Advanced Study of SDN/OpenFlow controllers

Alexander Shalimov
Applied Research Center for Computer Networks
Moscow State University
ashalimov@arccn.ru

Dmitry Zuikov
Applied Research Center for Computer Networks
dzuikov@arccn.ru

Daria Zimarina
Moscow State University
zimarina@lvk.cs.msu.su

Vasily Pashkov
Applied Research Center for Computer Networks
Moscow State University
vpashkov@arccn.ru

Ruslan Smeliansky
Applied Research Center for Computer Networks
Moscow State University
smel@arccn.ru

ABSTRACT

This paper presents an independent comprehensive analysis of the efficiency indexes of popular open source SDN/OpenFlow controllers (NOX, POX, Beacon, Floodlight, MuL, Maestro, Ryu). The analysed indexes include performance, scalability, reliability, and security. For testing purposes we developed the new framework called hcprobe. The test bed and the methodology we used are discussed in detail so that everyone could reproduce our experiments. The result of the evaluation show that modern SDN/OpenFlow controllers are not ready to be used in production and have to be improved in order to increase all above mentioned characteristics.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.4 [**Performance of Systems**]

General Terms

Design, Measurement, Performance, Reliability, Security

Keywords

SDN, OpenFlow, Controllers evaluation, Throughput, Latency, Haskell

1. INTRODUCTION

Software Defined Networking (SDN) is the most discussed topic of networking technology of recent years [1]. It brings a lot of new capabilities and allows to solve many hard problems of legacy networks. The approach proposed by the SDN paradigm is to move network's intelligence out from the packet switching devices and to put it into the logically centralized controller. The forwarding decisions are done first in the controller, and then moves down to the overseen switches which simply execute these decisions. This gives us a lot of benefits like global controlling and viewing whole network at a time that helpful for automating network operations, better server/network utilization, and etc. Understanding all benefits Google and Microsoft recently switched to SDN in their data centers [2, 3].

In SDN/OpenFlow paradigm switches provide common and vendor-agnostic protocol for remote controller called OpenFlow [4]. This protocol provides the controller a way to discover the OpenFlow-compatible switches, define the matching rules for the switching hardware and collect statistics from switching devices. A switch presents a flow table abstraction: each flow table entry contains a set of packet fields to match, and an action (such as send-out-port, modify-field, or drop). When an OpenFlow Switch receives a packet it has never seen before, for which it has no matching flow entries, it sends this packet to the controller via a packet_in message. The controller then makes a decision on how to handle this packet. It can drop the packet, send packet to a port via a packet_out message, or it can add a flow entry directing the switch on how to forward similar packets in the future via flow_mod message.

An SDN/OpenFlow controller for networks is like an operating system for computers. But the current state of the SDN/OpenFlow controller market may be compared to early operating systems for mainframes, which existed 40-50 years ago. Operating systems were control software that provided libraries of support code to assist programs in runtime operations just as today's controllers are special control software, which interacts with switching devices and provides a programmatic interface for user-written network management applications.

Early operating systems were extremely diverse, with each vendor or customer producing one or more operating systems specific to their particular mainframe computers. Every operating system could have radically different models of command and operating procedures. The question of operating system efficiency was of vital importance and, accordingly, was studied a lot [5]. However, it was difficult to reproduce efficiency evaluation experiments applied to operating systems because of a lack of methodology descriptions, a mixture of running applications that created the workload

CEE-SECR '13, October 23 - 25 2013, Moscow, Russian Federation.

ACM 978-1-4503-2641-4/13/10 ...$15.00.
http://dx.doi.org/10.1145/2556610.2556621

in the experiments, unique hardware configurations, etc [6]. The same is true for today's SDN/OpenFlow controllers.

At present, there are more than 30 different OpenFlow controllers created by different vendors/universities/research groups, written in different languages, and using different runtime multi-threading techniques. Thus, the issue of efficiency evaluation has emerged.

The latest evaluation of SDN/OpenFlow controllers, published in 2012 [7], covered a limited number of controllers and focused on controller performance only. During the last year new controllers appeared, and the old ones were updated. This paper was intended as impartial comparative analysis of the effectiveness of the most popular and widely used controllers. In our research we treat the term "controller effectiveness" as a list of indexes, namely: performance, scalability, reliability, and security. Later on we will introduce the meaning of every index from this list.

For testing controllers we used a well known tool cbench [8]. However, we were not satisfied with the capabilities of cbench, and so we developed our own framework for SDN/Open-Flow controllers testing, which we called hcprobe [9]. We developed our own solution because the capabilities of cbench, the traditional benchmark used for controllers evaluation, cover only a small range of possible test cases. It is perfectly fitting for rough performance testing, but it does not provide the fine-tuning of test parameters. We use hcprobe to create a set of advanced testing scenarios, which allow us to get a deeper insight into the controller reliability and security issues: the number of failures/faults during long-term testing, uptime for a given workload profile, and processing malformed OpenFlow messages. With cbench and hcprobe we performed an experimental evaluation of seven freely available OpenFlow controllers: NOX [10], POX [11], Beacon [12], Floodlight [13], MuL [14], Maestro [15], and Ryu [16].

2. HCPROBE

Similar to conventional benchmarks, hcpobe emulates any number of OpenFlow switches and hosts connected to a controller. Using hcprobe, one can test and analyse several indexes of controller operation in a flexible manner. Hcprobe allows to specify patterns for generating OpenFlow messages (including malformed ones), set traffic profile, etc. It is written in Haskell and allows users to easily create their own scenarios for controller testing.

The key features of hcprobe are:

- correct OpenFlow and network stack packet generation;
- implementation in a high-level language, which makes it easier to extend;
- API for custom tests design;
- introducing an embedded domain-specific language (eDSL) for creating tests.

The architecture of hcprobe (see Figure 1):

- **Network datagram**: The library, which provides typeclass-based interface for easy creation of packets for network protocols: Ethernet frames, ARP packets, IP packets, UDP packets, TCP packets. It allows to set a custom value for any field of the header and generate a custom payload.

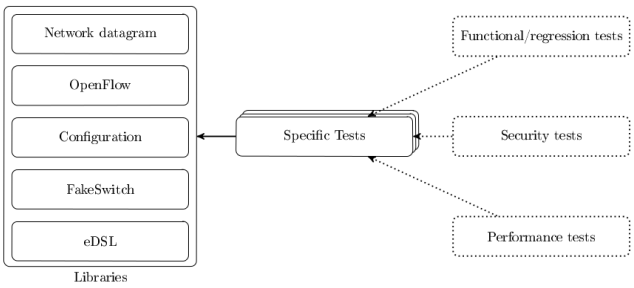

Figure 1: The hcrobe architecture

- **OpenFlow**: The library for parsing and creating OpenFlow messages.
- **Configuration**: The library that provides the routines for configuration of the test's parameters using command line options or a configuration file.
- **FakeSwitch**: Basic "fake switch", which implements the initial handshake, manages the interaction with the controller and provides callback/queue-based interface for writing tests.
- **eDSL**: Provides easier interface for creating tests.

Using these modules, it is possible to create specified tests for SDN/OpenFlow controllers, including functional, security and performance tests. The most convenient way to create tests is to use the embedded domain-specific language.

Hcprobe domain-specific language provides the following abilities:

- create OpenFlow switches;
- write programs describing the switch logic and run programs for different switches simultaneously;
- create various types of OpenFlow messages with custom field values and also generate custom payload for PacketIn messages;
- aggregate statistics on message exchange between OpenFlow controller and switches.

For a basic example, which illustrates the above features, see Figure 2.

```
1 fakeSw <- config $ do
2                 switch $ do
3                     addMACs [1..450]
4                     features $ do
5                       addPort [] [] [OFPPF_1GB_FD, OFPPF_COPPER] def
6                       addPort [] [] [OFPPF_1GB_FD, OFPPF_COPPER] def
7
8 withSwitch fakeSw "127.0.0.1" 6633 $ forever $ do
9     let payload = putDefaultEthernetFrame
10        msg     = putOFMessage $ do
11                    putOFHeader $ do
12                        putHdrType OFPT_PACKET_IN
13                        putPacketLength 42 -- Bad length!
14                    putPacketIn $ do
15                        putPacketInData pl
16        send msg
```

Figure 2: The hcprobe test example (incorrect PacketIn OF message)

So hcprobe provides the framework for creating various test cases to study the behaviour of OpenFlow controllers

processing different types of messages. One can generate all types of switch-to-controller OpenFlow messages and replay different communication scenarios, even the most sophisticated ones, which cannot be easily reproduced using hardware and software OpenFlow switches. The tool could be useful for developers to perform functionality, regression and safety testing of controllers.

3. EXPERIMENTAL SETUP

This section explains our testing methodology and technical details of experimental evaluation.

3.1 Test bed

The test bed consists of two servers connected with a 10Gb link. Each server has two processors Intel Xeon E5645 with 6 cores (12 threads), 2.4GHz and 48 GB RAM. Both servers are running Ubuntu 12.04.1 LTS (GNU/Linux 3.2.0-23-generic x86_64) with default network settings. One of the servers is used to launch the controllers. We bind all controllers to the cores of the same processor in order to decrease the influence of communication between the processors. The second server is used for traffic generation according to a certain test scenario. We use two server setup instead of a single server configuration where both a controller and a traffic generator run on the same server. Our previous measurements have shown that the throughput of internal traffic is rather unpredictable and unstable, with throughput surging or dropping by an order. Also, using two servers we can guarantee that traffic generation doesn't affect the controller operation.

3.2 Controllers

Based on the study of available materials on twenty four SDN/OpenFlow controllers, we chose the following seven open source controllers:

- NOX [10] is a multi-threaded C++-based controller written on top of Boost library.
- POX [11] is a single-threaded Python-based controller. It is widely used for fast prototyping of network applications in research.
- Beacon [12] is a multi-threaded Java-based controller that relies on OSGi and Spring frameworks.
- Floodlight [13] is a multi-threaded Java-based controller that uses Netty framework.
- MUL [14] is a multi-threaded C-based controller written on top of libevent and glib.
- Maestro [15] is a multi-threaded Java-based controller.
- Ryu [16] is Python-based controller that uses gevent wrapper of libevent.

Initially we included in our study Trema [18], Flower [17] and Nettle [19] controllers. But after several experiments we understood that they were not ready for evaluation under heavy workloads.

Each tested controller runs an L2 learning switching application provided by the controller. There are several reasons for this choice. It is quite simple and at the same time representative. It uses all of the controller's internal mechanisms, and also shows how effective the chosen programming language is by implementing single hash lookup.

We relied on the latest available sources of all controllers dated April, 2013. We run all controllers with the recommended settings for performance and latency testing, if available.

4. METHODOLOGY

Our testing methodology includes performance and scalability measurements as well as advanced functional analysis such as reliability and security. The meaning of each term is explained below.

4.1 Performance/Scalability

Performance of an SDN/OpenFlow controller is defined by two characteristics: throughput and latency. The goal is to obtain maximum throughput (number of outstanding packets, flows/sec) and minimum latency (response time, ms) for each controller. The changes in this parameters when adding more switches and hosts to the network or adding more CPU cores to the server where the controller runs show the scalability of the controller.

We analyse the correlation between controller's throughput and the following parameters:

1. the number of switches connected to the controller (1, 4, 8, 16, 64, 256), having a fixed number of hosts per switch (10^5);
2. the number of unique hosts per switch (10^3, 10^4, 10^5, 10^6, 10^7), having a fixed number of switches (32).
3. the number of available CPU cores (1–12).

The latency is measured with one switch, which sends requests, each time waiting for the reply from the controller before sending the next request.

All experiments are performed with cbench. Each cbench run consists of 10 test loops with the duration of 10 secs. We run each experiment three times and take the average number as the result.

The scripts with the methodology realization can be found in the repository: https://github.com/ARCCN/ctltest.

4.2 Reliability

By reliability we understand the ability of the controller to work for a long time under an average workload without accidentally closing connections with switches or dropping OpenFlow messages from the switches.

To evaluate the reliability, we measure the number of failures during long-term testing under a given workload profile.

The traffic profile is generated using hcprobe. We use flow setup rate recorded in Stanford campus in 2011 as a sample workload profile (Figure 3). This profile stands for a typical workload in a campus or office network during a 24-hour period: the highest requests rate is seen in the middle of the day, and by night the rate decreases.

In our test case we use five switches sending OpenFlow PacketIn messages with the rate varying from 2000 to 18000 requests per second. We run the test for 24 hours and record the number of errors during the test. By error we mean either a session termination or a failure to receive a reply from the controller.

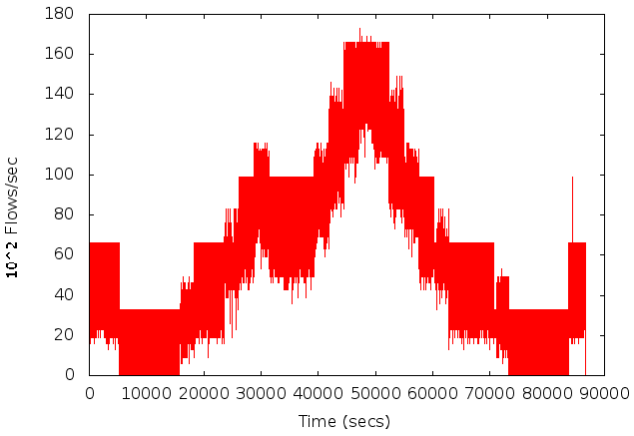

Figure 3: The sample workload profile

4.3 Security

To identify any security issues, we have also examined how the controllers manipulate malformed OpenFlow messages.

In our test cases we send OpenFlow messages with incorrect or inconsistent fields, which could lead to a controller failure.

4.3.1 Malformed OpenFlow header.

Incorrect message length. PacketIn messages with modified length field in OpenFlow header have been generated. According to the protocol specification [4], this field stands for the whole message length (including the header). Its value was set as follows: 1) length of OpenFlow header; 2) length of OpenFlow header plus 'a', where 'a' is smaller than the header of the encapsulated OpenFlow message; 3) a number larger than the length of the whole message. This test case examines how the controller parses the queue of incoming messages.

Invalid OpenFlow version. All controllers under test support OpenFlow protocol v1.0. In this test case we start the session with announcing OpenFlow v1.0 during the handshake between the switch and the controller, then set an invalid OpenFlow protocol version in PacketIn messages.

Incorrect OpenFlow message type. We examine how the controller processes the OpenFlow messages, in which the value of the 'type' field of the general OpenFlow header does not correspond to the actual type of the encapsulated OpenFlow message. As an example we send messages with OFPT_PACKET_IN in the 'type' field, but containing PortStatus messages.

4.3.2 Malformed OpenFlow message.

PacketIn message: incorrect protocol type in the encapsulated Ethernet frame. This test case shows how the controller's application processes an Ethernet frame with a malformed header. We put the code of the ARP protocol (0x0806) in the Ethernet header, though the frame contains the IP packet.

Port status message: malformed 'name' field in the port description. The 'name' field in the port description structure is an ASCII string, so a simple way to corrupt this field is not to put '\0' at the end of the string.

5. RESULTS

In this section we present the results of the experimental study of the controllers performed with cbench (performance/scalability tests) and hcprobe (reliability and security tests).

5.1 Performance/Scalability

5.1.1 Throughput

CPU cores scalability (see Figure 4). Python controllers (POX and Ryu) do not support multi-threading, so they show no scalability across CPU cores. Maestro's scalability is limited to 8 cores as the controller does not run with more than 8 threads. The performance of Floodlight increases steadily in line with the increase in the number of cores, slightly differing on 8-12 cores. Beacon shows the best scalability, achieving the throughput of nearly 7 billion flows per second.

The difference in throughput scalability between multi-threaded controllers depends on two major factors: the first one is the algorithm of distributing incoming messages between threads, and the second one is the mechanism or the libraries used for network interaction.

NOX, MuL and Beacon pin each switch connection to a certain thread, this strategy performs well when all switches send approximately equal number of requests per second. Maestro distributes incoming packets using round-robin algorithm, so this approach is expected to show better results with an unbalanced load. Floodlight relies on Netty library, which also associates each new connection with a certain thread.

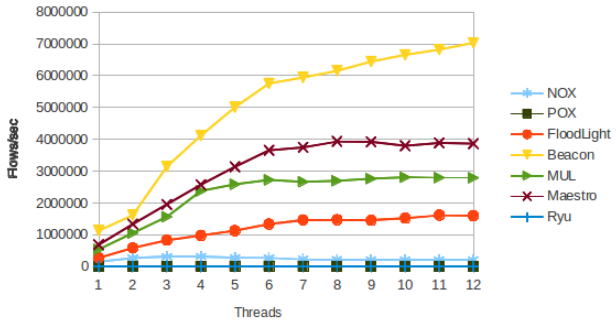

Figure 4: The average throughput achieved with different number of threads (with 32 switches, 10^5 hosts per switch)

Correlation with the number of connected switches (see Figure 5). The performance of POX and Ryu does not depend on the number of switches, as they do not support multi-threading. For multi-threaded controllers adding more switches leads to better utilization of available CPU cores, so their throughput increases until the number of connected switches becomes larger than the number of cores. Due to the round-robin packets distribution algorithm, Maestro shows better results on a small amount of switches, but the drawback of this approach is that its performance decreases when a large number of switches (256) is connected.

Correlation with the number of connected hosts (see Figure 6). The number of hosts in the network has immaterial

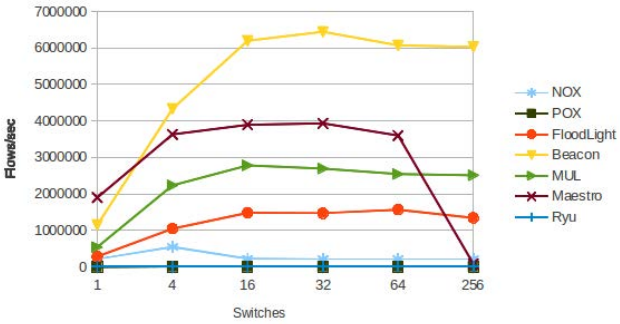

Figure 5: The average throughput achieved with different number of switches (with 8 threads, 10^5 hosts per switch)

influence on the performance of most of the controllers under test. Beacon's throughput decreases from 7 to 4 billion flows per second with 10^7 hosts, and the performance of Maestro goes down significantly when more than 10^6 hosts are connected. This is caused by specific details of Learning Switch application implementation, namely, the implementation of its lookup table.

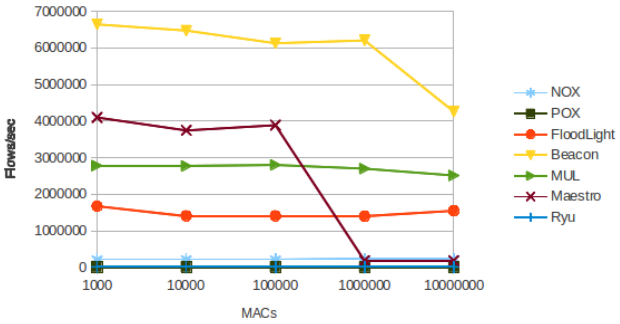

Figure 6: The average throughput achieved with different number of hosts (with 8 threads, 32 switches)

5.1.2 *Latency*

The average response time of the controllers demonstrates insignificant correlation with the number of connected hosts. For the average response time with one connected switch and 10^5 hosts see Table 1. The smallest latency has been demonstrated by MuL and Beacon controllers, while the largest latency is typical of python-based controller POX.

NOX	91,531
POX	323,443
Floodlight	75,484
Beacon	57,205
MuL	50,082
Maestro	129,321
Ryu	105,58

Table 1: The minimum response time (10^{-6} secs/ flow)

The results of performance/scalability testing show that Beacon controller achieves the best throughput (about 7 billion flows/sec) and has a potential for further throughput scalability if the number of available CPU cores is increased. The scalability of other multi-threaded controllers is limited by 4-8 CPU cores. Python-based controllers POX and Ryu are more suitable for fast prototyping than for enterprise deployment.

5.2 Reliability

The experiments have shown that most of the controllers successfully cope with the test load, although two controllers, MuL and Maestro, start to drop PacketIns after several minutes of work: MuL has dropped 660,271,177 messages and closed 214 connections, and Maestro has dropped 463,012,511 messages without closing the connections. For MuL, failures are caused by problems with Learning Switch module that cannot add new entries to the table, which leads to packet loss and the closing of connections with switches.

5.3 Security

5.3.1 *Malformed OpenFlow header*

Incorrect message length (1). Most of the controllers will not crash on receiving this type of malformed messages but close the connection with the switch that sends these messages. The two controllers that crash upon receiving messages with incorrect value of length field are Maestro and NOX. Maestro crashes only when these messages are sent without delay, however, if we set a delay between two messages to 0.01 second the controller replies with PacketOuts. Ryu is the only controller whose work is not affected by malformed messages, it ignores the messages without closing the connection.

Invalid OpenFlow version (2). POX, MuL and Ryu close the connection with the switch upon receiving a message with invalid protocol version. Other controllers process messages regardless of an invalid protocol version and reply with PacketOuts with version set to 0x01.

Incorrect OpenFlow message type (3). MuL and Ryu ignore these messages. MuL sends another FeaturesRequest to the switch and after receiving the reply closes the connection with the switch. Other controllers parse them as PacketIn messages and reply with PacketOuts.

5.3.2 *Malformed OpenFlow message*

PacketIn message (4). MuL closes the connection with the switch after sending an additional FeaturesRequest and receiving the reply. Other controllers try to parse encapsulated packets as ARP and reply with a PacketOut message. POX is the only controller which detects invalid values of ARP header fields.

Port status message (5). All controllers process messages with malformed ASCII strings correctly, reading the number of characters equal to the field length.

The summary of the security tests results is shown in the Table 2, where: red cell — the controller crashed, orange cell — the controller closed the connection, green cell — the controller successfully passed the test. If the cell is yellow, the controller processed the message without crashing or closing the connection, but the error was not detected, which is a possible security vulnerability.

Thus, the experiments demonstrate that sending mal-

	1	2	3	4	5
NOX					
POX					
Floodlight					
Beacon					
MuL					
Maestro					
Ryu					

Table 2: Security tests results

formed OpenFlow messages to a controller can cause the termination of a TCP session with the switch or even the controller's failure resulting in a failure of a network segment or even of the whole network. The expected behaviour for the OpenFlow controller on receiving a malformed message is ignoring it and removing from the incoming queue without affecting other messages. If the controller proceeds the malformed message without indicating an error, this could possibly cause incorrect network functioning. So it is important to verify not only the OpenFlow headers, but also the correctness of the encapsulated network packets.

6. CONCLUSION

We present an impartial experimental analysis of the currently available open source OpenFlow controllers: NOX, POX, Floodlight, Beacon, MuL, Maestro, and Ryu.

The key results of the work:

- The list of the controllers under the test has been extended as compared to previous studies.
- We present a detailed description of the experiment setup and methodology.
- Controller performance indexes have been upgraded using advanced hardware with a 12 CPU threads configuration (previous research used maximum 8 threads). Maximum throughput is demonstrated by Beacon, with 7 billion flows per second.
- Reliability analysis showed that most of the controllers are capable of coping with an average workload during long-term testing. Only MuL and Maestro controllers are not ready for the 24/7 work.
- Security analysis highlighted a number of possible security vulnerabilities of tested controllers which can make them targets for the malware attacks.

The results show that these aspects have to be taken into account during development of new SDN/OpenFlow controllers and should be improved for the current controllers. The current controllers have bad scaling over the cores and will not able to meet increasing demands in communication for future data centers. The security and reliability have highest priorities for enterprise networks.

We also present hcprobe — the framework, which perfectly fits the demands of fine-grained functional and security testing of controllers. It is a good base for a full test suite for testing SDN/OpenFlow controllers on compliance to the OpenFlow specification (like OFTest for OpenFlow switches [20]).

7. REFERENCES

[1] M. Casado, T. Koponen, D. Moon, S. Shenker. *Rethinking Packet Forwarding Hardware*. In Proc. of HotNets, 2008

[2] S. Jain, A. Kumar, S. Mandal. *B4: experience with a globally-deployed software defined wan*. In proceedings of ACM SIGCOMM conference, ACM, 2013

[3] C. Hong, S. Kandula, R. Mahajan. *Achieving high utilization with software-driven WAN*. In proceedings of ACM SIGCOMM conference, ACM, 2013

[4] Open Networking Foundation. *OpenFlow Switch Specication* https://www.opennetworking.org/sdn-resources/onf-specifications/openflow

[5] D. Ferrary. *Computer Systems Performance Evaluation*. N.J.:Prentice-Hall, Englewood Cliffts,1978

[6] D. Ferrary, M. Liu. *A general purpose software measurement tools*. Software Practice and Experience, v.5, 1985

[7] Amin Tootoonchian, Sergey Gorbunov, Martin Casado, Rob Sherwood. *On Controller Performance in Software-Defined Networks*. In Proc. Of HotIce, April, 2012

[8] Cbench. http://docs.projectfloodlight.org/display /floodlightcontroller/Cbench

[9] Hcprobe. https://github.com/ARCCN/hcprobe/

[10] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. *NOX: towards an operating system for networks*. SIGCOMM Computer Communication Review 38, 3 (2008), 105-110.

[11] POX. http://www.noxrepo.org/pox/about-pox/

[12] Beacon. https://openflow.stanford.edu/display/Beacon

[13] Floodlight. http://Floodlight.openflowhub.org/

[14] Mul. http://sourceforge.net/p/mul/wiki/Home/

[15] Zheng Cai, *Maestro: Achieving Scalability and Coordination in Centralized Network Control Plane*, Ph.D. Thesis, Rice University, 2011

[16] Ryu. http://osrg.github.com/ryu/

[17] FlowER. https://github.com/travelping/flower

[18] Trema. http://trema.github.com/trema/

[19] Nettle. http://haskell.cs.yale.edu/nettle/

[20] Big Switch Networks *OFTest – Validating OpenFlow Swtiches*. http://www.projectfloodlight.org/oftest/