Evaluating OpenFlow Controller Paradigms

Marcial P. Fernandez
Universidade Estadual do Ceará (UECE)
Av. Paranjana 1700
Fortaleza - CE - Brazil
marcial@larces.uece.br

***Abstract*—The OpenFlow architecture is a proposal from the Clean Slate initiative to define a new Internet architecture where the network devices are simple, and the control and management plane is performed by a centralized controller. The simplicity and centralization architecture makes it reliable and inexpensive, but the centralization causes problems concerning controller scalability. An OpenFlow controller has two operation paradigms: reactive and proactive. The performance of both paradigms were analyzed in different known controllers. The performance evaluation was done in a real environment and emulation. Different OpenFlow controllers and distinct amount of OpenFlow devices were evaluated. The analysis has demonstrated the shortcoming of reactive approach. In conclusion, this paper indicates the effectiveness of a hybrid approach to improve the efficiency and scalability of OpenFlow architecture.**

***Keywords*-Openflow; OpenFlow Controller; Performance evaluation.**

I. INTRODUCTION

The OpenFlow [1] architecture is a proposal from the Clean Slate initiative to define an open protocol that sets up forward tables in switches. It is the basis of the Software Defined Network (SDN) architecture, where the network can be modified by the user. This proposal tries to use the most basic abstraction layer of the switch, it is the definition of forward tables, in order to achieve better performance. The OpenFlow protocol can set a tuple of condition-action assertion on switches like forward, filter and also, count the packets that matches the condition. The network management is performed by the OpenFlow Controller maintaining the switches simple, only with the packet forwarding function.

The OpenFlow architecture provides several benefits: (1) OpenFlow centralized controllers can manage all flow decisions reducing the switch complexity; (2) a central controller can see all networks and flows, giving global and optimal management of network provisioning; and (3) OpenFlow switches are relatively simple and reliable, since forward decisions are defined by a controller, rather than by a switch firmware [2]. However, OpenFlow couples two characteristics: a central controller and simple devices, that result in scalability problems.

As the number of OpenFlow switches increases, relying on a single controller for the entire network might not be feasible for several reasons: (1) the amount of control messages destined to the centralized controller grows with the number of switches; (2) with the increase of network diameter, some switches will have longer setup delay, independently where the controller is placed [2]; and, (3) since the system is bounded by the controller processor power, setup times can grow significantly when the number of switches and the size of the network grow.

Two paradigms were implemented on some OpenFlow controllers: the NOX/C++ Controller [3], the POX/Python Controller [3], the Trema/Ruby Controller [4] and the Floodlight/Java Controller [5]. In each controller and device the reactive and proactive approaches were implemented and they were evaluated.

The rest of the paper is structured as follows. In Section II, we present some related work. Section III introduces the OpenFlow architecture fundamentals; and in Section IV, we present the evaluation methodology and tests. Section V shows the results and Section VI concludes the paper.

II. RELATED WORK

Tootoonchian and Ganjali proposed the HyperFlow [6], a mechanism that changes the controller paradigm from centralized to distributed. HyperFlow tries to provide scalability, using as many controllers as necessary to reach it, but keeping the network control logically centralized. The proposal uses a publish/subscribe messaging system to propagate controller events to others, maintaining the database consistent. However, this approach can only support global visibility of rare events such as link state changes, not frequent events such as flow arrivals.

Another proposal, the DevoFlow [2], aims to deal with the scalability problem by devolving network control to switches with an aggressive use of flow wildcard and introducing new mechanisms to improve visibility. They introduced two new mechanisms to be implemented on switches: *rule cloning* and *local actions*.

The Source-Flow controller [7], proposed by Chiba, Shinohara and Shimonishi, uses a similar approach. It tries to reduce the number of flow entries using a MPLS-like tunneling approach in order do reduce the Ternary Content-Addressable Memory (TCAM) used space.

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-245-5

In this paper, we evaluate some OpenFlow controller's performance working in reactive and proactive approach. Then, we propose a new controller architecture performing a hybrid approach, making better use of both paradigms.

III. OpenFlow Architecture

The OpenFlow architecture has several components: the OpenFlow controller, the OpenFlow device (switch), and the OpenFlow protocol. Figure 1 shows the components of an OpenFlow architecture. The OpenFlow approach considers a centralized controller that configures all devices. Devices should be kept simple in order to reach better forward performance and the network control is done by the controller.

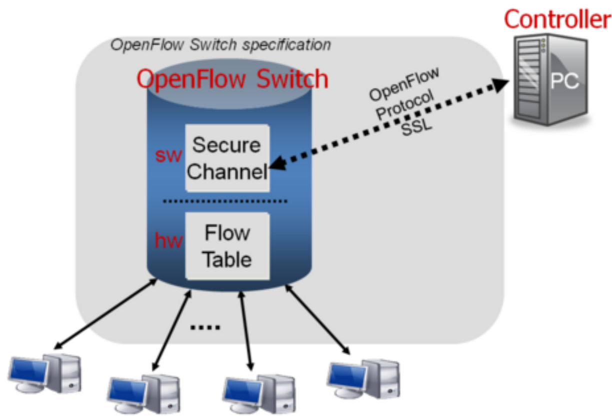

Figure 1. The OpenFlow architecture [1]

The OpenFlow Controller is the centralized controller of an OpenFlow network. It sets up all OpenFlow devices, maintains topology information, and monitors the overall status of entire network. The OpenFlow Device is any OpenFlow capable device in a network such as a switch, router or access point. Each device maintains a Flow Table that indicates the processing applied to any packet of a certain flow. The OpenFlow Protocol works as an interface among the controller and the switches setting up the Flow Table. The protocol should use a secure channel based on Transport Layer Security (TLS).

The controller updates the *Flow Table* by adding and removing Flow Entries using the OpenFlow Protocol. The Flow Table is a database that contains Flow Entries associated with actions to command the switch to apply some actions on a certain flow. Some possible actions are: forward, drop and encapsulate.

Each OpenFlow device has a Flow Table with flow entries as shown in Figure 2. A Flow Entry has three parts: Rule, Action and Statistics. The Rule field is used to define the match condition to a specific flow; Action field defines the action to be applied to this flow, and Stat field is used to count the rule occurrence for management purposes. When a packet arrives to the OpenFlow Switch, it is matched against

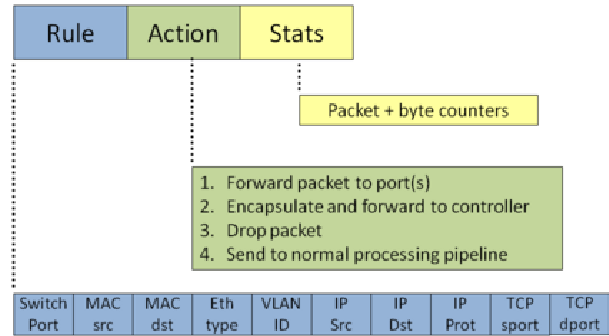

Figure 2. The OpenFlow Flow Entry [8]

Flow Entries in the Flow Table. The Action will be triggered if the flow Rule is matched and then, the Stat field is updated. If the packet does not match any entry in the Flow Table, it will be sent to the Controller over a secure channel to ask for an action. Packets are matched against all flow entries based on some prioritization scheme. An entry with an exact match (no wildcards) has the highest priority. Optionally, the Flow Table could have a priority field (not shown in figure) associated with each entry. Higher number indicates that the rule should be processed before.

The Openflow Controller presents two behaviors: reactive and proactive. In the **Reactive** approach, the first packet of flow received by switch triggers the controller to insert flow entries in each OpenFlow switch of network. This approach presents the most efficient use of existing flow table memory, but every new flow causes a small additional setup time. Finally, with hard dependency on the controller, if the switch loses the connection, it cannot forward the packet.

In the **Proactive** approach, the controller pre-populates the flow table in each switch. This approach has zero additional flow setup time because the forward rule is defined. Now, if the switch loses the connection with controller, it does not disrupt traffic. However, the network operation requires a hard management, e.g., requires to aggregate (wildcard) rules to cover all routes.

The OpenFlow Protocol uses the TCP protocol and port 6633. Optionally, the communication can use a secure channel based on TLS. The OpenFlow Protocol supports three types of messages [8]:

1) Controller-to-Switch Messages: These messages are sent only by the controller to the switches; they perform the functions of switch configuration, modifying the switch capabilities, and also manages the Flow Table.

2) Symmetric Messages: These messages are sent in both directions reporting on switch-controller connection problems.

3) Asynchronous Messages: These messages are sent by the switch to the controller to announce changes in the network and switch state. All packets received by the switch are compared against the Flow Table. If the packet matches

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-245-5

any Flow Entry, the action for that entry is performed on the packet, e.g., forward a packet to a specified port. If there is no match, the packet is forwarded to the controller that is responsible for determining how to handle packets without valid Flow Entries [8].

It is important to note that when the OpenFlow switch receives a packet to a nonexistent destination in the Flow Table, it requires an interaction with the controller to define the treatment of this new flow. At least, the switch will need to send a message to the controller with regards to the new packet received (message Packet-In). If the path is already predefined (there is an entry in Flow Table), this procedure is not necessary, reducing the amount of messages exchanged through the network and reducing the processing at the controller.

Furthermore, the maintenance of unused Flow Entries in the switch Flow Tables requires fast TCAM memory. Therefore it is necessary to remove unused flows using a time-out mechanism. If a flow previously excluded by time-out restarts, it is necessary to reconfigure all switches on the end-to-end path.

A. OpenFlow Device

An OpenFlow device is basically an Ethernet switch supporting OpenFlow protocol. But there are different implementation approaches: OpenFlow-enabled switch and OpenFlow-compliant switch.

The OpenFlow-enabled switch uses off-the-shelf hardware, i.e., traditional switches with OpenFlow protocol that translate the rule according to the hardware chipset implementation. The OpenFlow-enabled switch re-uses existing TCAM, that in a conventional switch has no more than only few thousands of entries for IP routing and MAC table. Considering that we need at least one TCAM entry per flow, in a current hardware, it would not be enough for production environments. The Broadcom chipset switches based on Indigo Firmware [9], e.g., Netgear 73xxSO, Pronto Switch and many other, are example of this approach.

The OpenFlow-compliant switch uses specific network chipset, designed to provide better performance to OpenFlow devices. The OpenFlow philosophy relies on matching packets against multiple tables in the forwarding pipeline, where the output of one pipeline stage is able to modify the contents of the table of next stage. Some example are devices based on the EZChip NP-4 Network Processor [10]. But, nowadays, there are few commercial OpenFlow-compliant switches, one example is the NoviFlow Switch 1.1 [11].

B. OpenFlow Controller

All functions of the control and management plane are performed by the controller. It has full network topology information and the location of hosts and external paths (MAC and routing tables). When a switch receives a packet in which there is no entry in its Flow Table, it forwards the message to the controller asking for the action to take upon this new flow. The controller can define the port that the flow must be forwarded to or take other actions, such as dropping the packet. The controller must set the entire path by sending configuration messages to all switches from the source to the destination.

Scalability and redundancy are possible using a stateless OpenFlow control, allowing simple load-balancing over multiple devices [1]. Due to the OpenFlow centralized architecture, controller scalability issues have received attention by researchers. Many authors focus on distributed architectures to improve the scalability problem.

The most common OpenFlow controller operation mode is the reactive. In the reactive mode, the controller listens to switches passively and configures routes on-demand. It receives messages of connected hosts from the switches and treats ARP message from hosts in order to maintain a global MAC table. Upon receiving an ARP message, the controller looks for the destination host location and sets the path by sending OpenFlow messages to affected switches. After a time out, an unused entry is excluded from the table. The reactive behavior allows a more efficient use of memory (MAC table) at the cost of the re-establishing path later, if necessary. In the proactive mode, paths are set up in advance. This comes at the cost of lower memory space efficiency and the requirement for a priori setup of all paths.

The controller performance is a central issue of an OpenFlow architecture. A controller can only support a limited number of flow setups per second. In the former Ethane experiment [12] with similar approach, i.e., only one controller sets many switches, each controller could support 10K new flows per second. Another work, Tavakoli et al. [13] shows that one NOX controller can handle a maximum of 30K new flow installs per second maintaining a flow install time below 10 ms. From the network side, Kandula et al. [14] measure on a 1500-server cluster datacenter the creation of 100K new flows per second, implying a need for, at least, four OpenFlow controllers. As the OpenFlow philosophy relies on a single controller, we can question the feasibility of OpenFlow use in (not so big) production datacenter.

Several approaches have proposed new OpenFlow Controller architectures and implementations. One of the first approaches was the NOX Controller [3], a centralized controller that implements a reactive and proactive approach. The NOX default operation mode is the reactive, but offer an API to allow users to set flows in a proactive mode. The NOX framework is used as the basis for many controller's implementation, for example, the POX controller. POX controller is a pure Python controller, redesigned to improve the performance compared to original Python NOX. The former NOX was redesigned to provide a pure C++ controller.

The Floodlight [5] is Java-based OpenFlow Controller, forked from the Beacon controller developed at Stan-

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-245-5

ford. The Floodlight controller is an open-source software Apache-licensed, supported by a community of developers. It offers a modular architecture, easy to extend and enhance.

The Trema [4] is an OpenFlow controller framework developed in Ruby and C. It is basically a framework, including basic libraries and functional modules that work as an interface to OpenFlow switches. Several sample applications are provided to permit execution of different controllers, making easy the extension to new features.

IV. Evaluating OpenFlow Controller's Paradigm

To evaluate the OpenFlow controller's performance, two scenarios were built: (1) a real network with Netgear GSM-7328SO switches and (2) a virtualized network using Mininet [15]. In each scenario, a host generates traffic to cross the entire network topology simulating a production network.

On the server, the OpenFlow controller under evaluation is installed. In the host, it runs the Cbench benchmark software to stress the controller's capacity. There is also another host to generate real traffic crossing the network using Mausezahn software [16]. The evaluation does not intend to compare the controller software; the main objective is to compare the performance and the behavior of each controller using the reactive and proactive approach.

For traffic generation, Mausezahn software was used [16]. Mausezahn is an open-source traffic generator written in C, which allows to send nearly any possible packet. In order to stress the OpenFlow controller, Mausezahn generates packets with one million different IP addresses, forcing the switch to send one million of Openlow requests to the controller. IP address was used instead of MAC address, to simulate a normal OpenFlow operation, where an ARP request starts the route request to the controller.

To evaluate the controller performance, CBench software was used [17]. CBench is a performance measurement tool designed for benchmarking OpenFlow controllers. The benchmarking measurement is the amount of flows per second that can be processed by the controller.

The controller code has been modified to implement the reactive and proactive behavior. A special configuration message was used to set the switches to work in a reactive or proactive way. The Indigo Firmware was also modified to receive this message and set the switch to work in reactive or proactive manner. The OpenVswitch code was changed to implement the same behavior on virtualized Mininet environment.

A. Evaluation Scenario

The tests were executed in two environment: real network and emulated network. The real network was made with Netgear GSM-7328SO switches running on modified Indigo Firmware release 2012.03.19, as shown in Figure 3.

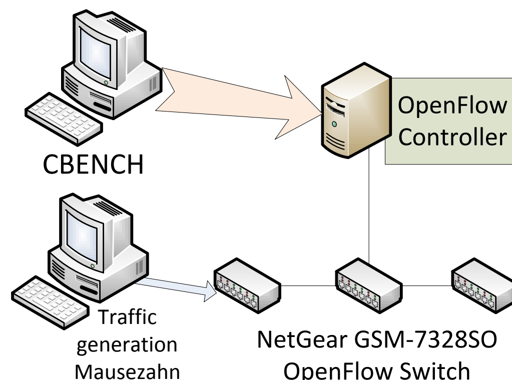

Figure 3. Real switch scenario

The second test environment chosen was the OpenFlow emulation over virtual machine using Mininet [15]. This model permits an evaluation in emulation environment using only one computer. Mininet imposes a restriction on link layer configuration, e.g., we cannot specify link bandwidth or error rate, but for the validation this environment was suitable to obtain the results.

The experiment was built over VMware Workstation 8. In the virtualized environment, Mininet was used [15]. Mininet is a network emulator used to create Software Defined Networks (SDNs) scenario in Linux environment. The Mininet system permits the specification of a network interconnecting "virtualized" devices. Each network device, hosts, switches and controller are virtualized and communicate via Mininet. The traffic flow used in this test is generated by Mausezahn. A Python script is used to create the topology in Mininet and the traffic flows setup are received from a remote OpenFlow controller.

Therefore, the test environment implements and performs the real protocol stacks that communicate with each other virtually. The Mininet environment allows the execution of real protocols in a virtual network. The virtual topology created in Mininet platform is shown in Figure 4.

B. Evaluation Procedure

To define the experiment, initially it is necessary to specify the hosts and network that will be used. The OpenFlow controller has the responsibility to define the best path to connect all hosts.

To evaluate the controller performance into the test topology the *Cbench* [17] program was included, part of the OpenFlow suite that creates and sends a large amount of OpenFlow messages to the controller in order to test its performance. These messages do not represent any real

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-245-5

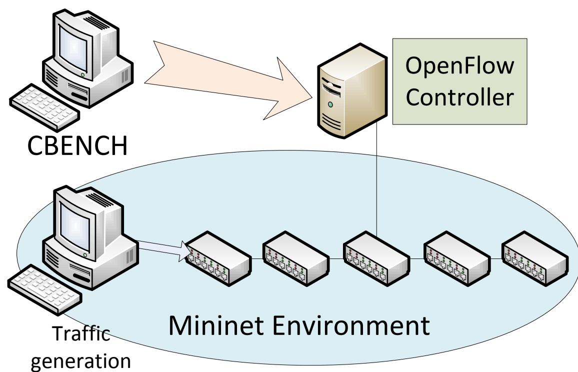

Figure 4. Virtual switch scenario

network topology, only “deceive” the controller that handles and sends messages believing it is dealing a larger network. As result, we obtain the number of OpenFlow messages the controller can support per second, besides the messages sent by real switches or virtualized switches in Mininet.

The tests with the Cbench, simulated the presence of more than 100 switches, in addition to the 50, 100 and 200 virtualized switches topology created on Mininet. In each round, 16 tests were performed, and in these tests the average and the standard deviation were calculated. Finally, the graph was plotted of average performance with 95% confidence interval. Since we are interested in studying the system in equilibrium, we do not consider the first 2 minutes of data as the warm up period.

In a normal OpenFlow network, when a host performs an ARP request to find the destination address and the switch has no such address in its Flow Table, the switch makes a query to the OpenFlow controller. The controller will decide what action would be applied to this flow, e.g., choose the path from origin to the destination to forward the packets to destination host. To optimize the memory usage in the switches, a timeout (in our experiment 10 seconds) sets the removal of this entry on its flow table, forcing the path rebuild whenever it is necessary.

In the Proactive approach, the path is already predefined for the necessary time and therefore it is not necessary to ask the controller to build the path from source to destination. The proactive approach is implemented on controller’s and switch’s code. The reduction of messages to discover a new path among switches allows the increase of controller performance, as shown on the results in Section V below.

V. RESULTS

The performance test results are shown in the following graphs. They show the performance of each controller in how many OpenFlow messages could be processed according to the amount of additional switches which were created in real scenario and in Mininet environment.

Figure 5 shows the performance of each OpenFlow controller in real switch network, shown in Figure 3. We can see that every controller, independently of architecture and programming language, has better performance in the proactive approach.

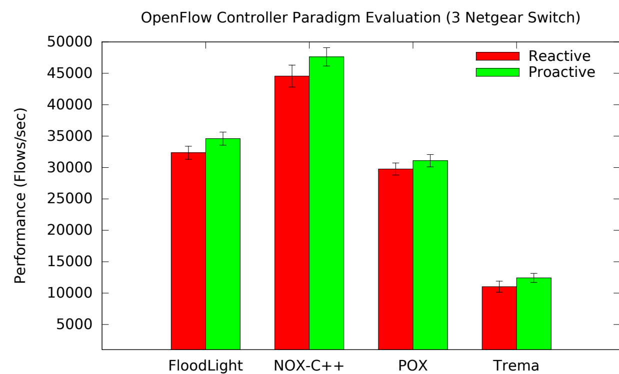

Figure 5. Netgear switch network evaluation

Figure 6 shows the performance of each OpenFlow controller in emulated Mininet environment, shown in Figure 4. We can see similar results, proactive approach gives better performance.

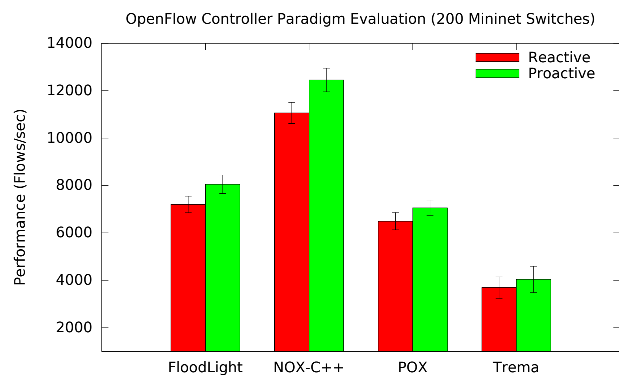

Figure 6. Mininet 200 switch network evaluation

Figure 7 shows the NOX-C++ controller performance from 3 real switches to 200 virtual switches. The performance measured by Cbench benchmark reduces according to the increase of number of switches in network.

The improvement is due to proactive controller receives less request message from switch because the path is already set. Receiving fewer messages from “real” network allows

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-245-5

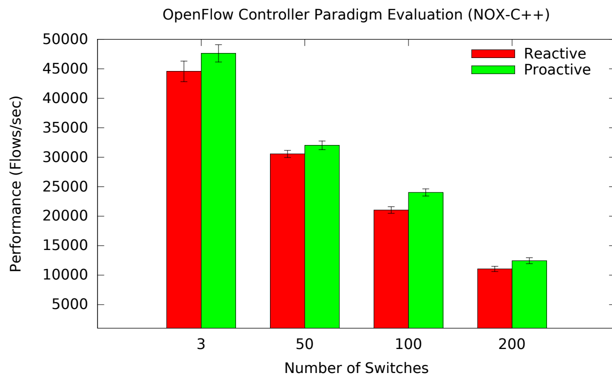

Figure 7. NOX-C++ controller evaluation

the controller to handle more "fake" messages from *Cbench* program. We can also notice that increasing the number of virtualized switches in Mininet, the performance will be reduced.

The proactive operation improves the controller performance. As the user pre-configures a path, all packets from this flow already have a Flow Entry on all switch's Flow Table, the switches do not need to send a message to the controller. Then, the reduction of messages sent by switches reduces the amount of messages received by the controller, providing capacity to controller dealing more messages from other on-demand flows.

However, the proactive approach requires the controller know the traffic flows in advanced to configure the paths before it is used. The reactive approach reduces the controller's performance but requires less configuration effort. In reactive approach, it is necessary only to configure an IP address to put the network in operation.

VI. Conclusion and Future Works

Although we consider that the OpenFlow architecture has a prominent future due to its simplicity and suitability to new technologies, its centralized architecture causes a scalability problems. This problem has been studied by researchers from several points of view.

This paper analyzes the performance of different OpenFlow controller operating in reactive and proactive approach. All evaluation's results show the increase on controller performance when it used the proactive approach. However, we agree that reactive approach makes easy on the network operation and management.

Our proposal tries to indicate a possible solution to this problem, by adding a new intelligent switch that sets the paths on-demand, improving the performance of proactive approach but maintaining the facility of reactive approach. The controller acts like a learning switch without management interference but can perform a proactive set up using Reinforcement Learning. The results show an improvement in controller performance due to the fact of reduction on control messages.

But some heuristics will be able to set the paths in advance automatically, improving the performance maintaining the simplest configuration. The use of Reinforcement Learning, proposed in Boyan and Littman [18], and also, by Peshkin and Savova [19], can define the routes without the user intervention.

As future work, it will be interesting to improve the system manageability implementing the Reinforcement Learning mechanism to set the routes based on traffic behavior. Another proposal, is to adjust the OpenFlow switch Flow Table time-out based on traffic behavior.

Acknowledgment

We would like to thank Broadcom Corporation for their support.

References

[1] 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 Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee, "Devoflow: cost-effective flow management for high performance enterprise networks," in *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets '10. New York, NY, USA: ACM, 2010, pp. 1:1–1:6.

[3] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[4] NEC, "Trema Openflow Controller," Last accessed, Aug 2012. [Online]. Available: http://trema.github.com/trema/

[5] D. Erickson, "Floodlight Java based OpenFlow Controller," Last accessed, Aug 2012. [Online]. Available: http://floodlight.openflowhub.org/

[6] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. USENIX Association, 2010, p. 3.

[7] Y. Chiba, Y. Shinohara, and H. Shimonishi, "Source flow: handling millions of flows on flow-based nodes," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 465–466, August 2010.

[8] B. Heller, "Openflow switch specification, version 1.0.0," Last accessed, Dec 2011. [Online]. Available: www.openflowswitch.org/documents/openflow-spec-v1.0.0.pdf

[9] D. Talayco, "Indigo OpenFlow Switching Software Package," Last accessed, Jun 2012. [Online]. Available: http://www.openflowswitch.org/wk/index.php/IndigoReleaseNotes

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-245-5

[10] O. Ferkouss, I. Snaiki, O. Mounaouar, H. Dahmouni, R. Ben Ali, Y. Lemieux, and O. Cherkaoui, "A 100gig network processor platform for openflow," in *Network and Service Management (CNSM), 2011 7th International Conference on*. IEEE, 2011, pp. 1–4.

[11] NoviFlow, "NoviFlow Switch 1.1," Last accessed, Sep 2012. [Online]. Available: http://www.noviflow.com/index.asp?node=2&lang=en

[12] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '07. New York, NY, USA: ACM, 2007, pp. 1–12.

[13] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to the Datacenter," in *Proceedings of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[14] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 202–208.

[15] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets '10. New York, NY, USA: ACM, 2010, pp. 19:1–19:6.

[16] H. Haas, "Mausezahn Traffic Generator Version 0.4," Last accessed, Jan 2012. [Online]. Available: http://www.perihel.at/sec/mz/

[17] R. Sherwood and K.-K. Yap, "Cbench (controller benchmarker)," Last accessed, Nov 2011. [Online]. Available: http://www.openflowswitch.org/wk/index.php/Oflops

[18] J. Boyan and M. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," *Advances in neural information processing systems*, pp. 671–671, 1994.

[19] L. Peshkin and V. Savova, "Reinforcement learning for adaptive routing," in *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, vol. 2. IEEE, 2002, pp. 1825–1830.

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-245-5