Performance Implication and Analysis of the OpenFlow SDN Protocol

Aliyu Lawal Aliyu, Peter Bull & Ali Abdallah
Centre for Cloud Computing
School of Computing & Digital Technology
Birmingham City University
Email: (Aliyu.Lawal-Aliyu, Peter.Bull & Ali.Abdallah)@bcu.ac.uk

***Abstract*—Software Defined Networks provide the ability to manage networks from a centralised point through separating control plane from the data plane. This brings opportunities in terms of manageability, flexibility and cost savings in network operations. This centralisation, however, also brings about a potentially serious performance bottleneck and poses a scalability issue in high performance networks.**

This paper investigates performance of Software Defined Networks in general, and the OpenFlow protocol, to provide insight into the components of control path delay incurred by packets and ways to optimise flow forwarding. Two Openflow controllers (Floodlight and Pox) were used to validate performance measurements in relation to their theoretical composition. Secondly, the packet processing dynamics of switches, in particular OpenVSwitch are examined, looking at the control packet forwarding behaviour in the kernel module to meet high performance network and traffic engineering demand.

Keywords: SDN, OpenFlow, Delay, Performance and Scalability.

I. Introduction

The emergence of Software Defined Networking (SDN) changes the way networks operate by decoupling control plane from the data plane and delegating it to an external controller, through which network operations are managed [1]. This development causes a drastic change in the networking industry because SDN provides flow-level control and global visibility of flows in networks for fine grained control [2]. This allows network operators to specify their requirements at the application level, thereby making network management easier and introducing new services. These features attract data centres and high performance networks to adopt SDN for the purpose of orchestration and automation e.g real-time virtual machine migration, and optimisation for network virtualisation.

With the adoption of SDN in this environment, two critical issues need to be addressed in order to cope with traffic in data centers and high performance networks:

- The delay accumulated by flows (datapath requests) as they traverse the control path.
- How many flows a controller can process per second.

An additional potential issue that contributes to this latency is software forwarding in switches, which impacts latency sensitive application. These switches are now used as mainstream switches in cloud computing and virtualised environment, with the most commonly used being Open Virtual Switch (OVS)[3], where it is used as an edge switch and operates in high performance networks.

This paper focuses on:

- Evaluating the factors that contribute to latency associated with flow processing in OpenFlow controllers.
- Evaluating the implication of using both hardware and software switches and the significant performance constraint imposed on software switches.
- Proposed solutions for both OpenFlow controllers and the switches to perform optimally in high performance environments.

The rest of the paper is organised as follows: Section II explores the background of the OpenFlow protocol and discusses the related work in OpenFlow performance. Section III provides details of the factors that influence OpenFlow performance and proposed mitigation techniques. Section IV discusses the performance implications of software switches in comparison with hardware switches used in OpenFlow networks. Finally, section V concludes and discusses future directions.

II. Background

Current SDN implementations largely focus on the use of the OpenFlow protocol to carry control messages from the logically centralised controller to the switches in the data plane. The controller obtains a global view of the whole network from the different forwarding elements under its control [4]. This helps in managing the network from a centralised point with complete visibility. However there are proven concepts where the controller implementation is distributed for redundancy, failover and reliability. For instance (ONIX [5] and ONOS [6]), these controllers still maintain global knowledge of the network under their own domain.

Both centralised and distributed SDN controllers offer automated enforcement of network policies with fewer human errors and can automatically react to changing varying network load (thus change in network state) [7]. However an important issue to note is the performance bottleneck introduced in the communication link between the switches and the controller [8]. This is due to the control path (slow path) link between the switch and controller being slower than the hardware based (fast path) Application-Specific Integrated Circuits (ASICs), which forward packets at line rate [9].

Current deployments of SDN within virtualised environments often make use of software switches [10]. These provide network access to virtual machines and provide packet forwarding and processing at the data plane, which is handled by either software or hardware switches.

A need arises to understand in detail the performance implications of software processing, and possible ways to accelerate flows in high performance networks (enterprise and data centre) due to large volume of flows that must be process in few milliseconds to avoid congestion and delay. In an OpenFlow implementation incoming packets from switches invoke the controller path for flow rule set up, which is slow and increase the latency of the path as more packets arrive [11]. This is because the current network architectures are not designed to operate in that manner, and the architecture of SDN introduced new communication entities between the centralised controller and forwarding elements.

A. Related Work

The following are examples of projects attempting to address OpenFlow performance issues:

- OFLOPS [11] is a framework that tests the performance of OpenFlow enabled switches with a remote controlled application. OFLOPS helps in determining the strength and weaknesses in terms of performance between the forwarding engine in the switches and remote application.
- Devoflow [8] is a framework that improves the scalability of OpenFlow by managing the overhead caused by sending packets to control path. The framework modifies the Openflow model and allows it to load balance data center traffic in a more efficient way.
- Scotch-Overlay [9] is a mechanism that leverages high control path capacity of OVS to increase the performance of control path. This consolidates the control path capacity of many switches to be utilised during peak load and abnormal traffic surge.

The frameworks examined did not provide detailed examination of the implication of delays in propagation and flow rule insertion. It is crucial, however, to understand these implications and provide ways to improve and avoid these delays, to support future high performance networking.

III. Performance Implications of OpenFlow

According to [7], the granularity offered by OpenFlow brings about correctness in flow policy implementation without per switch configuration, and near optimal management of flows due to a global network view. Despite this, there is significant performance implication with high performance networks like data centre and enterprise [8][11]. Traffic from data centres are of high volume and commodity switches are not designed to handle thousands of flow entries because of their flow table built capacity. Therefore running large OpenFlow network with complete visibility poses scalability issue for high performance network operation.

The design of OpenFlow brings about overhead, and costs ranging from polling for flow statistics from the switches by the controller and the distributive system cost of different switches requiring the controller to handle flows. A single controller with wide visibility of all flows poses a centralised bottleneck and consequently scalability becomes an issue [8]. Section III-A, III-B and III-C discuss in detail the factors that cause OpenFlow performance issues.

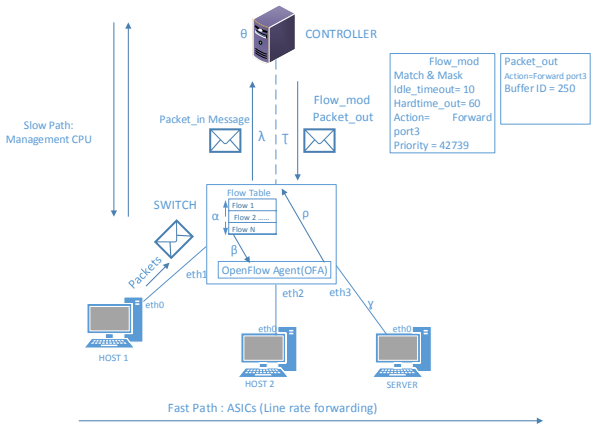

Fig. 1. OpenFlow Operation

A. Implementation Overhead

In OpenFlow implementation, when a packet arrives (`Packet_in`) the switch checks the flow against known flow entries. These flow entries are looked up in either the hash tables or linear tables for software switches, or Ternary Content Addressable Memory (TCAM) for hardware switches [9]. To quantify this:

- Let α designate the time it takes for the packet lookup process when there is no match in flow table, the table miss is applied to the packet.
- Let β be the latency in moving the packet to OpenFlow Agent (OFA).
- Let ρ be the time for the OFA to encapsulate the `Packet_in` message to the controller.
- Finally, λ is the delay incurred through transmitting the `Packet_in` message across the control path to the controller.

Depending on the implementation whether using hardware or software switches, bandwidth between data and control plane is finite, as is the compute capacity. This can hinder the rate at which flows are set up, as only thousands of flows per second can be implemented. Figure 1 depicts the described process:

Bandwidth in the control path (slow path) is negligible due to costs, unlike data path (fast path) which uses ASIC line card on hardware to forward at line rate which is fast. For instance the line card forwarding rate of a HP procurve 5406zl is 320Gb/s (fast path) as compared to the CPU slow path which is just 80 Mbps (slow path). Thus the control path is slower than data path [8][12][11].

B. Packet Modification Delay

Flow-based networking requires the attention of the controller periodically, which causes the accumulation of additional delay and latency in the control plane [13][14]. The controller set up flow rules based on organisational policy and global network state, and in this scenario θ represents the time it takes for the controller to set up flows.

C. Flow Table Update Delay

Installing flow entries in a flow table induce significant delay [12]. The flow table in OpenFlow switches serves as the memory that holds all active flows and is similar to the Forwarding Information Base (FIB) in conventional routers. The controller pushes flow rule settings back to the switch and the flow table updates and reorders the rules based on the idle timeout in the flow table and implemented TCAM algorithm. The delay incurred for this process can be referred to as τ.

After a `flow_mod` operation the switch updates it table and (if required) reorders the flow entries. An OpenFlow barrier message is received to ensure that change is reflected on the data plane. For hardware the completion time for successful FIB update is in the hundreds of microseconds range [15]. There is an accumulated delay associated with flow insertion in the flow table, and it grows as the number of flow entries increases. This is the time it takes for a packet to be processed by the controller and push down to the switch for forwarding based on the rule sets defined. `Flow_add` takes less time to update unlike `flow_mod` which is found to cause a higher delay in some hardware due to cross vendor variation in OpenFlow implementation [2][11]. As more entries are populated in TCAM, the more complex updating of flow entries becomes. Irrespective of flow type, however, software switches are not affected by any additional delay for flow insertion caused by `flow_add` or `flow_mod`. For an exact-match flow entry, forwarding is done at line rate (hardware) or hash tables for software, and the last delay is forwarding to the exit interface referred to as γ. Summing up the total propagation and processing delay, the flow rule insertion delay can be calculated as the summation of all delays as follows: Let flow rule insertion delay = η,

$$\eta = \alpha + \beta + \rho + \lambda + \theta + \tau + \gamma \quad (1)$$

In a typical reactive flow set up that requires the attention of the controller for any ingress packet, then all the delay parameters affect the packet that is being processed which makes equation (1) to be true.

For proactive flows there is no invocation of the control path because flows are set beforehand and the controller has prior knowledge of the arriving flows. Proactive flows are initiated by stimuli from external network events like movement of virtual machine from one physical to another, backup, load balancing of elephant and mice flows etc. The first packet from an external source that triggers a proactive flow will not experience delay caused by α, β, ρ and λ because they are not triggered from the switch to controller but rather from the application to controller. The total delay experienced by proactive flows are θ, τ and γ, because α, β, ρ and λ all equal to zero in a proactive flow setting as shown in equation (2).

$$\eta = \theta + \tau + \gamma \quad (2)$$

To depict the effect of the control path delay, the Mininet [16] emulation environment is used. Both the links and the hosts are instantiated in network namespaces a feature of Linux kernel which is a virtual container. Internet Control Message Protocol (ICMP) ping request and reply are used to measure the delay using the POX and Floodlight controllers in the Mininet emulation environment. The test is carried out on a PC with Intel Core i5-4200M and 16GB RAM. The network setup is depicted in Figure 1, which consists of a single switch with three hosts and a centralised controller. For the complete flow rule insertion delay that satisfies equation (1), the first `packet_in` event invokes the control path to forward packet to the controller. This triggers the subsequent delays that lead to a successful implementation of flows in the switch flow table.

The resulting Round Trip Time (RTT) delay between the two hosts is shown in Figure 2. The first `packet_in` event, where packets are sent to the controller, resulted in a delay of 32ms using POX and 15ms with Floodlight. This is in comparison to a delay of average mean of 0.302ms for Floodlight and 1.253ms for POX. Note that the higher delay in Pox is the result of flow hard timeouts causing the need for the re-installation of flow rules.

TABLE I
PROPAGATION DELAYS

α	Flow rule lookup process by switch using software tables, or TCAM in hardware.
β	Time it takes for a non-matching flow to be sent to the Opeflow agent (OFA) in the switch.
ρ	OpenFlow agent encapsulate packet and generate a Packet_in message.
λ	Delay caused by invoking the control-path.
θ	Time for controller to determine how packets should be handled based on established policy settings or global network state.
τ	Delay incurred for pushing the flow rule back to the switch to update the flow table.
γ	Switch forwarding based on flow rule.

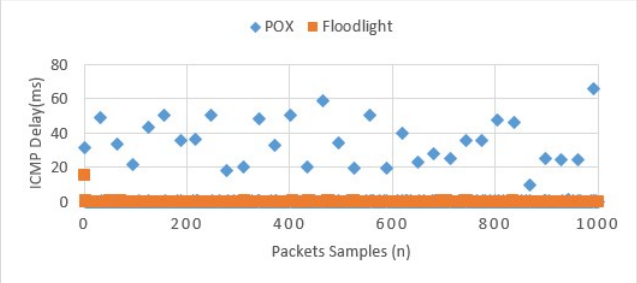

Fig. 2. Flow Rule Insertion Delay

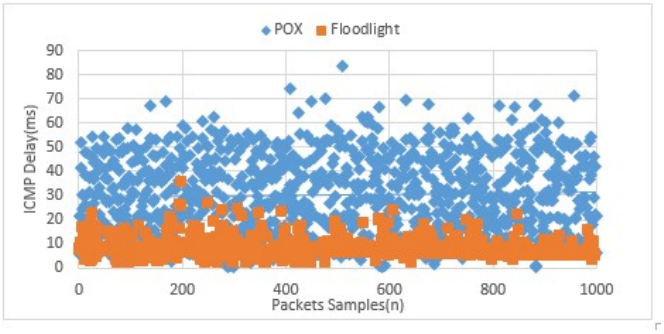

Fig. 3. Control Path delay for First `Packet_in` Event Only

TABLE II
CONTROLLER PROGRAMMING LANGUAGES AND DOMAIN OF APPLICATION

Language	Controller	Domain	Thread
Java	OpenDayLight	Data Center	Multi
Java	ONOS	Telecom / Data Center	Multi
Java	Floodlight	Campus	Multi
Java	Iris	Telecom	Multi
Python	Ryu	Campus / Research	Single
C	MUL	Data Center	Multi
C++	Runos	Telecom / Data Center	Multi
C, Python	ONIX	WAN / Data Center	Multi
C, Ruby	Trema	Data Center	Multi

The delay found from contacting the controller is further analysed in Figure 3, where stream of `packet_in` messages were sent to the controller. It can be seen that the delay varies greatly, due to the emulation environment used, and for the fact that this is reliant on the invocation of a software based controller.

D. OpenFlow Controllers

The type of controller used influences the performance of the network. The performance of a controller is measured based on number of `packet_in` messages processed per second (throughput) and the average time the processor takes to process events (latency). [8].

Another key factor to consider is the programming language used for controller development. Java is one of the best choice due to its cross platform, multi-threading support and modular capabilities. Python does not scale well due to its inability to multi-thread. C and C++ controllers provide improved levels of performance, however, they tend to have issues of memory management. The .NET languages are not cross platforms and limited to single environment operation. Table II presents the programming languages used by some of the controllers with their domain of application, and threading ability.

The throughput of multi-threaded controllers depends on two factors, firstly the algorithm responsible for distributing flows across various threads, and secondly the the mechanism used in network interaction or the respective libraries used by the controller [17].

IV. DATA PLANE DEVICE PERFORMANCE

OpenFlow switches are currently implemented either in hardware or software based virtualisation. The following subsections discuss these switch types in further detail.

A. Hardware Switches

According to [10], hardware based switches benefit in terms of performance from efficient packet offloading mechanisms integrated in their Network Interface Cards (NICs). Hardware NICs have the ability to migrate packets to Direct Memory Access (DMA) and even CPU cache, referred to as Direct Cache Access (DCA), without the help of the CPU [18]. NIC can also support a modern multicore architecture e.g Receive Side Scaling (RSS), which places packets among different CPU queues for processing. These features are absent in software [19].

As [2] discusses, in traditional switching hardware, where the control, data and management planes are vertically integrated, the control plane minimally degrades the data plane performance, and forwarding is done at line rate for all flows. Conversely, in SDN OpenFlow enabled switches where the control plane functions are delegated to an external controller, a delay is experienced in flow set up [14]. However, according to [10], the speed at which this process happens depends on the following:

- General purpose CPUs that process packets at slower rates than special purpose ASICs.
- TCAM technology [20], which has a significant effect on the size of hardware flow tables, flow rule insertion rate, flow table entries, and flow rule complexity [8].
- The switch firmware, which can either buffer or delay flow insertion. In addition, some firmwares are capable of using a software table with predefined guidelines on migrating flows to hardware tables in the event of software table saturation.

For instance OVS allows 200,000 rule with flow set-up rate of 42,000 per second [21]. Conversely HP ProCurve 5406zl with K.15.10.0009 firmware support only 1500 flow table entries in hardware with 275 rule set up per second[22]. Due to these factors network emulators cannot make generalised conclusions of the performance fidelity of SDN networks because it is at its early stage unlike the traditional switches that have matured over the years and performance assumptions can be made with little variation in implementation.

1) Hardware Design Solution and Issues: Hardware design intended for OpenFlow will drastically improve performance but the consequent to this comes with ASICs cost, complexity and power consumption. Also, going by Moore's law which needs processor replacement after a certain period of time would not give a permanent relief. The control plane bandwidth will increase if the processing power increases. A possible alternate solution to this as argued by [2][9] is to use software based switches in commodity hardware devices because they have higher control plane bandwidth, but this would not be a cost effective solution to enterprise.

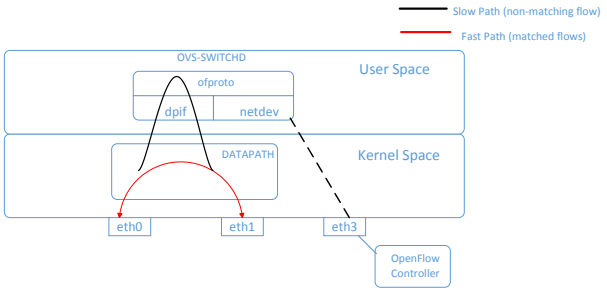

Fig. 4. OVS Architecture

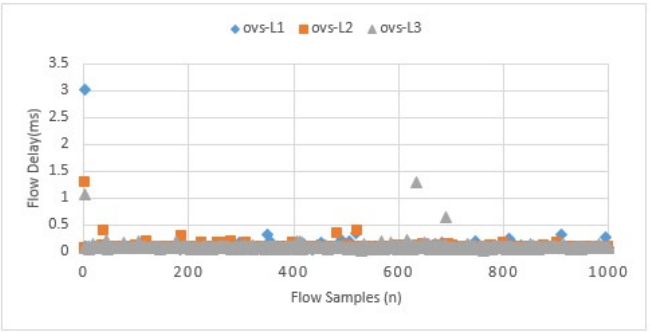

Fig. 5. OVS-Kernel matching for L1, L2 and L3

B. Software Switches

Virtual switches have become an integral part of any virtualised computing set up ranging from cloud computing and network virtualisation [10]. The most adopted and used virtual switch is Open Virtual Switch (OVS) which is vendor independent, flexible, open source and licensed under Apache 2.0 [3]. OVS is integrated in different hypervisors like Xen, KVM, and Qemu, and serve as a software switch [23]. OVS is heavily used in SDN implementations and cloud computing frameworks like Openstack.

1) Factors Affecting Performance of OVS: OVS leverages the UNIX/Linux kernel to implement OpenFlow switch functionality. Flow entries are stored in memory, and according to [24], forwarding is several orders of magnitude lower than hardware. In OVS architecture, the switch daemon (ovs-*vswitchd*) situated in userspace and data path (kernel module) are the most important component that affect performance. The data path (fast path) which is the kernel module is responsible for packet forwarding based on rules maintained in flow table to forward matched flows with corresponding actions.

vswitchd manages flows in the Kernel module. It contains various components (ofproto, netdev and dpif) as shown in Figure 4 that help configure OVS instances, connects external OF controllers and communicate with the underlying system.

2) Performance Implication: Delay and latency are observed in OVS on packets arriving without a match in kernel module, because packets are copied, encapsulated and sent to the user space which is slower and accumulates more delay. As more packets arrive to the kernel module without a matching rule in the cached entries, the higher the delay and latency in setting up and installing flows, since packets must go to vswitchd for flow rule set up. Additionally, the classifier in userspace is responsible for the lookup action for unmatched flows in the kernel module, there is latency associated with the lookup process which add up to overall performance constraint.

The userspace is responsible for managing all the flow set up in kernel including statistics and flow monitoring. Querying of managed flows from the kernel module is costly to the switch, because forwarding is carried out by management CPU which is slow as compared to ASIC packet processing.Figure 5 depicts the performance behaviour of forwarding functions

TABLE III
MAXIMUM RULE INSERTION

Layer	Maximum Delay (ms)
L1	3.011
L2	1.291
L3	1.297

TABLE IV
MINIMUM RULE INSERTION

Layer	Minimum Delay (ms)
L1	0.082
L2	0.081
L3	0.032

in the kernel module, without controller assistance from user-space.

The forwarding measurement is carried out for traffic in physical, data and network layer of the OSI model thus Layer 1 (L1), Layer 2 (L2) and Layer 3 (L3) matching. Layer 1 matches on physical ports, while Layer 2 uses MAC addresses and Layer 3 uses IP addresses. This is implemented in the Mininet environment, as seen from Figure 5.

The first sets of flows that arrived have higher delays due to flow setup latency. Subsequent flows matched on the already cached entries in kernel module. The setup delay is due to netdev, which is a module responsible for managing kernel module activities locally, and when a flow hits the kernel there is an associated delay in communication time between the kernel module and netdev. The respective rule insertion delays observed for L1, L2 and L3 are seen in Tables III and IV.

Note that the test is carried out in a virtualised environment

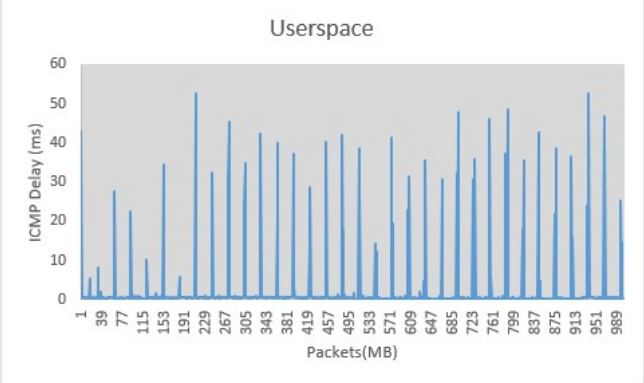

Fig. 6. User Space Packet Delay

(Mininet) leveraging operating system resources. Performance is therefore subjected to constraints like buffer size, delay, bandwidth and simulated packet loss.

For the vswitchd the delay is measured using POX as the OpenFlow controller as seen from Figure 6, an increase in delay is experienced due to additional load on the OVS kernel for encapsulating non-matching flow to userspace which is slowpath. With packets at userspace the delay turn out to be high with first packet having 39.2ms, and minimum delay at 0.101ms. And the mean for the userspace is 2.299ms. There are sudden spikes of flow delay set up at various intervals, these are due to flow timer expiration. The flows in kernel modules are managed from the vswithd and there is an associated cost in setting up flows due to the delegation of control to external OpenFlow controller.

V. Conclusion & Future Recommendations

OpenFlow networks are gradually deployed in current networks with most vendors providing support for the protocol in their networking devices. There is a gap, however, between these devices that are designed in a similar pattern to traditional network switches, and the requirements of a Software Defined Network.

A. Future Recommendations

The control path delay incurred by packets are due to the separation between control and data plane which is new to the architecture of networking devices and processors. Possible ways of accelerating flow forwarding in SDN are redesigning the software stack of OpenFlow agent (OFA) and the architecture of processors to have more power in handling control path request. These will increase throughput and reduce latency by enhancing flow processing and offloading at high rate.

Flow tables are memories that store flow entries. Hardware devices have small TCAMs to store flow entries but forward at line rate, however software switches have room for more flow entries but forward at slow rate. These problems can be alleviated in hardware devices by redesigning TCAMs to have bigger memory space and increasing the management CPU processing ability for the software switches.

Finally, the controller choice plays a major role in influencing flow processing capability. Controllers such as OpenDaylight, ONOS and Floodlight which are multi-threaded have the ability to leverage multiple CPU cores in processing flows. These make them scale as more flows are processed.Controllers should be modular with multi-platform support and whether distributed or centralised depending on the network implementation.

References

[1] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: a retrospective on evolving sdn," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 85–90.

[2] D. Y. Huang, K. Yocum, and A. C. Snoeren, "High-fidelity switch models for software-defined network emulation," *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 43–48, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2491185.2491188

[3] "Openvswitch," 2015. [Online]. Available: http://openvswitch.org/

[4] S. Latifi, "Emulating E nterprise N etwork E nvironments for F ast T ransition to Software-Defined Networking," pp. 294–297, 2014.

[5] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, Others, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," *OSDI, Oct*, pp. 1—-6, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924968

[6] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[7] 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.

[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.

[9] A. Wang, S. Chen, and T. V. Lakshman, "Scotch : Elastically Scaling up SDN Control-Plane using vSwitch based Overlay," pp. 403–414, 2014.

[10] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Performance characteristics of virtual switching," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. IEEE, 2014, pp. 120–125.

[11] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "Oflops: An open framework for openflow switch evaluation," in *Passive and Active Measurement*. Springer, 2012, pp. 85–95.

[12] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "OpenFlow switching: Data plane performance," *IEEE International Conference on Communications*, 2010.

[13] K. Benton, L. J. Camp, and C. Small, "OpenFlow vulnerability assessment," *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, pp. 151–152, 2013.

[14] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks." in *NDSS*, 2013, pp. 1–16.

[15] A. Shaikh and A. Greenberg, "Experience in black-box OSPF measurement," *Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement Workshop IMW 01*, pp. 113–125, 2001.

[16] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, pp. 1–6.

[17] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers," *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia on - CEE-SECR '13*, pp. 1–6, 2013.

[18] Intel, "Intel Inut Output Acceleration Technology," 2014. [Online]. Available: http://www.intel.com/content/www/us/en/wireless-network/accel-technology.html

[19] L. Rizzo, M. Carbone, and G. Catalli, "Transparent acceleration of software packet forwarding using netmap," *Proceedings - IEEE INFOCOM*, no. 257422, pp. 2471–2479, 2012.

[20] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[21] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *12th USENIX symposium on networked systems design and implementation (NSDI 15)*, 2015, pp. 117–130.

[22] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about sdn flow tables," in *International Conference on Passive and Active Network Measurement*. Springer, 2015, pp. 347–359.

[23] S. Hamadi, I. Snaiki, and O. Cherkaoui, "Fast path acceleration for open vSwitch in overlay networks," *2014 Global Information Infrastructure and Networking Symposium (GIIS)*, pp. 1–5, sep 2014.

[24] B. Pfaff and B. Davie, "The open vswitch database management protocol," 2013. [Online]. Available: http://openvswitch.org/