput between two hosts such that one host acts as a server and the other as a client. The performance metrics can be measured either with TCP or UDP packets.

Mininet - Mininet is a network simulator used to create scalable SDNs using Linux processes. Mininet is used in this project to simulate the topology and test the traffic flows. A Python script is used to create the topology in Mininet and the traffic flows are received from a remote OpenFlow controller.

OpenFlow – A computer networking protocol that can be run on generic hardware, software and vendor specific devices. This protocol allows separating the control and data paths from the same device to different devices, thus giving more control over the flow of data through the network [2].

OFP - The OpenFlow protocol packets, which are exchanged between the controller and the Open Vswitches. Each OpenFlow packet has a header and flow modification fields. The flow modification field has the match type information for classification of user based on source MAC address or ingress ports or Virtual LAN (VLAN) identifier.

4 Scope and Assumptions

The authors will, for the purposes of experimentation, setup a sample network consisting of three to five OpenFlow switches and Floodlight. The switches in turn will be connected to personal computers or traffic generators. The assumption of the research is that the authors will be able to closely emulate a production network. The research will be limited to studying the benefits of load balancing using OpenFlow in a lab-emulation of a production network. Despite the best efforts of the authors, emulation of a real production network will not be possible to a
high degree of perfection and this will be a limiting factor to the research. The results from this study can then be extended to real production networks. Examples of such networks are University networks, Enterprise networks and networks in data centers [3]. The intended audience of this paper will be the university authorities or other researchers interested in setting up OpenFlow in their networks to boost innovation and experimentation.

5 Importance and prior work

This question is significant as it explores the use OpenFlow for research in campus networks such as a University network. The question was first addressed in Stanford University where OpenFlow has been deployed in a part of the Computer Science department’s current network [4]. What distinguishes this paper from that deployment is that OpenFlow originated from Stanford University and this paper will explore the deployment of OpenFlow in an environment where support is provided to deploy the network with no initial backing from the University. Thus the researcher will need to convince the University about the benefits of OpenFlow in order to be allowed to deploy it. The networking industry at large is looking for more support for OpenFlow outside of the current deployments and companies supporting it [5]. The analysis from this paper can also be used by enterprises that wish to determine whether OpenFlow can be deployed for packet forwarding and load balancing applications in their production networks.

6 Research Methodology

The questions that have been put forth by this paper are both qualitative and quantitative in nature. The qualitative research is to determine the feasibility of using OpenFlow switches, and the quantitative research is to measure the load balancing capabilities of OpenFlow devices against traditional ones. The researchers will be deploying a test network of OpenFlow switches and an OpenFlow controller and will benchmark its load balancing performance. The statistics that need to be looked at are the load-balancing throughput across the network, the cost of traditional equipment versus OpenFlow equipment and the scalability of each network in terms of time needed to add resources to expand the network. Most of these questions can be answered quantitatively whereas the resulting analysis will be qualitative in nature.

An important limiting factor of this research is the size of the network. While real production networks may consist of large number of switching nodes, a lab based network will only be a fraction of that, leading to test results that may be misleading when thinking about the scale of the network. Another factor that needs to be taken into consideration is the whether the traditional and the OpenFlow networks will be comparable in nature. Most switches available in the lab have only a few Gigabit ports and mostly Megabit ports whereas the devices running OpenFlow will mostly have Gigabit ports. Thus, the authors will need to come up with a standard size that will be used as the definition of a single unit of the network.

6.1 Easy script usage

The Mininet software package includes examples of scripts that extend the Mininet platform and make it easy to use. While studying OpenFlow, we came across the issue of
backward compatibility for network engineers. The OpenFlow project has not grown much beyond its laboratory use right now and that makes it difficult for current network engineers who have very little understanding of software to understand and implement OpenFlow in their labs. In order to create an easy method of designing a network in Mininet, we created a set of scripts based on some of the examples listed in the Mininet source code. The miniedit.py script by Bob Lantz [6] functions simply allows users to create a topology and run it in real time in Mininet. The missing functionality is allowing a user to save the topology and to create a network based on the user’s requirements. We have modified the above script to include that functionality.

Since most of the Mininet implementation is in Python, we used a module called ConfigObj to allow the user to create a `.py` file that holds custom network topology. This allows users who do not have any scripting knowledge to use the GUI provided by the script to design a network topology, save it in a file and use it with Mininet quickly and easily. The following in an example of a topology we created using the Miniedit script.

![Figure 1: Mininet Topology](image)

When we save the above topology, a python file is created using the format that Mininet understands. The file generated for the above topology is shown in Appendix A.

After we tackled the problem of making an easy to use GUI for network engineers, the next step was to control the switches according to flows. As defined earlier, flows are sent from the controller to the switches to define the paths that data must take in the network. Each flow defined corresponds to a data path. The OpenFlow documentation defines a flow using the switch MAC address, an ingress port and an action command that decides from which port the data will egress. We created a script – flowmaker.py that lets the users define a set of flows based on the above values. The users are given the option of selecting the switches and the rules through which they wish to define their flows across the network.

By creating the two mentioned utilities, we not only designed a user-friendly network topology generator but also helped in bringing OpenFlow closer to its current and prospective users. We submitted the code we created for peer review on the Mininet mailing list and will be including it in the source code of the Mininet module if approved.
6.2 Network Description:

This section describes the network setup for the load balancing of host traffic across Open Vswitches. Figure 2 shows five Open Vswitches linked in a closed fashion and two hosts connected to each of these switches. The network topology was simulated in Mininet software. C0 in figure 2 represents the floodlight controller which pushes the flows onto each individual switches. The controller’s Internet Protocol (IP) address is 10.0.2.2 and the interface connecting the Mininet switches is 10.0.2.15.

![Network View](image)

Figure 2: Network View

Figure 3 illustrates two paths to reach switch 6 (S6) from switch 2 (S2). The first path is along S2->S3->S6 and the second along S2->S10->S8->S6. Host 1 (H1) and Host 5 (H5) are categorized as gold customers with the requirement for premium service and H11 and H12 as silver customers with the requirement for best effort service. The test case involved the load balancing of user traffic from hosts H1 and H11 on S2 to hosts H5 and H12 on S6. Hence packet from H1 switches via path 1 (least resistant path) and packets from H11 via path 2. The user traffic was generated using continuous ping packets and we used Wireshark to trace the OpenFlow packets.
The test results showed that H1 pings H5 only via path 1 and H11 pings H12 only via path 2. The test also showed that H1 does not ping H11 unless there was a flow defined in the controller. The load balancing was achieved based on the classification of host’s source MAC address and ingress ports of Open Vswitches. For the traffic classification based on source MAC address and ingress ports, six flows were defined for the successful ping between H1 and H5. Similarly eight flows were defined for the ping between H11 and H12. The test results showed that two flows were required per switch.

Figure 4 shows the OFP header extracted using Wireshark tool. The trace result shows six “Flow Mod”, in accordance with the six flows injected from the controller to the Mininet switches. The trace result also confirms the match type as “Ethernet src Address”.

Figure 3: Traffic Paths
The aforementioned test results conclude that the load balancing of user traffic on S2 was successful with classification based on ingress and source MAC address. Considering the classification of traffic based on ingress-based flows, for H1 ping H5, the flow for S2 describes that a packet received on port 1, forward onto port 2.

7 Performance Analysis

This section provides the results of the performance of Open Vswitches and the Floodlight controller. The performance of the OpenFlow controller and the Open Vswitches generated by Mininet are limited by three major factors namely, the underlying Operating System (OS), the allocated memory heap size and the available processing memory. In order to gauge the load balancing performance of the OpenFlow controller over Open Vswitches, we tested the TCP throughput across a set of hosts and Open Vswitches that were simulated using Mininet as shown in Figure 2.

7.1 Performance measurement

The performance of the Floodlight controller was measured using the CBench benchmarking tool. The Floodlight controller is a java based OpenFlow controller that is designed to run over standard operating systems. In order to optimize the controller performance without affecting the OS performance, we allocated a java heap size between 512MB and 1024MB. The Floodlight controller was tested over two standard operating systems namely,
Windows 7 and Ubuntu 11.04. To gauge the role of memory utilization in measuring the performance, we performed two tests over the Linux OS. The first test contained the entire load balancing controller packages while the second test contained the bare essential load balancing controller packages. The throughput performance of the Open Vswitches were conducted using Iperf over the two load balancing paths along the set of simulated hosts and switches as described in section 2.1.

7.2 Analysis and comparisons

Transport Control Protocol (TCP) throughput tests were conducted for the two load-balancing paths that represented gold and silver categories. Additionally two kinds of controller load balancing techniques namely ingress based flows and source-MAC address based flows were introduced for each path to gauge the overall TCP throughput performance based on the controller load balancing techniques. Prior to conducting the experiments we expected that the gold path displayed better TCP throughput than the silver path for both the load balancing techniques.

![Figure 5: Source-MAC address based flows](image)

![Figure 6: Ingress port based flows](image)

From the Figures 5 and 6, it can be observed that the gold path represented by Path1 provided better throughput characteristics over the silver path represented by Path2. Comparing the results from Figure 5 and Figure 6, we determined that ingress port based flows provided better throughput over source-MAC based flows. However we cannot conclude that the ingress port based flows is the desired load balancing technique. Ingress port based flows introduce
isolated one-to-one host communication channels and cannot scale well as it does not support one-to-many host communication channels. The source-MAC based flows introduce one-to-many host communication channels. One drawback of the source-MAC based flows is that they introduce slight performance bottlenecks by checking and processing both the ingress ports and the source-MAC addresses of the incoming frames. This resulted in a lower throughput as shown in Figure 5. However with better processors and available CPU Memory this bottleneck can be fixed.

![Figure 7: Floodlight Controller Performance](image)

![Figure 8: Controller performance in Linux OS based on error length](image)

Figure 7 provides controller throughput performance results. From the Figure 7, we can observe that the controller in a Linux OS environment performed better than the controller in a Windows OS environment. Therefore we can conclude that the Linux environment is more suitable for achieving higher controller performance in generic systems.

We expected that the controller with the essential load balancing packages fared better than the controller with the complete load balancing packages. However, from the Figure 7 we can observe that the controller performance with the bare essential load balancing packages provided similar throughput performance metrics against that of a controller with the complete
load balancing packages in a Linux environment. To derive a result from this uncertainty, we performed an error test on the standard deviation of the flows per second. From the Figure 8, we can observe that the controller with the bare essential load balancing packages has a smaller error length than the controller with the complete load balancing. From the Figure 7, we can also observe that the controller with performance tuning provided better average flows per second than the controller without performance tuning. With this, we can conclude that the controller with the essential load balancing packages performed better than the controller with all the load balancing packages.

8 Conclusion

After studying OpenFlow using the various tools, we have come to the conclusion that OpenFlow based switch networks are a good replacement for legacy switching networks that depend on a distributed network of switches that must be programmed individually. We concluded that the load balancing capabilities of the OpenFlow based switches exceed the legacy switches because those proprietary switches do not provide any load balancing. This capability is an important part of OpenFlow as it helps to differentiate and gain an edge over the currently available systems. During our experiments, we observed that using OpenFlow based switches helped reduce bandwidth wastage due to the efficient use of all links in a loop free topology. This allows for better utilization of switch ports in a production networks such as data centers and enterprises.

Another conclusion we came across is that the tools in the OpenFlow ecosystem can quickly and easily be extended to provide support and familiarity to the current set of network engineers who are not familiar with OpenFlow or Mininet. By extending those tools ourselves, we learned that OpenFlow has a long way to go before it comes completely in sync with current standards but that it is very easy to do so because of the availability of Application Program Interface’s (API’s) that can be used to build on top of the OpenFlow stack. This is important because the expectation of current industry professionals is that the new systems that are developing will not be very difficult to learn when compared to the systems they are already using.

Finally, we realized that there is a lot of functionality that can be implemented in OpenFlow to extend its usability in production networks. For example, the ability to load balance traffic using weighted round robin scheduling is something we would like to implement if we get the opportunity to extend our research onto a physical network of Open Vswitches instead of using Mininet.
References:


