OMCH-SDN: An Overlay multi-controller Hypercube-based topology for Software-defined Networks

Othmane Blial, Mouad Ben Mamoun, Redouane Benaini
ANISSE, Faculty of Sciences, Mohammed V University in Rabat,
Avenue des Nations Unies, Agdal, 10000 Rabat, Morocco
blial.othmane@gmail.com
ben_mamoun@fsr.ac.ma
benaini@fsr.ac.ma

Abstract— Multi-controller architectures have become very necessary for software-defined networks to provide more efficiency, scalability, flexibility and security. However, the controllers must be interconnected in a way that aims to improve the overall performance. In this paper, we propose the OMCH-SDN, an Overlay Multi-Controller Hypercube-based topology to interconnect controllers by using a Hypercube pattern to take advantage of its various mathematical characteristics. The proposal for this research is evaluated through simulations of a real overlay network. The results show that the OMCH-SDN model shows better results concerning the synchronization delay, the throughput as well as the end-to-end latency comparing to a flat-based SDN multi-controller topology.

Keyword - SDN, Multi-controller architecture, Hypercube-based topology

I. INTRODUCTION

The networking field knows recently a big revolution with the introduction of new concepts like, network automation, network function virtualization and software-defined networking, which aim to shape the network and improve its productivity.

Software-defined networking (SDN) [1], which is a new concept that separates the control plane from the data plane in network nodes, like switches and routers, mainly to limit the complexity and to enable the programmability in the network throughout all types of APIs, to make it more agile and flexible, especially the integration of new functionalities and services.

An SDN network in general has three layers: the infrastructure layer, the control layer, and the management layer, each layer has its own characteristics as well as its own components.

Southbound APIs (SB-APIs) like OpenFlow [2] for example, which the most recognized and used Southbound API in the SDN community. They allow communication between the infrastructure layer and the control layer, by giving the end user more power over the network, to adjust and configure smoothly the different components inside the forwarding plane. OpenFlow is supported and maintained by the Open Network Foundation (ONF) [3], and backed by many IT big enterprises, like Facebook and Google. There are many other SB-APIs, for instance OpFlex [4] from the worldwide networking leader, Cisco.

While, northbound APIs (NB-APIs) permit interaction between the controller and the management layer. They give a big opportunity to enable programmability inside the network, and automate many repetitive networking tasks. NB-APIs can be integrated with all sort of orchestration platforms like emerging and fast growing solution OpenStack [5] or automation stacks like Puppet [6], Chef [7], Ansible [8] and others.

The control layer, or the control plane is the most important piece and the smartest of an SDN network. Once it is configured by a network engineer or administrator, it takes care of all the rest of its own, from pushing policies to the different devices in the network to observing everything happening in the network and reporting it in the form of logs that can be very useful in the future to analyze the behavior of the network and solve problems that might occur.

SDN used at first apparition a control plane that has a single centralized controller, for example NOX [9], and Floodlight [10] which showed after a while many scalability, overall performance, and security limits. Therefore, SDN networks with multiple controllers have been proposed [11]. Hence, many SDN controllers have been proposed with support for multiple controllers, mainly the Open Source SDN Platform, OpenDayLight [12], and the Open Network Operating System, ONOS [13].

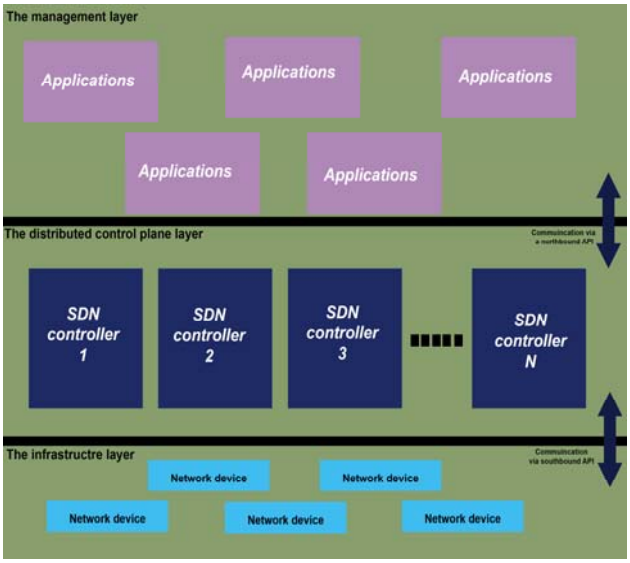

Fig. 1. The SDN distributed control plane architecture

In an SDN classic flat-based multi-controller topology for example, each controller is connected to two neighbors, except the edge ones, which are linked with just one controller. This topology interconnects the controllers among one level; it is suitable for designs with a few number of nodes. However, when the network scales to a certain number of controllers the time response between far nodes becomes too long.

As we explained this type of topologies present problems that are directly affecting performance and scalability.

For this reason, we propose in this paper, the use of an n-dimensional hypercube because of its advantageous mathematical properties that can be used to enhance the performance and increase the scalability. In fact, it has been deployed in many works, and demonstrates in the past its usefulness and its efficiency. For example, in the networking field, we find HyperCuP [14], which suggests a Hypercube-based peer-to-peer network introducing an efficient method for broadcasting and searching, or BlueCube [15], which offers a Bluetooth network based on a Hypercube, to support parallel computing and ensure fault tolerance. The Hypercube has been proposed in the context of SDN to connect between nodes in the underlying network [16], which is not the case in this paper.

Henceforth, we propose in this paper an Overlay Multi-controller Hypercube-based Software-define Network (OMCH-SDN), which consists of following a Hypercube pattern to connect between the controllers of the control plane using virtual tunnels. We consider this solution as an overlay solution because the controllers can be connected via the OMCH-SDN model independently of the underlying network, which means that controllers can be dispersed around the globe and still able to form the OMCH-SDN.

The remainder of this paper is organized as follows. In section 2, Material and Method, we first recall the definition of a Hypercube, its main properties and their benefits, and then, we describe our OMCH-SDN model. After that, we present our simulation description, by presenting the simulation Testbed, then the performance indices, the synchronization delay, the throughput and the end-to-end latency, which we are going to measure. In section 3, Results and Discussion, we present, discuss and analyze the simulation results. Finally, in Section 4, we give a Conclusion.

II. MATERIAL AND METHOD

In this section, we first present the hypercube and its mathematical characteristics, then explain how we will build the OMCH-SDN model.

A. The n-dimensional Hypercube and its proprieties

A hypercube is one of the most efficient ways used to interconnect between objects to improve their interaction performance in many fields because of its different mathematical characteristics. In graph theory, the n-dimensional Hypercube graph (H_n), with a total number of 2^n nodes, and $n2^{n-1}$ edges has the following properties:

Property 1: The longest path in H_n is n.

Hence it is better than a flat topology with 2^n nodes, where the longest path is 2^n - 1.

Property 2: H_n has an Euler tour for $n \geq 2$.

Which means that H_n is fully connected and each vertex has an even degree. This implies that H_n has a high connectivity and high fault tolerance.

Property 3: H_n has a Hamiltonian cycle for $n \geq 2$.

Which can be viewed as cyclic Gray Code of all 2^n binary strings of length n, thus, each node in H_n can be represented in an n-bit format. A Gray Code is used to represent each number from 0 to $2^n - 1$ in a binary form of length n, with the condition that adjacent numbers need to have just one bit that differs in their Gray Code representations. In other words, in a H_n, two nodes are directly connected if their binary representations differ with just one bit. This property will help us to assign to each controller node a Gray Code sequence, which will permit us to build our OMCH-SDN topology.

B. The OMCH-SDN model

The OMCH-SDN is an SDN network that has a control plane that contains multiple controllers connected following the hypercube pattern using virtual tunnels. Additionally, the OMCH-SDN is an overlay network, which means that the controllers can be at different locations in the world since the virtual tunnels can be created using routable public IP addresses. In other words, two controllers can be adjacent even if they need to go through multiple routers and switches in the underlying network.

The OMCH-SDN is constructed, using Gray Code sequencing system and basic interaction between the controllers and using the GRE (Generic Routing Encapsulation) [17], which is tunneling protocol that can be used to connect two devices remotely.

The OMCH-SDN constructing process has three steps:

Step 1 is the Setting up process where we have c controllers, each one has a routable IP address, and we give to each controller an environment variable with its binary representation (Gray Code sequence), and another environment variable that represents its ID.

Step 2 is The Solicitation process where each controller will send the other controllers a message with its own Gray Code sequence and its ID, looking for those that have their Gray Code string differs with just one bit. Then, create for them a matching number of locally GRE interfaces to attach the GRE tunnels later.

Step 3 is The GRE Tunnels Forming process where each controller will build the GRE tunnels using the GRE interfaces and the routable IP addresses with its neighbors.

After that, the OMCH-SDN constructing process is complete. But before that, even though controllers will have routable IP address, which means that they can reach each other's, when we will create the GRE tunnels, we will assign new private IP addresses to theses GRE interfaces, so we need a mechanism to enable routing. For this reason, we can use for example Quagga [18], a routing framework that can be employed in Linux systems to provide basic and advanced routing between various Linux servers.

To understand how the OMCH-SDN constructing process works, we will show you a use case for a hypercube of dimension k=3.

In a three-dimensional hypercube, we have eight nodes (controllers) and twelve links. Each node can be represented in a Gray Code sequence of three bits. To start the OMCH-SDN constructing process, we should follow the steps already explained.

First, the setting up process where each controller will have a routable IP address, and two environment variables (binary representation + ID).

TABLE I. The Binary Presentation Of OMCH-SDN Of Dimension 3

The controller	**ID NUMBER**	**Gray Code**
C1	01	000
C2	02	001
C3	03	010
C4	04	011
C5	05	100
C6	06	101
C7	07	110
C8	08	111

Second, each controller will send its binary presentation to other controllers aiming to find those that differ with just one bit. In this case, each controller will find three adjacents. Therefore, it will create three GRE interfaces that will be used to form the GRE virtual tunnels inter-controllers. Finally, each controller will build three GRE virtual tunnels to establish the OMCH-SDN model.

The following table summarizes the two final steps, while showing the GRE interfaces IP addresses as well as the controllers that will be connected to each controller within this hypercube-based topology:

TABLE II. The Hypercube Constructing Process For The OMCH-SDN Of A Dimension 3

The controller	GRE INTERFACES	It will be connected to
C1	Gre0:192.168.12.1 Gre1:192.168.13.1 Gre2:192.168.15.1	C2,C3,C5
C2	Gre0:192.168.12.2 Gre1:192.168.24.2 Gre2:192.168.26.2	C1,C6,C4
C3	Gre0:192.168.13.3 Gre1:192.168.34.3 Gre2:192.168.37.3	C1,C4,C7
C4	Gre0:192.168.34.4 Gre1:192.168.24.4 Gre2:192.168.48.4	C2,C3,C8
C5	Gre0:192.168.15.5 Gre1:192.168.56.5 Gre2:192.168.57.5	C1,C6,C7
C6	Gre0:192.168.68.6 Gre1:192.168.26.6 Gre2:192.168.56.6	C2,C5,C8
C7	Gre0:192.168.37.7 Gre1:192.168.78.7 Gre2:192.168.57.7	C3,C5,C8
C8	Gre0:192.168.48.8 Gre1:192.168.68.8 Gre2:192.168.78.8	C4,C6,C7

The final illustration of the OMCH-SDN of dimension three will look like the showing in figure 1:

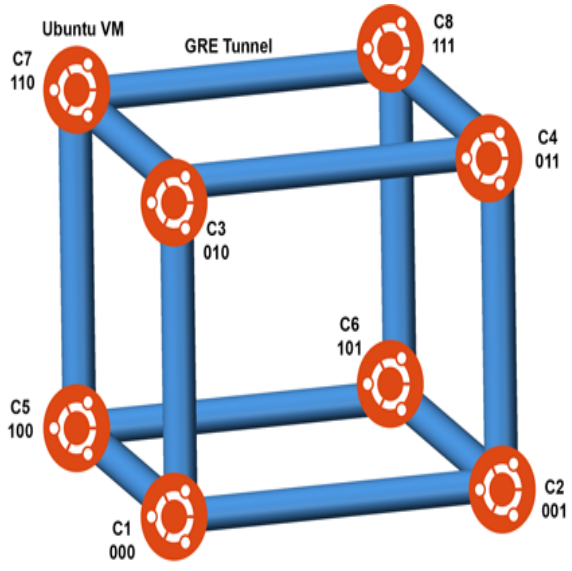

Fig. 2. The OMCH-SDN of a dimension 3.

C. Simulation description of OMCH-SDN model

In this subsection, we are going to describe the Testbed, which is built on the top of a real overlay network, present the performances indices that we are going to measure, which are the synchronization delay and the throughput inter the edge controllers.

1) Testbed description:

Our Testbed is using a Hosting Service called ATLENTIC.NET [19] to which we have subscribed to perform this simulation. It provides us with a cloud based environment that is billed by the hour.

To build our real overlay network, we are going to provision Ubuntu Linux servers in the USA-EAST-2 region in New York City. Each server has 2 CPUs and 4G of RAM. Additionally, each server is reachable on layer three connectivity to the other servers via the Internet.

In order to evaluate our proposition, we build our OMCH-SDN for dimension k = 3, which has eight controllers, where each one is up and running on one of our Linux servers that we have provisioned earlier. We use the open source project, OpenDayLight [12] as an SDN controller, the Beryllium release, because it supports multiple controllers, and has an engaging community that discusses, supports and contributes to this project. OpenDayLight supports an SDN-enabled network with multiple controllers, by building a logically centralized architecture as a cluster which is highly consistent thanks to the continuous synchronization inter-controllers mechanism.

Finally, we attach to each controller to a simulated underlying network generated by Mininet [20] with a particular number of nodes (OVS switches [21] + hosts), we take for example 9 underlying nodes (3 OVS switches, where each switch is connected to 2 hosts) by controller, in total 72 nodes.

The following figure summarizes the testbed for a hypercube-based topology:

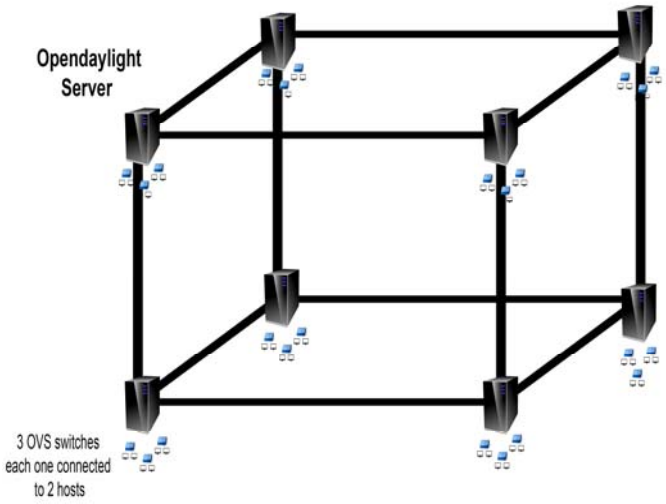

Fig. 3. The Testbed for a hypercube based topology of dimension 3 with 8 controllers

We will also similarly build a flat-based topology with same number of controllers and underlying nodes, to compare with, like you see in the following figure:

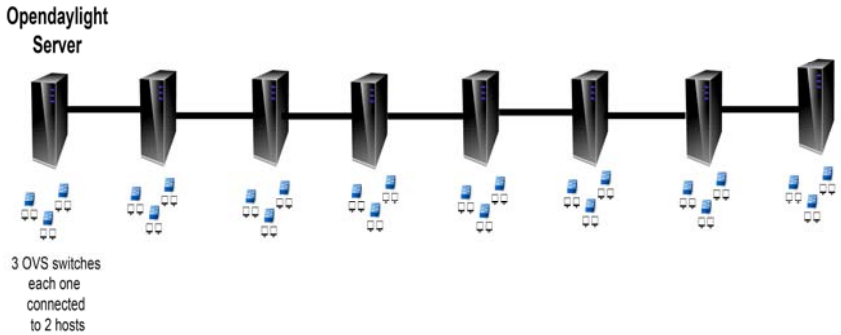

Fig. 4. The Testbed for a flat based topology with 8 controllers

Next, in the following subsection, we are going to present the performance indices that we will measure.

1) Performance indices:

When evaluating a multi-controller architecture, in particular, the logically centralized which needs a strong synchronization, an important performance index is the synchronization delay, which measure the time consumed to declare that a neighboring controller is aware of an event that was generated on some other controller. For this reason, we are going the measure the synchronization delay between the farthest controllers in the case of 8 controllers' scenario, in a hypercube-based topology and in a flat topology.

Then, we will measure the throughput also between the farthest controllers in our topology. Taking into consideration that the maximum throughput offered by our Testbed is 600 Mbits/s, we would like to know how the OMCH-SDN model will react, and what specific value will reach.

After that we finish by measuring the end-to-end latency.

We use Wireshark [22] to compute the synchronization delay, which is a software that permits to see what is happing in the network at a microscopic level.

While, we use Iperf [23] to measure the throughput, which is a tool used for active measurements of the maximum achievable bandwidth in a network and others.

Finally, we measure the end-to-end latency using the Ping utility.

III. RESULTS AND DISCUSSION

In this section, we are going to present and discuss the simulation results, by showing the simulation results of the synchronization delay and the throughput.

A. *The synchronization delay*

When we lunch the generated Mininet underlying networks inside each OpenDayLight controller, which is already a member of the cluster, we go through two phases, the switching events building, which takes a just few seconds, then the watching phase, which is the lasting phase, where the cluster synchronization happens each time something is changed, added or removed in the network in addition to maintaining a continuous consistency checking to ensure high consistency among all controllers.

In the following figure, we measure the synchronization delay between the edge controllers within the watching phase, while we add, remove or update hosts and flow entries to their underlying networks.

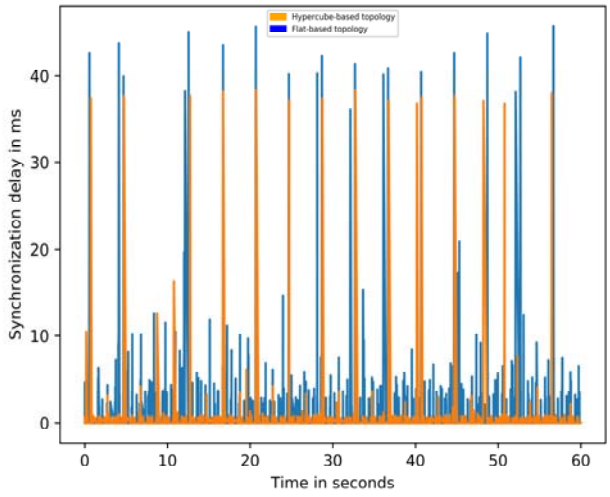

Fig. 5. The synchronization delay inter edge controllers regarding a hypercube-based and a flat-based topology

Figure 5 shows the synchronization delay inter edge controllers during a segment of time of 60 seconds, while adding, removing and updating hosts and flow entries to the underlying networks.

We notice globally the synchronization delay for the flat-based topology goes up to 46 ms, while it doesn't go more than 38 ms in the hypercube-based topology. When digging deeper, we see that some values that go beyond 30 ms, however, the synchronization delay for the most of the time is under the 1.5 ms for the hypercube-based topology, and goes between 9 ms and 2.5 ms for a flat-based topology.

When we sum up the synchronization delay for both topologies, we found that the hypercube-based topology is faster 61.88% than the hypercube-based topology.

It is so clear, why we had such an improvement, because in the case of a hypercube based topology, to go from one edge controller to another, we go through 3 controllers, while in the flat based topology we go through 7 controllers.

B. *The throughput*

In the following, we measure the throughput between the edge controllers within the watching phase for a duration of 90 seconds.

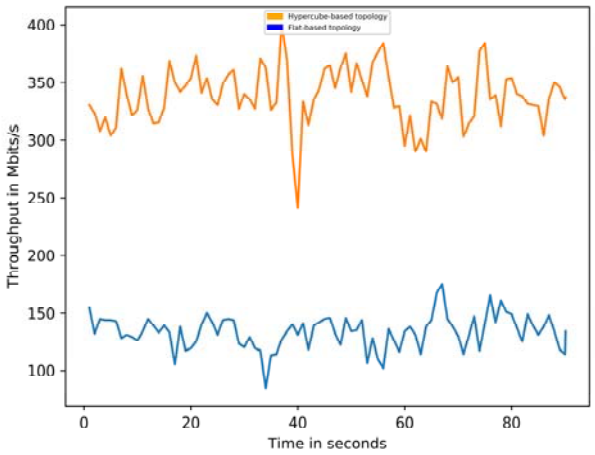

Fig. 6. The throughput inter edge controllers regarding a hypercube-based and a flat-based topology

Figure 6 shows that the throughput inter-controllers for a hypercube-based topology is approximately the twice the throughput for a flat based topology.

When measuring the sum of values to found the percentage of improvement, we have found that the hypercube-based topology optimizes the flat-based topology's throughput up to 153%.

Additionally, knowing that that maximum throughput offered by our Testbed from Atlentic.net is 600 Mbits/s, the hypercube-based topology can reach up to 66% of it, which can be considered very efficient.

C. *The end-to-end latency*

In the following, we measure the end to end latency using a simple ping, for both hypercube based and flat based topologies.

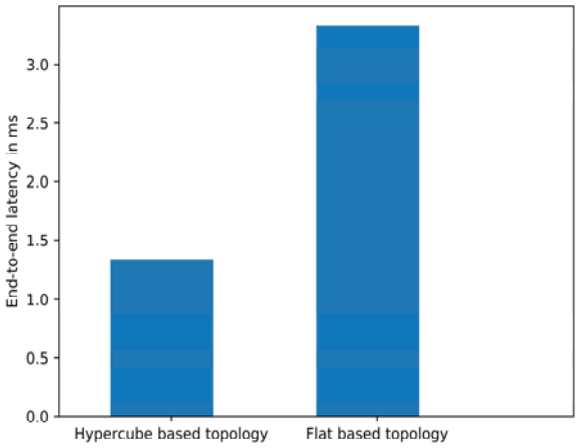

Fig. 7. The end-to-end latency inter edge controllers regarding a hypercube-based and a flat-based topology

Figure 7 shows that the end to end latency for a hypercube-based topology is lower in comparison to a flat based topology.

In addition, we have found that the end-to-end latency for the hypercube-based topology is 59.84% faster than the flat-based topology.

IV. CONCLUSION

SDN multiple controllers are necessary to enhance scalability, security and performance. The way we connect between these controllers may affect the overall performance of the SDN based network.

In this research paper, we have implemented and evaluated the OMCH-SDN topology, which is a Hypercube-based way that we have proposed to interconnect between multiple controllers in an SDN environment. The results demonstrated that the OMCH-SDN model has favorably and significantly succeeded to enhance many network performance measurements, like the synchronization delay, which become more 62% faster in comparison to a flat-based topology. In addition, the throughput has been improved and was able to use more of the available throughput.

In general, the OMCH-SDN model has helped to improve the overall performance of a logically centralized multi-controller SDN network.

REFERENCES

[1] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: a comprehensive survey,” Proceedings of the IEEE, vol. 103, no. 1, pp. 14–76, 2015.
[2] B. Pfaff, B. Lantz, B. Heller, et al., OpenFlow Switch Specification, 2012.
[3] "Open source software for creating private and public clouds.", OpenStack, 2016. [Online]. Available: https://www.openstack.org/.
[4] Open Networking Foundation (ONF); Available from: https://www.opennetworking.org/
[5] M. Smith, M. Dvorkin, Y. Laribi, V. Pandey, P. Garg, and N. Weidenbacher, “OpFlex Control Protocol,” Internet Draft, Internet Engineering Task Force, April 2014. [Online]. Available: http://tools.ietf.org/html/draft-smith-opflex-00
[6] "Puppet - The shortest path to better software", Puppet, 2016. [Online]. Available: https://puppet.com/.
[7] "Chef – Embrace DevOps | Chef", Chef, 2016. [Online]. Available: https://www.chef.io/.
[8] Red Hat, I. (2017). Ansible is Simple IT Automation. [online] Ansible.com. Available at: https://www.ansible.com/.
[9] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “NOX: towards an operating system for networks,” Comp. Comm. Rev., 2008
[10] Project Floodlight. (2017). Floodlight OpenFlow Controller -. [online] Available at: http://www.projectfloodlight.org/floodlight/.
[11] Othmane Blial, Mouad Ben Mamoun, and Redouane Benaini, “An Overview on SDN Architectures with Multiple Controllers,” Journal of Computer Networks and Communications, vol. 2016, Article ID 9396525, 8 pages, 2016. doi:10.1155/2016/9396525
[12] "The OpenDaylight Platform | OpenDaylight", Opendaylight.org, 2016. [Online]. Available: https://www.opendaylight.org/.
[13] ONOS. (2017). Open Network Operating System. [online] Available at: http://onosproject.org/.
[14] M. Schlosser, M. Sintek, S. Decker and W. Nejdl, "HyperCuP — Hypercubes, Ontologies, and Efficient Search on Peer-to-Peer Networks", In: Proc. of Agents and Peer-to-Peer Computing (AP2PC 2002), LNCS 2530, 2002.
[15] C. Chang, C. Chang and J. Sheu, "BlueCube: Constructing a hypercube parallel computing and communication environment over Bluetooth radio systems", Journal of Parallel and Distributed Computing 66 (2006), 1243-1258.
[16] B. Andrus, J. Vegas Olmos, V. Mehmeri, I. Monroy, S. Spolitis and V. Bobrovs, "SDN data center performance evaluation of torus and hypercube interconnecting schemes", 2015 Advances in Wireless and Optical Communications (RTUWO), 2015.
[17] "RFC 2784 - Generic Routing Encapsulation (GRE)", Tools.ietf.org, 2016. [Online]. Available: https://tools.ietf.org/html/rfc2784.
[18] "Quagga Software Routing Suite", Nongnu.org, 2016. [Online]. Available: http://www.nongnu.org/quagga/.
[19] "Business Hosting Services by Atlantic.Net", Atlantic.Net, 2016. [Online]. Available: https://www.atlantic.net/.
[20] M. Team, "Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet", Mininet.org, 2016. [Online]. Available: http://mininet.org/.
[21] “Open vswitch,” http://openvswitch.org.
[22] "Wireshark · Go Deep.", Wireshark.org, 2016. [Online]. Available: https://www.wireshark.org.
[23] V. GUEANT, "iPerf - The TCP, UDP and SCTP network bandwidth measurement tool", Iperf.fr, 2016. [Online]. Available: https://iperf.fr/.