INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
ISSN 1841-9836, 11(5):697-708, October 2016.

NVP: A Network Virtualization Proxy for Software Defined Networking

B. Pinheiro, E. Cerqueira, A. Abelem

Billy Pinheiro*, Eduardo Cerqueira, Antonio Abelem
Federal University of Para
Brazil, Para, Belem
*Corresponding author: billy@ufpa.br
cerqueira@ufpa.br, abelem@ufpa.br

Abstract: The combination of Network Function Virtualization (NFV) and Software Defined Networking (SDN) can improve the control and utilization of network resources. However, this issue still requires proper solutions to virtualize large-scale networks, which would allow the use of SDN and Virtualization in real environments. Thus, this paper proposes a virtualization architecture for SDN that relies on a proxy-based approach. The NVP (Network Virtualization Proxy) is a virtualization proxy that intercepts messages exchanged between controllers and switches SDN enabling network virtualization. An implementation of the proposal was developed as a proof of concept and load testing was performed showing that the solution can provide network virtualization in a scalable manner, using less than 2.5 MB of memory to manage 100 switches performing simultaneous requests, whereas FlowVisor requires more then 200 MB.
Keywords: SDN, virtualization, NVP, FlowVisor.

1 Introduction

The Internet was originally designed to provide network services to a closed community, and has today become an undeniable success worldwide, with users of various types of services everywhere on the planet.

Specific adaptations have historically been proposed and implemented at the emergence of new demands. This approach, despite having attended to momentary needs, has generated increasing complexity and cost of maintaining the Internet. Moreover, the higher the number of these adjustments, the greater is the complexity of the resulting architecture, making it even more difficult to overcome future challenges, in a situation commonly referenced as Internet "ossification", increasingly resistant to structural changes [1].

In this context, Software Defined Networking (SDN) is today one of the most relevant solutions for Future Internet environment, and the OpenFlow protocol implementation is the best-known method [2]. Along with SDN, the field of Network Function Virtualization (NFV) has grown so as to allow resources to be shared and dynamic topologies to be created [3].

The main concept for working with network virtualization is the division into slices, to optimize the use of their resources (nodes or links) and to separate it into different logical instances. For example, this approach has been used in the FIBRE project, to create a common space between Brazil and the EU for future experimental Internet research into network infrastructure and distributed applications with more then 40 OpenFlow nodes [4].

The partitioning of the network enables the actions taken in one of the network slices, to not interfere with others, even if they are sharing the same physical infrastructure. In traditional architectures, the network is sliced through VLAN's technique, but, with the diversity of network models, the structure of VLANs makes the experiments with others protocols rather difficult to manage [1].

Currently, FlowVisor [5] is the leading virtualization solution available for OpenFlow networks. It acts as a layer of virtualization located between OpenFlow switches and the network controller allowing resources to be shared, but maintaining the isolation of a virtual network from others. A major problem with this solution is the scalability since the recommended requirements for operating the FlowVisor are 4 processor cores and 4 GB Java heap. Depending on the size and use of the network, these requirements can increase [6].

This article proposes the Network Virtualization Proxy (NVP), a new model of network virtualization that improves the use of computational resources in the virtualization process, through an innovative architecture for greater scalability. To prove the effectiveness of the proposal, an implementation has been developed, and load tests were performed comparing NVP with FlowVisor.

This article is structured as follows: Section 2 presents SDN virtualization related works. In Section 3, the NVP is detailed. In Section 4, the tests performed and their results are discussed. Finally, Section 5 presents the conclusions and future works.

2 Related Work

FlowVisor is a well-known networks virtualization solution for SDN OpenFlow environment [5]. It is proposed as a special controller that acts as a transparent proxy, located between the OpenFlow switch and the network controller. By using FlowVisor, it is possible divide the switches and have their resources controlled by more than one controller at the same time, thus enabling the creation of virtual networks with logical isolation between them.

The FlowVisor adopts an hierarchical architecture that allows multiple FlowVisor networks to be interlinked making virtualization of an already virtualized resource possible. In order to function seamlessly, it acts as a controller for the switches and as an OpenFlow switch for the controllers. Thus, it must store the information of the switches and controllers to enable the routing of messages between them. However, its architecture generates an overhead, because the same entity responsible for managing the virtualized network's information is responsible for forwarding the generated messages, including the packet_in. These are packets coming through the network and sent to the controller for review, which can consume a lot of resources to be forwarded between FlowVisors to reach the controllers destination.

The Prefix-based Layer 2 Network Virtualization (L2PNV) [7] is an extension of FlowVisor, with the main objective of providing a level 2 virtualization engine without using VLAN, instead basing the virtual networks created in Media Access Control (MAC) on the address of origin and destination. However, it was necessary to modify FlowVisor, the switch firmware used to support a modified version of OpenFlow and the host since it was necessary to enable MAC masking for these elements. Despite being discussed in the article that a test environment was created using the proposed solution and the necessary changes in the above-mentioned elements fit the demands of the proposal, no results related to scalability of the solution or comparison with other existing solutions were presented. In fact, no evaluation was submitted.

Aiming to overcome FlowVisor's major limitation, the fact that virtual topologies that may be created with FlowVisor are restricted to a subset of the physical topology, the Advisor [8] was proposed. This solution leverages the VLAN tag to differentiate between virtual links and the virtual network, making it possible to create virtual links by joining different physical links. However, the solution that the authors validate necessitates the manual configuration of network devices and the sending of dpctl commands. Furthermore, the solution limits the use of the VLAN header, leaving the less transparent solution for the user. In addition, tests have shown an increase in terms of latency when compared it to FlowVisor. Again, load tests with a large network, such as the FIBRE network, showing the scalability of the solution were not performed.

To solve the problem of virtual links, Vertigo [9] was proposed. An extension of FlowVisor, it basically adds a layer of intelligence which enables the creation of virtual topologies over the physical topology. In this case, it creates virtual links interconnecting physical links. Moreover, it may provide a complete virtual structure, since it is possible to virtualize the network nodes together with the links. The proposal was implemented in the testbed OFELIA, where it was possible to prove that the solution can, in fact, provide virtual links. It is possible to verify the data taken from the article itself, whereby the solution shows an increase of 44.70 % in overhead when compared with FlowVisor, a fact that makes the scalability of the proposal questionable.

The FlowVisorQoS proposed [10] solves the problem of ensuring bandwidth usage per slice in FlowVisor. Thus, it is possible to define minimum requirements for Quality of Service (QoS) that will be respected in the switches, ensuring that each flow does not exceed the specified limits. For the implementation of the solution, changes in the firmware of the switches were performed. However, despite achieving the solution's objectives, tests to show the solution behavior in a large-scale environment were not performed.

OpenVirteX [11] builds on the design of FlowVisor, and functions as an OpenFlow controller proxy , but, differently of FlowVisor, OpenVirtex provides Virtual Links and slices isolation. The hardware requirements for both FlowVisor and OpenVirtex are the same and they requires 4 core processors and 2GB RAM [12].

The Flow-N [13] is, to our knowledge, the first work on network virtualization that addresses scalability problems. It was proposed as an extension of the NOX controller. It enables virtualization-based containers and uses relational databases to map between the physical and virtual network topologies. The scalability tests presented in the study show an almost stable latency with 100 virtual networks created, while FlowVisor starts with a low value but increases to almost equalize the value of Flow-N. Unfortunately, only one chart is presented, and the proposal can not be evaluated further while details about the operation of the solution are not provided.

As discussed in the papers presented in this section, the virtualization solution for SDN must provide isolation, virtual links, and be prepared to support a large quantity of equipment, because all other services will be running virtualized over the physical equipments

Table 1 summarizes the main features of the proposed virtualization in this section. It was concluded that none of the protocols can fully meet all the requirements cited above in order to provide a reliable and scalable communication that might require an SDN for the virtualization.

Table 1: Virtualization Solutions Comparison

Proposal	Scalability	Virtual Links	Transparent	Isolation
FlowVisor	No	No	Yes	Partial
L2PNV	No	No	Yes	Partial
ADVisor	No	Yes	No	Partial
VeRTIGO	No	Yes	Yes	Partial
FlowVisorQoS	No	No	Partial	Yes
OpenVirtex	No	Yes	Yes	Yes
Flow-N	Partial	Yes	Yes	Partial

Finally, is important to emphasize that, despite the clear importance of scalability for virtualization solutions in the SDN environment, only one proposal attempts to address this issue. However, it does not provide the information necessary for a more satisfactory evaluation, such as the comparative numbers of supported switches, number of controllers, and number of virtual networks.

3 Network Virtualization Proxy (NVP)

As presented in Section 2, the fundamental requirements for virtualization in SDN are virtual links to enable a greater variety of topologies and scalability, aiming to cover a larger number of devices in the physical substrate. Such features expected for this technology could actually be used as a basis for production networks.

A key feature of the FlowVisor architecture is to enable its use hierarchically. This feature brings high flexibility regarding different possible topological configurations. However, this feature imposes an important limitation for network scalability, since it introduces more elements between the controller and the switch, as the hierarchical level increases.

The NVP enables virtualization scalability in an efficient way. Thus, the main contributions of our proposal are the following:

- Allow resources of OpenFlow switches to be shared between different controllers;
- Increase the network scalability, allowing the virtualization layer to support a high number of switches and controllers without generating a big network overhead;
- Provide a global network abstraction, in which all available resources are not separated hierarchically, i.e., increasing the number of network devices does not increase the number of elements between switches and controllers;
- Enable a formal modeling of virtualized resources.

In order to better structure the display and use of NVP, an CIM extension (Common Information Model) was developed to allow modeling of SDN using UML (Unified Modeling Language) [14].

3.1 NVP Architecture

The NVP is composed of two elements: the Proxy Flow Switch (FPS) and the Flow Proxy Manager (FPM). Its design was inspired in OpenFlow itself, in which the complexity of control is the removal of switches and passed to an external control entity. In this case, the FPS acts as a forwarder and FPM concentrates all of the system virtualization's information and intelligence. Its architecture is shown in Figure 1.

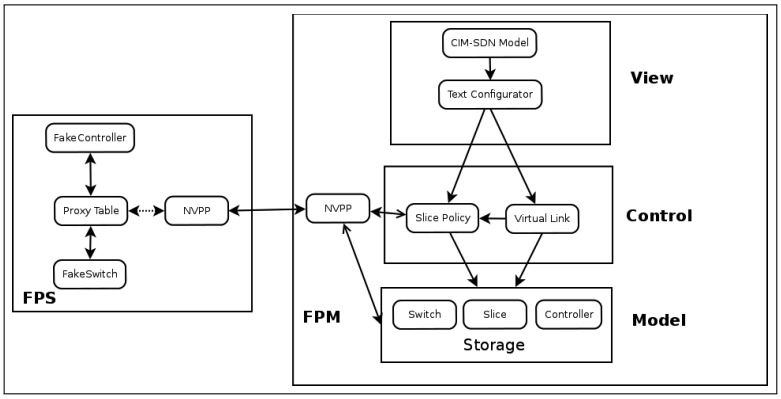

Figure 1: NVP Architecture

Flow Proxy Switch (FPS)

It is the entity responsible for forwarding the information between switches and controllers. It will exchange messages with OpenFlow switches and controllers and pass them on to the FPM.

After the network start up, FPS use is dedicated to consulting its routing table and deciding to which controller/switch the messages should be forwarded. It is formed of the components: FakeController, FakeSwitch, ProxyTable, and Network Virtualization Proxy Protocol - (NVPP).

- **FakeController:** This is an entity that acts as an OpenFlow controller, passing the idea on to the OpenFlow switch that is connected to the network controller. It is responsible for ensuring the transparency from the switches side and use NVPP to send the information obtained from the switches to the FPM;
- **FakeSwitch:** Like FakeController, it is responsible for ensuring transparency from the side of the controllers and providing information to those obtained about the resources that each controller will access (with the FPM through NVPP). It is an entity that acts as an OpenFlow controller, communicating the idea to the OpenFlow switch that is connected to the network controller;
- **Network Virtualization Proxy Protocol (NVPP):** It is a binary protocol, similar to OpenFlow, developed to enable communication between FPSs and FPM. Because of the need to maintain FPS as simple as possible, we chose to develop a binary protocol that can meet the need for communication between entities, generating as little computational overhead possible. The data types used in the NVP are similar to those used in OpenFlow protocol, and its use follows the same logic defined in the OpenFlow protocol specifications. [15];
- **Proxy Table:** It is a table that stores which resources of a given switch can be used by a particular controller. It is composed of the fields ctr _addr and ctr _port, which uniquely identify the network controller, the field datapath_id which identifies to which switch this flow is related, and the fields in_port and dl_vlan that identify the port and the vlan associated with the virtual network.

Table 2: Proxy Table

ctr_addr	ctr_port	datapath_id	in_port	dl_vlan
192.168.1.2	6633	2	1	2
192.168.1.1	6633	2	1	1

In the example of Table 2, an action of the flow type_mod arrival of a controller with ip 192.168.1.2 and running on port 6633 will be forwarded exclusively to the switch with the datapath_id 2 on port 1 and vlan 2. If one of the fields differ, the packet is discarded.

Flow Proxy Manager (FPM)

It is the element that performs network virtualization's effective control. It will contain a database with information about the switches, controllers, and the network slices. All the slices changes must pass through it, and it should embed the rules in FPS. A FPM will be used in each domain- there may be several FPS associated with it- allowing the network to be scalable, since the actual load generated in the network through the packet_in, and these will be handled only by the PFSs. Its architecture follows the pattern Model-view-controlller (MVC) and is described next.

- **Storage:** It is the component responsible for storing the switches' information, controllers, and the network slices. The data stored here will be used by the Slice Policy to feed the FPS;

- **Slice Policy:** Responsible for the slices creation, allowing the creation of policies and associating resources to controllers required by the administrator. It should use the information from switches and policies present in the Storage and create slices in FlowTable;
- **Virtual Link:** This element is responsible for mapping the physical links and enabling the creation of virtual links by setting up forwarding rules in the switches. Thus it is possible to interconnect continuous physical links and provide the abstraction of a virtual link directly connecting two ports of the switches;
- **Text Configurator:** It is the NVP configuration interface. It is similar to the configuration file used by FlowVisor to facilitate the adoption of NVP and uses JSON (JavaScript Object Notation) syntax;
- **CIM-SDN Model:** This is an XMI (XML Metadata Interchange) file that models the entire network to be virtualized and the virtual networks themselves. It uses the CIM-SDN [14], which is a CIM extension that allows the representation of SDN elements in a UML diagram, ensuring the consistency the network. It also maintains an updated documentation for easy understanding.

The NVP has its architecture divided into two main elements: the FPS and the FPM, aiming to provide, at the same time, scalability and a global network view. Scalability is the result of multiple FPSs that can be used in the network in order to provide the best use of the resources of the switches and controllers. The global view is available at FPM, which contains all the network information and can provide easy integration to other networks through interconnection with other FPMs from other networks.

3.2 Applying the NVP

Typically, NVP is located between the switches and the network controller. It is possible to see in Figure 2 as the components of NVP relate with other network components. One FPS can suit various switches and controllers, and, to enable network scalability, the number of FPSs can be increased to suit more switches and controllers. The FPM is unique to the network and is responsible for providing a global network view.

In Figure 3, the FPS and the MPF activity diagrams are presented. In the case of FPS, actions are started by the controllers and switches. The information of switches and controllers is collected, and subsequently, ProxyTable is used to determine the forwarding of packets. In FPM, the administrator starts activities, providing the CIM-SDN model as input to generate the creation of slices. It is worth noting that a packet is never forwarded to the MPF; only the FPS handles packets and does it using only ProxyTable as a decision resource.

The use of NVP is similar to FlowVisor plus the CIM-SDN. However, it is possible to directly use the configuration file, a fact that enables an easy integration with other tools that already have FlowVisor as the NFV.

4 Proposal Evaluation

An implementation to prove the concept of NVP was developed and its performance was compared with FlowVisor. The metrics used in the evaluation were the number of requests and cpu loads and memory usage.

In order to ensure the most efficient proposal implementation, the development of the NVP prototype was performed in C++ using the BOOST::ASIO [1]. In this version, FakeController

[1] http://www.boost.org/library

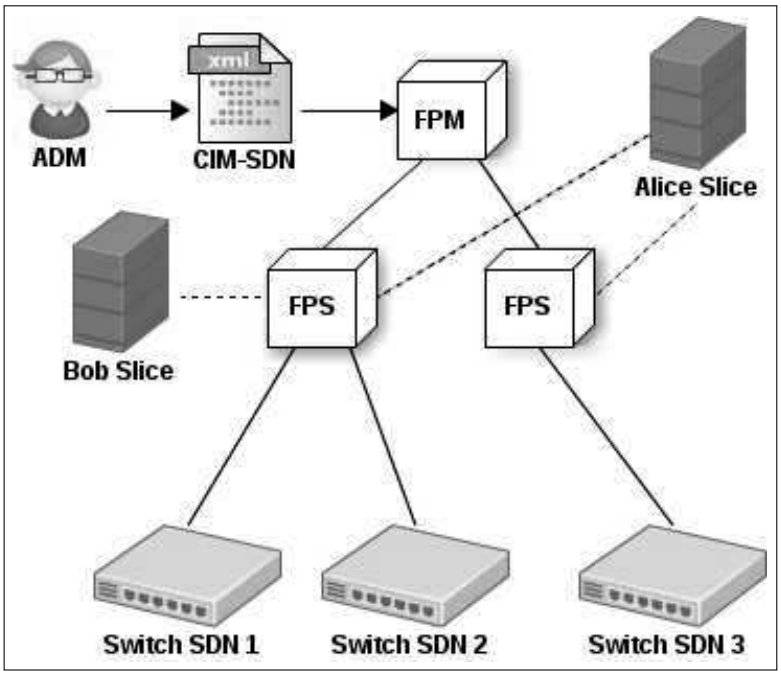

Figure 2: Applying the NVP

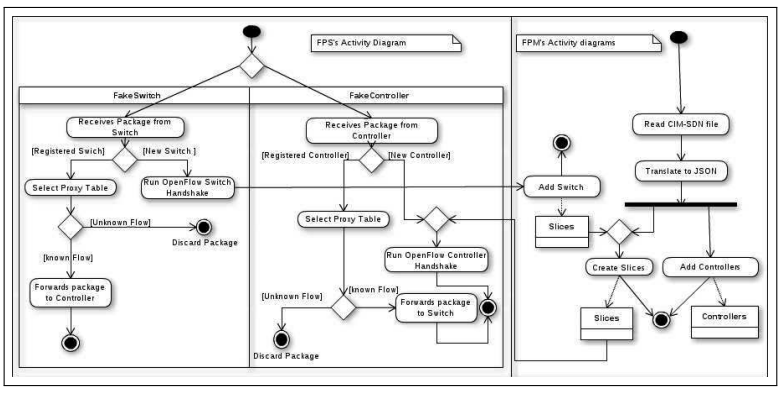

Figure 3: NVP Activity Diagrams

and FakeSwitch components were implemented in their entirety, as well as a short version of ProxyTable.

The only external library used was a file defining the types of data available in OpenFlow itself, to enable communication between the switches and controllers. The current implementation of FPS allows it to act as a proxy virtualization, although it may not include all elements of the proposal, since the FPM has not been fully developed.

4.1 Test Methodology

The methodology for evaluation and validation of NVP is based on the application execution in a real environment, but, with tools that emulate OpenFlow switches and allow the creation of a large number of elements for the experiment, the use of real OpenFlow switches is not necessary [16].

The tool used to emulate OpenFlow switches was CBENCH, which is part of the framework oflops [2]. Using the tool, it is possible to define the number of switches to be emulated and indicate to which network controller it must connect.

The controller used in the experiments was the POX, having switches emulated by CBENCH as clients. The experiments were performed on the same machine so that network traffic would not influence the response time of the components. The machine is a CPU Intel core 2 quad 2.5GHz with 4GB of RAM and uses Debian Wheezy.

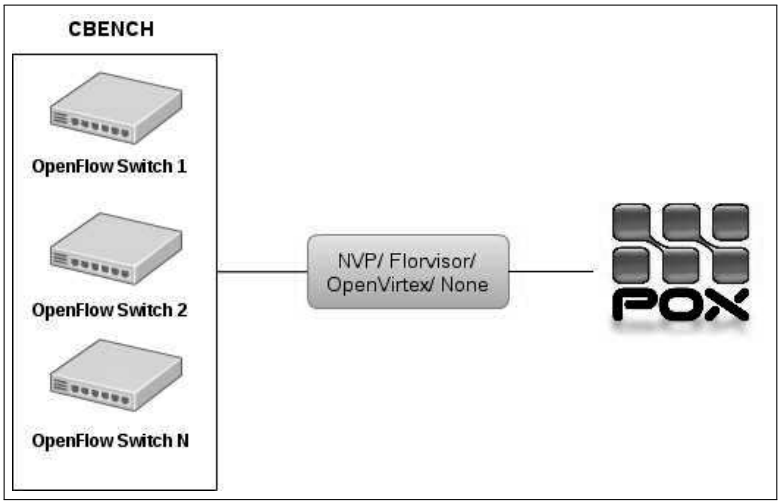

Figure 4: Experiments topology

The experiments used the POX controller [3] to handle the requests from the switches created with the CBENCH following this topology in Figure 4. Three experiments were performed, the first with the NVP between the switches and the controller, the second with FlowVisor taking the place of NVP, and the last experiment without any virtualization tool. Ten replicates were performed for each experiment using the number 1, 5, 10, 20, 40 and 100 switches to generate requests simultaneously.

4.2 Results Analysis

The goal of these experiments was to validate the NVP, demonstrating load tests with it to verify its behavior and compare it with the FlowVisor. The POX is presented, not for the

[2]http://archive.openflow.org/wk/index.php/Oflops
[3]http://www.noxrepo.org/pox/about-pox/

purpose of direct comparison, because it perform different tasks on the network, but to demonstrate the extent to which NVP, FlowVisor and OpenVirtex introduce overhead when acting as virtualization proxies.

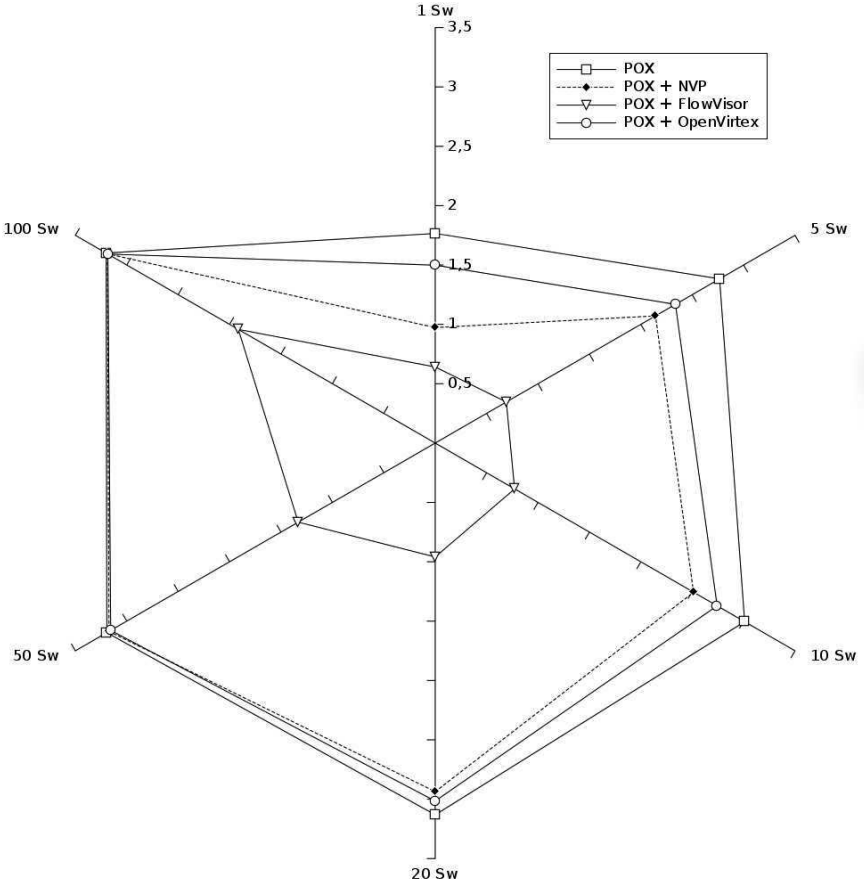

Figure 5: Reply per milisecond X Number of Switches

Figure 5 shows the number of responses that the controller can produce per millisecond. The direct comparison with FlowVisor shows a huge difference between the number of requests answered for each virtualization proxy. For one switch, the NVP is 53.68% better than FlowVisor, and, when the number of switches achieves 100, the NVP results are 89.83% better, showing a great increase in performance. When the NVP is compared with OpenVirtex their results are getting almost the same according to the increase of the number of switches, but for a small number of equipment the OpenVirtex has a better result.

Also, it is possible to see, when compared with POX, that the NVP shows fewer responses, as expected, since the message must be sent to the POX anyway. However, as the number of switches increases, it is possible to notice a reduction in the difference in the environment both with and without NVP. For one switch, the performance loss is 44.90 %, but it will decrease to 22.53 % with five switches until arriving at 1.10 % for 100 switches, showing a better performance when the number of switches is elevated.

Figure 6 shows the cpu load percentage of NVP, FlowVisor and OpenVirtex during the experiments. It is possible to observe that, even with requests for 100 switches at the same time, the CPU usage by NVP was less than 12% while FlowVisor gets, for the same number of switches, 107% of CPU, i.e., more than one CPU core is needed to handle the requests. The OpenVirtex shows a better result than FlowVisor, but uses more than 3 times the amount of memory when compared with NVP.

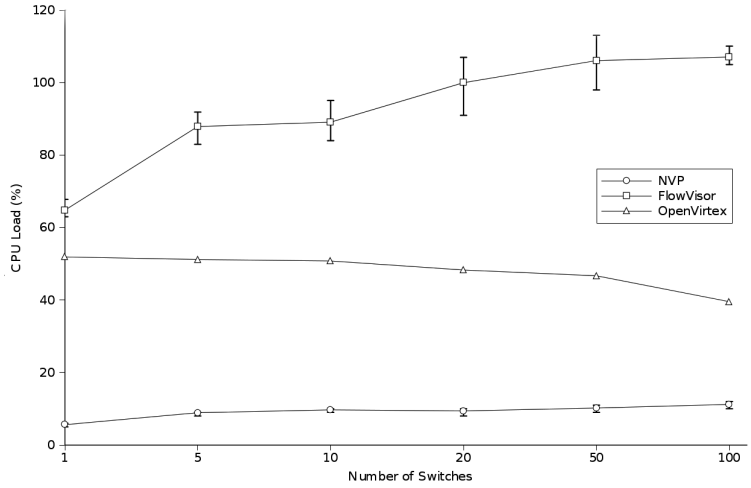

Figure 6: Cpu load time(%) X Number of Switches

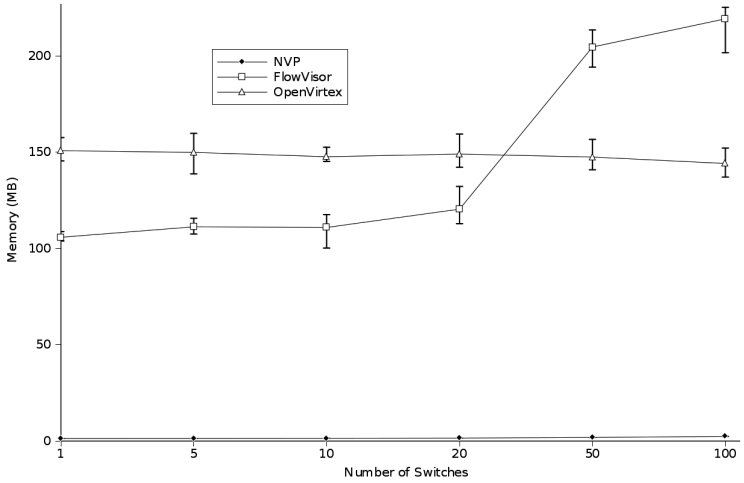

Figure 7: Memory Usage (MB) X Number of Switches

Figure 7 presents the memory consumption of NVP and FlowVisor during the experiments. The values presented are RSS (Resident Set Size), which is the physical memory the process currently uses. Again, even with 100 switches, total memory usage did not exceed 2.5 MB, showing that the solution exhibits a good memory usage. NVP would therefore be more attractive to clients, as opposed to FlowVisor, which uses more then 200 MB to handle the 100 switch requests. The memory used by OpenVirtex stays almost constant in 150 MB, overcoming in 60 times the memory used by NVP.

5 Conclusions and Future Work

This article proposes a framework for SDN virtualization, named Network Virtualization Proxy (NVP). The purpose of NVP is to provide a mechanism for scalable virtualization with isolation between virtual networks by creating and enabling the creation of virtual links. The innovative aspect of this work is that the proposed solution combines scalability and global network view.

The results presented in the experiments to allow for a validation of the proposal for the use of machine resources that will host the virtualization solution. Thus, NVP has demonstrated that it can act as a virtualization proxy without using many host resources, specifically, consuming, in our experiments, fewer than 2.5 MB of memory to manage connections originating from 100 different outfits. Also, all the comparisons with FlowVisor show a better results for the NVP proposal.

As future work, we intend to verify the individual limits of FPS and FPM, as well as examine a more detailed comparison with other virtualization proposals.

Bibliography

[1] Alcober, J. et al. (2013);Internet future architectures for network and media independent services and protocols, *15th International Conference Transparent Optical Networks, (ICTON)*, 1-4.

[2] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. White paper, Open Networking Foundation, Palo Alto, CA, USA (April 2012).

[3] King, D., Ford, C.(2013); A critical survey of Network Functions Virtualization (NFV), https://www.ietf.org/proceedings/86/slides/slides-86-sdnrg-3.pdf, 1-8.

[4] Sallent, S. et al. (2012); Fibre project: Brazil and europe unite forces and testbeds for the internet of the future, In: *Korakis, T., Zink, M., Ott, M. (eds.) TRIDENTCOM. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer, 44: 372-372.

[5] Sherwood, R. et al. (2009 ; *Flowvisor: A network virtualization layer. OpenFlow Switch Consortium*, Tech. Rep.

[6] Bastin, N. (2011); Flowvisor System Requirements, https://openflow.stanford.edu/display/DOCS/System+Requirements. Accessed: 2014-06-09.

[7] Matias, J., et al. (2012); Implementing layer 2 network virtualization using openflow: Challenges and solutions. *Software Defined Networking (EWSDN), 2012 European Workshop On*, 30-35.

[8] Salvadori, E., Corin, R.D., Broglio, A., Gerola, M. (2011); Generalizing virtual network topologies in openflow-based networks, *Global Telecommunications Conference (GLOBECOM 2011)*, IEEE, 1-6.

[9] Corin, R.D. et al.(2012), Vertigo: Network virtualization and beyond. In: Software Defined Networking (EWSDN), *2012 European Workshop On*, 24-29.

[10] Gomes, V.S. et al. (2013); Flowvisorqos: Aperfeicoando o flowvisor para aprovisionamento de recursos em redes virtuais definidas por software, *IV Workshop de Pesquisa Experimental da Internet do Futuro-Simposio Brasileiro de Redes de Computadores e Sistemas Distribuidos (SBRC 2013)*, Brasilia, 35-41.

[11] Al-Shabibi, A. et al. (2014); Openvirtex: Make your virtual sdns programmable. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, *HotSDN'14*, ACM, New York, NY, USA , 25-30.

[12] OnLab; OpenVirtex System Requirements. http://ovx.onlab.us/getting-started/installation/. Accessed: 2015-07-14.

[13] Drutskoy, D., Keller, E. (2013), Rexford, J.; Scalable network virtualization in software-defined networks, *Internet Computing*, IEEE, 17(2): 20-27

[14] Pinheiro, B., Chaves, R., Cerqueira, E., Abelem, A. (2013); Cim-sdn: A common information model extension for software-defined networking, *Globecom Workshops (GC Wkshps), 2013 IEEE*, 836-841.

[15] ONF; OpenFlow Switch Specification. https://www.opennetworking.org/images/stories/ downloads/sdn- resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf. Accessed: 2015-07-14 (2009).

[16] Jarschel, M. et al.(2011); Modeling and performance evaluation of an openflow architecture, *Proceedings of the 23rd International Teletraffic Congress. ITC'11*, 1-7.