Does OpenFlow Really Decouple The Data Plane from The Control Plane?

Thiago M. Peixoto*, **Alex B. Vieira***, **Michele Nogueira**‡, **Daniel F. Macedo**†
*Computer Science Department, Federal University of Juiz de Fora, Brazil
†Computer Science Department, Federal University of Paraná, Brazil
‡Computer Science Department, Federal University of Minas Gerais, Brazil
Emails: thiagomoratori@ice.ufjf.br, alex.borges@ufjf.edu.br, michele@inf.ufpr.br, damacedo@dcc.ufmg.br

***Abstract*—Software Defined Networks (SDNs) offer flexibility to current networks, allowing operators to manage network elements using software on an external server. SDNs are founded on a key feature: the separation of the control plane from the data plane. OpenFlow is the most popular SDN southbound interface today. However, does OpenFlow really decouple the data plane from the control plane? This is the leading question in this work. The literature has sought to quantify the impact of OpenFlow commands from control plane on data plane performance. Particularly, we argue that it is possible to damage the date plane by too many flow updates. Attackers, for instance, can use this effect in a cloud environment to reduce the performance of a collocated virtual network. However, it is not clear what is the exact impact of this coupling on production hardware and software switches. We investigate this through experiments, under representative scenarios, and propose a threshold mechanism to mitigate the effect of malicious administrators. We have observed that both hardware and software switches suffer from this limitation, presenting an average RTT degradation of up to 12.35% in the hardware switch, and 25.9% on the software switch. Finally, the proposed mechanism mitigates the lack of decoupling and malicious behavior.**

***Index Terms*—SDN, Performance Analysis, Resilience.**

I. Introduction

Software Defined Network (SDN) is a paradigm that offers flexibility and agility to current networks, facilitating network management and enabling efficient network configuration. SDN separates the control plane (routing process) from the data plane (forwarding process of data packets) [1]. Hence, the network intelligence is centralized in controllers, whereas switches are responsible for packets forwarding in accordance to decisions made by the controllers. This simplifies network management, improving network performance.

SDNs rely on a key feature: the separation of the control plane from the data plane. However, *does OpenFlow really decouple the data plane from the control plane?* This is the leading question in this work. Indeed, a number of factors may influence SDN data plane performance, from switches architecture to Openflow particularities and its distinct versions. Moreover, ordinary switches have influenced the OpenFlow design and, hence, control and data planes may not be perfectly decoupled. Since both planes may share computational resources, such as CPU and memory, we argue that it is possible to damage data plane performance by erroneous, or too many, control plane actions, as flow updates.

Many datacenters employ software and hardware SDN switches today [2]. Software switches forward traffic among VMs in the same physical machine, whereas hardware switches forward traffic among physical machines or racks. Hardware switches may employ Ternary Content-Addressable Memories (TCAMs) to perform rule matching. However, some of the matches and other SDN operations might still require the use of a generic CPU. Some switches store part of the flow table in TCAMs (e.g., for wildcard matching), while others store in the SRAM (e.g., exact matches, which may resort to hashing) [3]. Since the flow table update processes compete for CPU with the forwarding operations, it is possible to affect data packet QoS when the controller sends a high number of requests to the OpenFlow switch.

There are studies that assess the performance of OpenFlow controllers [3]–[5]. But, only a few focus on the data plane of OpenFlow switches and their implementations [3]. Those studies jointly evaluate the OpenFlow data plane and the SDN controller, however they do not evaluate the impact of actions in one plane to the other.

This paper evaluates the decoupling level between the SDN data and control planes, and it presents a mitigation mechanism to reduce damages the interference a plane causes to the other. Particularly, we conduct a series of experiments using OpenFlow 1.0 on hardware and software switches to quantify the impacts a plane imposes on each other. Results show that SDN control and data planes are not completely decoupled, and both software and hardware switches suffer impacts from the control plane actions. Notably, the impact on software switches is higher due to a tight reliance on the CPU. In fact, under the presence of malicious flows, software switches present almost 26% higher RTT, while hardware switches present up to 12.4% higher RTT. Our results also show that flowtable modifications impact the data plane QoS and, since OpenFlow does not define the flow computation model, we expect the same issue to distinct OpenFlow versions.

Finally, we discuss on how to avoid those unintended effects under different scenarios. We propose a controller component to overcome the performance degradation problem by rerouting flows. On average, the proposed mitigation mechanism reduces the performance degradation.

This paper proceeds as follows. Section II provides a brief background on OpenFlow. Section III presents the evaluation

of the rule changes effect on the performance of data flows. Section IV details the proposed mechanism. Section V discusses related works. Finally, Section VI concludes the paper.

II. OpenFlow: Background

OpenFlow [6] is an open protocol that allows a central entity to program the flow tables in switches and routers. The protocol defines an API that interconnects the network equipment (data plane) to a controller (control plane) [7], [8]. The SDN controller pushes down changes to the switch/router flow-table allowing to network administrators a number of actions, as optimizing traffic and testing new configurations or applications. OpenFlow has become the *de facto* platform in SDN, being used in the majority of SDN deployments and research [9], and it is available in equipment from different companies, as HP, NEC, Cisco, Juniper, Huawei and others [7].

When a packet arrives at an OpenFlow switch, it processes the packet following three major steps:

1) The OpenFlow switch tries to match the current packet to any flow table entry rule. OpenFlow allows a number of rule components (e.g., IP/Port source/destination, switch port and transport layer protocols – UDP/TCP). Moreover, OpenFlow allows explicit match or a wildcard match on rule fields. A wildcard match means the switch does need a perfect match of the values in a specific field. However, it also supports an explicit match, that demands a perfect match between rule and the flow.
2) If the flow matches an existing rule, the switch follows the existing flow table entry action. For example, it may forward the packet to a specific port.
3) If the flow mismatches all existing rules, the switch sends the packet to the SDN controller. The controller processes the packet and sends back to the switch a new rule to be installed in its flow table. Then, the switch takes the proper action to treat the packet.

These OpenFlow features allow network administrators to dynamically reconfigure the network. There are several OpenFlow controllers available, such as NOX/POX, Pyretic, ONIX [9]. Each distinct controller allows to access the OpenFlow API using different programming languages, as C, C++, Python and Java. Despite the existence of newer versions (i.e., version 1.5), version 1.0 is still the most popular.

III. Evaluating The Decoupling Level Between Data and Control Planes

OpenFlow provides a programming interface, however it does not how flows and flowtable updates should be processed. Since control plane and data plane actions are performed by the same same processing units, one may affect the other's processing time. Hence, in this paper we will show that the lack of standardized requirements for the isolation of the computing tasks of the control and data planes may generate adverse affects, which could be used by an attacker to degrade the performance of the network. This section presents the methodology employed to investigate the decoupling level between the control and data planes. This section shows the evaluation results considering both software and hardware switches.

A. Evaluation methodology

Fig. 1 shows the network topology employed in all the experiments. The network comprises of five devices connected to an OpenFlow switch, from which a transmitter and a receiver communicate. A SDN POX controller and two other hosts generate flows that trigger control plane updates. The network employs POX [10], following OpenFlow, version 1.0 once it is still the most popular. We expect our results generalize to other OpenFlow versions once OpenFlow does not define a flow computation model and, regardless the version, it acts just as an interface to program flows. Moreover, due to space constrains, we do not evaluate other popular controllers. Again, we expect similar results for other controllers, since our observations are tightly related to the architecture of the switches, and not to the controllers.

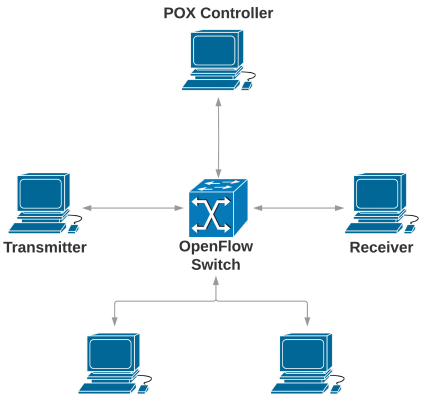

Fig. 1: Evaluation topology schematic

The evaluating topology follows two different scenarios: (i) a software-based virtualized one, and (ii) an off-the-shelf based one. In the software-based virtualized scenario, the network switch employs the Open vSwitch version 2.3.0. All other devices, including the controller, source and destination hosts, are virtual machines. A single Intel I7 computer keeps all virtual machines, with 32 GB of RAM, 2 SSD 256 GB hardrive and a core i7-4790. In the off-the-shelf based scenario, an Extreme Summit x460 switch is the basis to run the ExtremeXOS 15.4.2.8 Operational System, in a single core processor of 500 MHz. Each remaining device, including hosts and the controller, runs on top of light Linux (Ubuntu) configuration, with 512 MB of RAM and a single core, presenting enough resource to properly handle the evaluation.

A set of data transfers between transmitter and receiver occurs. In parallel, a set of network operations triggers actions in the network controller, such as the insertion, removal and update of flow rules in the OpenFlow switch. Hence, the results are only biased by the implementation of the OpenFlow switch flow table and by the communication between controller and switch. In case of rule insertion, the flow table starts empty

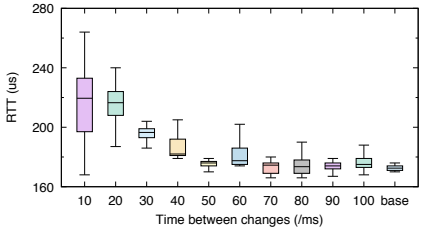

Fig. 2: Insertion of Rules (software)

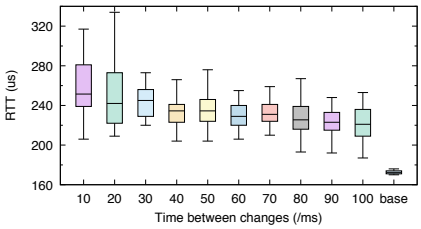

Fig. 3: Rule Removal (software)

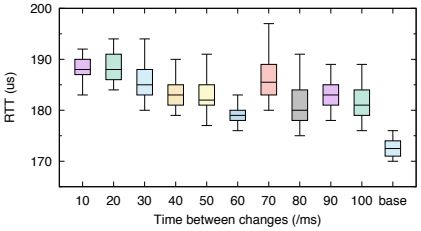

Fig. 4: Changing Rules (software)

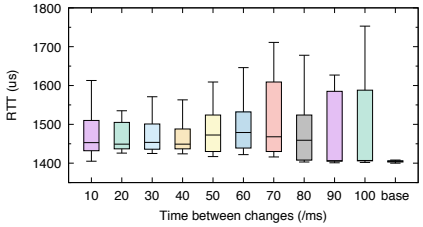

Fig. 5: Insertion of Rules (hardware)

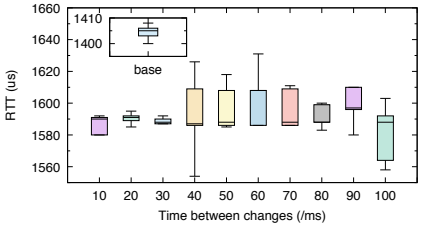

Fig. 6: Rule Removal (hardware)

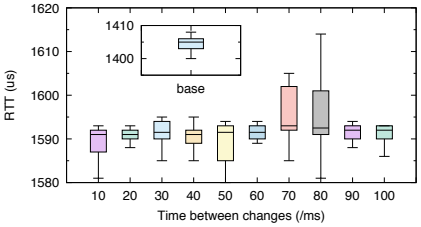

Fig. 7: Changing Rules (hardware)

in the switch. Then, rule inclusion occurs based on the well-known source/destination IP and port 4-tuple to isolate the effects of wildcards [3]. Rule insertion happens by varying source and destination IPs and ports at the desirable rate until the switch reaches the maximum number of rules it can properly handle. Hence, the maximum number of rules, while operating on OpenFlow mode, is of 750 k for the software switch, and 1200 for the hardware switch. Unless we tell otherwise, we discard the warm-up and cooling-off periods, each one corresponding to 10% of total experiment.

Rule deletion follows also variations in the tuple: source and destination IP and their ports. After a warm-up period, the removal process starts at a given rate. Finally, in order to update (change) rules, first the switch selects randomly one existing rule and them starts the rule deletion process.

To probe network conditions, the transmitter sends 1,000 data packets – at a rate of 10 packets/s – to the receiver, which in turn acknowledges each packet reception to the transmitter. The time between consecutive operations also varies (i.e., the inclusion/change/deletion rate) from 10 ms to 100 ms. A baseline case serves as reference for results comparison. In this case, the controller is just influenced by the flows between the transmitter and the receiver, and there are no parallel flows that may trigger new rules, influencing the original flow QoS. Results ignore the initial and final 10% of experiment periods (transient periods). Unless told otherwise, results are an average of 30 repetitions for each scenario setting with a confidence interval of 95%.

B. Evaluation results

1) Software-based scenario: This subsection shows the evaluation results for the decoupling level between the data and control planes on a software-based scenario with a virtualized switch. Figs. 2, 3 and 4 present box plots for the round trip time (RTT) between the two communicating devices (transmitter and receiver from Fig. 1). The time between consecutive actions generated by the controller varies. Box plots show median RTT value, confidence interval, and the 25^{th} and 75^{th} percentiles.

Fig. 2 shows that lower the time between consecutive rule inserts, higher is the RTT on the control plane. Indeed, while the control plane imposes a low load to the switch, RTT is still statistically equivalent to the baseline case, where the controller is just influenced by the existing flow between the transmitter and the receiver. For rule insertion with an interval lower than 50 ms, the addition of flow table entries impacts RTT. Rule insertion impacts the RTT by 7.38% on average (for all experiments), when compared to the baseline case.

Fig. 3 shows that the rule deletion also impacts the data plane. Deletion at any of the evaluated rates increases RTT between the transmitter and the receiver. As for the rule insertions, deletions also tend to increase the RTT for situations in which one observes a shorter time between consecutive actions. Deletions always present a higher impact when compared to insertions for all evaluated rates. When compared to the baseline case, RTT increases of 25.9% for all experiments. In Fig. 4, the RTT between communicating devices is slightly worse at any rate of rule modification when compared to the baseline case. Unlike the previous cases, there are no clear trends of growth to the RTT when the interval among actions decreases. In this case, RTT increases on average of 6.03%.

2) Off-the-shelf hardware scenario: This subsection shows the evaluation results for the decoupling level between the data and control planes on an off-the-shelf switch. Fig. 5 presents box plots regarding to rule insertion in a hardware-based switch, while the time between two consecutive network

controller actions varies. Differently to the software-based scenario, RTT does not present any increase trends when shortening the time between consecutive insertions. In fact, to a 95% confidence level, almost all experimental results are statistically equivalent, presenting interleaved confidence intervals. Only periods between two consecutive insertions lower than 80 ms present higher RTT than the baseline case. However, one can observe a wider RTT variation, in all experiments, when compared to the baseline case. This indicates that the control plane actions (insertions) impact the network jitter. The insertion of rules impacts the RTT by up to 6.5% in the analyzed settings.

The removal and update of rules clearly result in higher RTTs for all experiments, when compared to the baseline case. Removals increase the overall RTT of 12.35%, while rule updates increase the RTT by up to 14%. There are no clear trends, neither to increase or decrease of RTT values, while the time between two consecutive controller actions varies. For both, RTT variation also increases, indicating that these actions impact on network jitter.

3) Summary and further discussion: Results indicate differences between software- and hardware-based switches. Overall, software switches seem to present a higher impact on the data plane, with a clear trend to increase network RTT for more frequent control actions. The control plane also impacts hardware switches. In this case, despite the higher RTT and variation (jitter), one cannot observe a clear increasing trend.

The difference with regards to software and hardware implementations is mostly due to the memory organization in hardware and software switches. Hardware-based switches usually employ TCAMs and perform parallel searches, while software-based switches employ DRAM, and perform linear search or use hash tables [3]. Due to the difference in the computer system architecture (memory organization, processors and how each system performs I/O operations), these results do not compare the absolute performance of hardware-based versus software-based switches.

IV. Mitigating the Effects of Limited Plane Decoupling in OpenFlow

This section presents a mitigation approach to cope with the performance degradation resulted from the existing coupling level between the data and control planes. Also, the mitigation approach handles possible attacks on a Cloud environment that may take advantage of the weaknesses of the existing coupling level between the planes. To understand the possible attacks, one can suppose that a given company A hires a SDN cloud provider to host its platform. Knowing the provider hosting the company A, another given company B, competitor of A, creates a virtual SDN network of its own to attack company A. The virtual SDN yields flow insertions and deletions at maximum rate[1], slowing down packets of the company A. The performance degradation can also be unintended, i.e., in a network running many services over ONOS [11], a misconfigured or buggy service that generates too many flow updates.

Software switches can be modified to employ separate cores for flow updates and for flow matches. This change, however, may induce delays because of the spinlocks that try to avoid concurrency problems. There are two ways to cope with those situations in today's hardware switches. First, the traffic from affected services can be rerouted to other switches. This approach is evaluated in this section in detail. Second, if the problem takes place in a multi-administrator environment, the controller should identify ill-intentioned administrators to revoke their access to the network.

A. Using flow rerouting to mitigate attacks

In a nutshell, the proposed mitigation approach consists in executing a component on an OpenFlow controller which continuously monitors the network and the remaining components. If a misconfigured component intervenes the system —decreasing data plane performance—, the *protection* component reroutes flows to avoid the switches being influenced by the former misconfigured component.

We propose a solution based on a threshold. Algorithm 1 presents its steps. If the amount of rule modifications of a certain component is too high, then this component is marked as *suspect*, i.e., it is either misconfigured or intentionally attacking the network. Once such component is identified, the controller reroutes the flows of other components to avoid the switches suffering from a performance degradation. This proposal does not block the suspect module, since it is not possible to be certain that this behavior is anomalous.

From Algorithm 1, the traffic throughput determines if the data plane is being influenced by data plane actions. Differently from RTT, a switch traffic throughput can be easily accessed by the controller. Moreover, during an attack, one may expect that while RTT increase, the throughput decrease. Thus, if the throughput decreases below a given threshold, the network enables the backup switch, transfers the current suspect rules to the backup switch and disables the previous attacked switch.

Data: Throughput on the possible suspected switch
Result: Reroute from the possible suspected switch to the backup switch
Initialization;
while *there is traffic in the suspected switch* **do**
  reads new throughput on the suspected switch;
  verifies previous throughput value;
  if *new throughput - previous throughput / previous throughput* $\leq$ *decrease threshold* **then**
    enable backup switch;
    transfer flows from attacked to backup switch;
    disable attacked switch;

Algorithm 1: Approach to mitigate attacks.

[1]The maximum number of rules, while operating on OpenFlow mode, is of 750 k for the tested software switch, and 1200 for the tested hardware switch.

B. Evaluation setup

To evaluate the mitigation proposal, the test topology uses Intel based PCs (hosts and controllers) and a virtual switch (Open vSwitch) also running in an Intel PC. It is setup as shown in Fig. 8, with the following entities: a POX **controller**, which is monitored by the protection component; an **attacked (suspected) switch**, which suffers from a degraded performance; a **backup switch**, serving as an alternative route for the packets; and **one receiver and one transmitter** nodes.

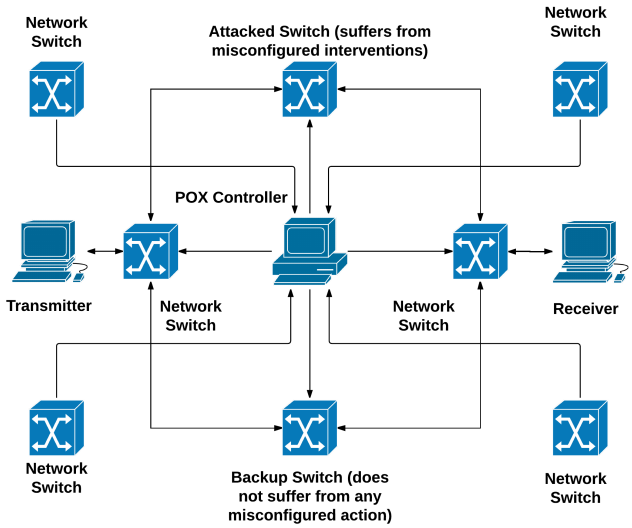

Fig. 8: Evaluation scenario for the proposed mitigation strategy

Evaluation results are from three different scenarios: *environment without attack*; *environment with attack*, in which the controller does not attempt to mitigate the attack; and *environment with attack and switch exchange*, where the protection component reroutes the flows to avoid the attack.

For the software switch, rule deletion is the OpenFlow operation that generates the larger impact on data plane performance. Therefore, to verify the performance of the proposed protection component, evaluations address rule deletion following the same methodology presented in Section III-A. Due to space constrains, this section presents only results to the system under a high rate of rules deletion (one removal every 10 ms) and a threshold of 12% in the decrease of network throughput. This value has been chosen because it is close to the mean value observed in the experiment results from the previous section.

C. Performance results

Fig. 9 shows the Cumulative Distribution Function (CDF) of RTT observed during the tests. For comparison, each plot presents all three curves at once. Dotted curve refers to the baseline case, where there is no attack. Dashed curve refers to the system under malicious nodes (or misconfigured equipments) trying to use the control plane as a way to impact the data plane. Solid curve refers to the proposed solution.

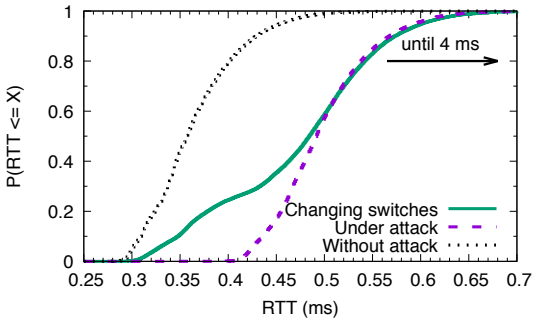

Fig. 9: RTT cumulative distribution function.

Fig. 9 shows that the system without attacks presents lower RTT values. Hence, the number of POX controller operations on switch does not affect the data Plane. Differently, a high rate of rule removal imposes a significant increase on RTT. While the baseline scenario presents 50% of measured RTT values lower than 0.34 ms, the scenario under attack presents 50% of the observed RTT values higher than 0.47 ms. In this scenario, there is no RTT values lower than 0.40 ms.

The mitigation solution indicates that the suspicious flow monitoring enhances the data plane performance. The mitigation solution enhances RTT in almost 5%, when compared to the scenario under attack. More than 30% of the observed RTT values to a system using the mitigation solution is lower than the minimum RTT observed in a system under attack without any countermeasure.

V. Related Work

In general, performance evaluation in SDN considers the control and data planes separately. Several studies attempt to understand how fast hardware and software switches are able to forward data from one port to another [3], [5], or frameworks for switch testing [12]. The performance evaluations of the control plane focus on how many new flows per second a given controller is able to handle [13]; the cost of control plane virtualization [14]; and how it scales [15].

In terms of security, an attacker may attempt to slow down the network by creating fake flows, i.e., generating a large amount of messages to the control plane [16]. In this case, one may employ techniques to identify such DDoS attacks [17]. The attacks on the control plane may also take place from stolen administrative credentials, as introduced by Matsumoto, Hitz and Perrig [1] and called a Malicious Administrator problem. They also proposes Fleet, a system to store k configurations, one for each administrator. Hence, the control plane can change configurations to handle problems in one of the k available configurations. Their work, however, did not detail how to identify a misconfiguration.

Flowspace virtualization –when the flow space is divided into many sections for a particular virtual network– can be used to protect networks that have multiple administrators [18]. FlowVisor is a virtualization system for OpenFlow switches [19]. It is a middleware that is executed in the

controller, allowing the creation of virtual switches over the physical infrastructure. There are other virtualization alternatives, such as OpenVirteX [20]. However, some of the switch resources, as CPU and memory, are shared among virtual networks [19]. Due to that, some hypervisors limit the rate of OpenFlow messages to avoid excessive CPU usage [21].

Medhi, Khalid and Khayan proposed NETAD. It presents four different anomaly detection algorithms for SDN to filter or control the packets as firewall [22]. TRW-CB (Threshold Random Walk with Credit Based Rate Limiting) uses sequential hypothesis tests (i.e., flow tests), analyzing each connection state to detect an attack. Rate-Limiting checks the amount of request in time intervals to detect malicious events. Maximum Entropy Detector uses an entropy calculator to classify packets into categories such as benign or abnormal [23]. Other works propose the use of backup paths to the case in which the main path fails [24], [25]. These works, however, are limited to anomalies in the data plane coming from user flows only.

Differently from related works, the main contribution of this work is to quantify this effect, and to provide a technique to identify malicious administrators, so that trustworthy ones are not throttled.

VI. Conclusions

This work investigated and quantified the limitation of OpenFlow in decoupling the data plane from the control planes. Particularly, this paper answered the two following questions: (i) what is the potential impact of a misconfigured or mis-intentioned control program on the data plane? and (ii) how to increase the resilience of the system to reduce this impact? The performance evaluation of popular hardware and software switches under different types of flow workloads and under the rule insertion, removal and changes led us to conclude that rule removal has the greatest impact on the network, followed by rule updates and insertions. The impact of rule removal and changes is similar to hardware and software switches, whereas rule insertions yield a minor impact on the network performance. Most importantly, the performance impacts come from the lack of a flow computation model in OpenFlow, and because of that are likely to occur in future versions of the specification, regardless of the controller being used. Hence, we proposed a mitigation mechanism that shows improvements in network performance in scenarios under attacks or without attack. As future work, we intend to use deep packet inspection and automatic classification of flows and controller action, to enhance the actual mitigation mechanism.

Acknowledgments

This research is partially funded by Capes, CNPq, and FAPEMIG.

References

[1] S. Matsumoto, S. Hitz, and A. Perrig, "Fleet: Defending sdns from malicious administrators," in *Proc. of the ACM HotSDN*, 2014.

[2] D. Kreutz, F. M. V. Ramos, P. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, p. 63, 2015.

[3] L. C. Costa, A. B. Vieira, E. d. B. e. Silva, D. F. Macedo, G. Gomes, L. H. A. Correia, and L. F. M. Vieira, "Performance evaluation of openflow data planes," in *Proc. of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017.

[4] M. Appelman and M. de Boer, "Performance analysis of openflow hardware," *University of Amsterdam, Tech. Rep*, pp. 2011–2012, 2012.

[5] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "Openflow switching: Data plane performance," in *Proc. of the IEEE ICC*, 2010.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," in *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, 2008, pp. 69–74.

[7] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," in *ACM SIGCOMM Computer Communication Review*, vol. 44, 2014, pp. 87–98.

[8] S. Sezer, S. Scott-Hayward, P.-K. 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," in *IEEE Communications Magazine*, vol. 51, no. 7, 2013, pp. 36–43.

[9] D. F. Macedo, D. Guedes, L. F. M. Vieira, M. A. M. Vieira, and M. Nogueira, "Programmable networks-from software-defined radio to software-defined networking," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 2, pp. 1102–1125, 2015.

[10] POX, "Pox wiki," July 2017, [Accessed in 25-August-2017]. [Online]. Available: https://openflow.stanford.edu/display/ONL/POX+Wiki

[11] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an open, distributed sdn os," in *Proc. of the ACM HotSDN*, 2014.

[12] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, *OFLOPS: An open framework for OpenFlow switch evaluation*, ser. Lecture Notes in Computer Science, 2012, vol. 7192 LNCS.

[13] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks." *Hot-ICE*, vol. 12, pp. 1–6, 2012.

[14] A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer, "Control plane latency with SDN network hypervisors: The cost of virtualization," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 366–380, 2016.

[15] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Computer Networks*, vol. 112, pp. 279–293, 2017.

[16] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "Sdn security: A survey," in *Proc. of the IEEE SDN For Future Networks and Services*, 2013.

[17] K.-y. Chen, A. R. Junuthula, I. K. Siddhrau, Y. Xu, and H. J. Chao, "SDNShield: Towards more comprehensive defense against DDoS attacks on SDN control plane," in *Proc. of the IEEE CNS*, 2016.

[18] T. Sasaki, A. Perrig, and D. E. Asoni, "Control-plane isolation and recovery for a secure SDN architecture," in *Proc. of the IEEE NetSoft*, 2016.

[19] R. Sherwood, G. Gibb, K.-K. Yap, G. Apenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer." in *Tech. Rep. OPENFLOWTR-2009-1*. OpenFlowSwitch.org, 2009.

[20] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "OpenVirteX: Make your virtual SDNs programmable," in *Proc. of the ACM HotSDN*, 2014.

[21] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 655–685, Firstquarter 2016.

[22] S. A. Mehdi, J. Khalid, and S. A. Khayam, *Revisiting Traffic Anomaly Detection Using Software Defined Networking*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2 2011, pp. 161–180.

[23] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: From concept to implementation," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2181–2206, Fourth quarter 2014.

[24] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Enabling fast failure recovery in openflow networks," in *Proc. of the International Workshop on the Design of Reliable Communication Networks (DRCN)*, 2011.

[25] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, and P. Skoldstrom, "Scalable fault management for openflow," in *Proc. of the IEEE ICC*, 2012.