FlowMon: Detecting Malicious Switches in Software-Defined Networks

Andrzej Kamisiński
Department of Telecommunications
AGH University of Science and Technology
Kraków, Poland
kamisinski@kt.agh.edu.pl

Carol Fung
Department of Computer Science
Virginia Commonwealth University
Richmond, Virginia, USA
cfung@vcu.edu

ABSTRACT

Software-Defined Networking (SDN) introduces a new communication network management paradigm and has gained much attention recently. In SDN, a network controller overlooks and manages the entire network by configuring routing mechanisms for underlying switches. The switches report their status to the controller periodically, such as port statistics and flow statistics, according to their communication protocol. However, switches may contain vulnerabilities that can be exploited by attackers. A compromised switch may not only lose its normal functionality, but it may also maliciously paralyze the network by creating network congestions or packet loss. Therefore, it is important for the system to be able to detect and isolate malicious switches. In this work, we investigate a methodology for an SDN controller to detect compromised switches through real-time analysis of the periodically collected reports. Two types of malicious behavior of compromised switches are investigated: packet dropping and packet swapping. We proposed two anomaly detection algorithms to detect packet droppers and packet swappers. Our simulation results show that our proposed methods can effectively detect packet droppers and swappers. To the best of our knowledge, our work is the first to address malicious switches detection using statistics reports in SDN.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—Security and protection (e.g., firewalls); C.2.3 [**Computer-Communication Networks**]: Network Operations—Network monitoring

General Terms

Algorithms, Performance, Security

SafeConfig'15, October 12, 2015, Denver, Colorado, USA.

DOI: http://dx.doi.org/10.1145/2809826.2809833.

Keywords

Anomaly Detection; Malicious Behavior; Network Security; Software-Defined Networking; SDN; OpenFlow

1. INTRODUCTION

Software-Defined Networking (SDN) introduces a new communication network management paradigm and has gained much attention from academia and industry due to its fast growing potential in the market. One primary goal of SDN is to allow a network controller, representing the *control plane*, to overlook and manage the entire network by configuring routing mechanisms for underlying switches. The switches, also called the *data plane*, are solely responsible for data forwarding according to their forwarding tables. Routing computation and network management are handled by the controller. The forwarding table entries on switches are determined and managed by the controller through a secured communication protocol between switches and the controller.

More specifically, most enterprises adopt OpenFlow [11] as communication protocol for secure and efficient communication between switches and the controller. As specified in the OpenFlow protocol [15], switches must forward a request to the controller upon receiving a packet of a new flow. The controller receives the request and computes a routing path for the new flow and notifies the corresponding switches to update their forwarding tables, which is an indication of the establishment of a new flow path. In order to compute the optimal routing paths, the controller needs to know the network topology and its status, such as the present workload on all switches and links. To help the controller with optimal routing computation, switches periodically report the network status to the controller, including the status of the switches and flows. For example, the *port statistics* specify the volume of traffic going through each switch port, and *flow statistics* report the volume of each flow passing through each switch. The controller can specify the frequency of the reports from a switch.

However, switches may contain vulnerabilities that can be exploited by attackers. Vulnerabilities can be manufactured on purpose or may result from programmers' errors and bad coding practices. Once a vulnerability is exploited by an adversary, the switch is said to be compromised. A compromised switch may not only lose its normal functionality, but it may also be manipulated to be malicious. For example, the switch can drop packets passing through it, or direct packets to a wrong port. If the majority of switches in a network are compromised, they may be able to collude

to bring down the entire network and make it difficult to diagnose by providing false information to the controller.

In order to lower down the risk of entire network failure, an SDN network may include switches from multiple different manufacturers. This way the probability of having a large number of switches compromised is decreased. However, the probability of having a few switches compromised is increased. Therefore, it is important for the system to be able to detect and isolate the compromised switches. How to detect compromised switches utilizing the current OpenFlow protocol is the focus of this paper.

To address the above problem, we propose FlowMon[1], a malicious switches monitoring and detection system using the OpenFlow protocol. In this solution, the controller analyzes the collected port statistics and the actual forwarding paths to detect malicious switches in a network. More specifically, two types of abnormal behavior are investigated. One is called *packet dropper*, where the switch purposefully drops packets passing through it. The other is called *packet swapper*, where packets are forwarded to a different port than it was intended. We propose the algorithms for controllers to detect the two malicious behaviors of switches based on the collected reports and the observed forwarding paths. We evaluate our proposed methods in a simulated SDN environment implemented in ns-3.

Several study cases on simulation have shown that controllers equipped with FlowMon can effectively detect packet droppers and packet swappers in almost all cases. To the best of our knowledge, this paper is the first proposed solution to utilize the OpenFlow protocol to detect malicious switches in SDN.

2. RELATED WORK

SDN is currently attracting significant attention from both academia and industry. A group of network operators, service providers, and vendors have recently created the Open Networking Foundation [1], an industrial-driven organization, to promote SDN and standardize the OpenFlow protocol [11]. There have also been standardization efforts on SDN at the IETF, IRTF, and other standards producing organizations.

Security issues of SDN have also been a popular topic in recent years. [10] indicates that there are many attack vectors to an SDN network, involving all layers of the SDN architecture. Switches in adversary mode [13, 16] can not only launch DoS[2] attacks against the controller, they can also be uncooperative in routing flow packets. It is therefore critical for the controller to be able to monitor and diagnose uncooperative or malicious switches. Many SDN performance monitoring solutions have been proposed in the past few years [3, 4]. However, none of them has provided a mechanism to monitor uncooperative switches.

The compromised switches can be a highly likely-to-happen event in SDN networks. Several papers [6, 12] have mentioned that diversity of a network is a desired strategy to increase the robustness of the network. It is recommended that an SDN network employ switches from multiple manufacturers and companies. However, it also increases the probability that one or few switches in the network are compromised and attack the rest of the network infrastructure.

[1]FlowMon — Flow Monitor
[2]Denial of Service

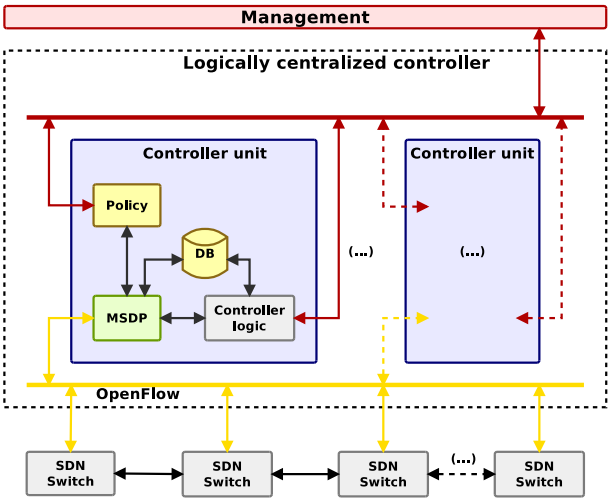

Figure 1: The proposed SDN architecture including additional detection and prevention mechanisms.

The fault diagnosis problems in the engineering domain are commonly solved by machine learning or inference methods [8]. The machine learning approach [2, 7] relies on sufficient training data that the system is operated under normal condition. However, their solution is not designed for SDN network and their required input for analysis cannot be easily obtained in SDN context. Several proposals regarding fault detection and localization for communication networks have been presented [9, 14]. However, those solutions are based on distributed networks where their assumption that sensors are deployed to detect abnormality is not applicable to SDN networks.

The work that is the closest to ours is from Du et. al [5]. In their paper, they use a simple threshold to decide if a switch is malicious or not based on the flow volume through the switch. However, they are not able to detect other types of malicious switches such as packet droppers and packet swappers specified in this paper.

In our work, we focus on SDN-specific environment, where the controller is able to collect and analyze periodic statistics reports from switches regarding the current status of traffic through the OpenFlow protocol. Our work utilizes the existing information from statistics reports and detects anomalous switches based on the collected data.

3. THE PROPOSED ARCHITECTURE OF AN SDN

In Software-Defined Networks, particular network devices are managed by a central entity and cooperate within a hierarchical structure, according to their roles. While the concept of SDN offers high flexibility in terms of the overall management tasks, it also raises new challenges related to the security of network infrastructure. Therefore, additional protection measures should be introduced that will improve the ability of an SDN to defend itself against different kinds of attacks carried out by compromised internal devices.

Figure 1 shows the general network architecture that we propose in our solution. The considered model of an SDN consists of three main groups of network devices. The first group contains all SDN switches, the second group repre-

sents the logically centralized controller which consists of one or more interconnected controller units, and finally, the third group contains all management devices. The standard functionality of controller units is extended to provide a protection layer against malicious switches. In the following sections, we discuss the roles of each device and functional block.

3.1 SDN switches

Switches in SDN networks forward user traffic from the source nodes to the corresponding destination nodes. Whenever a new traffic flow is detected at a switch, the switch receives routing instructions from the SDN controller and then redirects packets of this flow to the appropriate network interface. Communication with the controller occurs on a separate management interface with the aid of the OpenFlow protocol. The switch maintains certain statistics related to the handled traffic and may provide the selected information to the controller on its request. As switches are directly exposed to attacks launched either by users or external adversaries, it is possible that some switches in the network may be compromised following a successful attack. In such a scenario, the compromised devices might be used to destabilize the network, or even take control of a controller if its implementation contains known vulnerabilities. Thus, protection of switches is crucial with regard to the security of the core SDN infrastructure.

3.2 Logically centralized controller

Logically centralized controller is either one or a set of independent, interconnected controller units that supervise the operation of switches and make routing decisions based on the configured policy. Each controller unit maintains complete information about the network topology, which allows it to react to failures effectively and compute optimal paths for traffic flows. As shown in Figure 1, it contains a functional block responsible for making decisions (Controller logic), and may contain a database as a place to store all the necessary information that may influence future actions (statistics, counters, logs). The proposed solution introduces another two functional blocks. The first one is called Malicious Switches Detection and Prevention (MSDP) and is responsible for a continuous, transparent analysis of the communication between the controller and switches via the OpenFlow protocol, as well as detection of any sign of malicious behavior observed in this communication. To determine whether certain actions should be classified as malicious, MSDP refers to the second block (Policy) which contains a set of rules. The rules are configured by network administrators and may be either explicit or generic. As an example, if a switch pretends not to support statistics reports required by the controller, a rule might be created that will isolate this switch from the network until its configuration is verified by an administrator. Then, the administrator may either tune the configuration or restore the original firmware of the switch.

3.3 Management devices

In Software-Defined Networks, management interfaces are needed mainly to configure policies, monitor network performance, set up devices and verify contracts. Access to such interfaces is restricted to network administrators. Consequently, it may be granted to a limited group of machines in the management team. In this paper, all such machines are referred to as Management devices.

In the next section we will discuss the major algorithms in the MSDP component that are used to detect malicious switches.

4. ALGORITHMS TO DETECT MALICIOUS SWITCHES

In an SDN, many reasons can cause the under-performance of the network. For example, a compromised or malfunctioning switch may drop packets or misroute packets. If the performance degradation is due to malicious switches, then isolating those switches may improve the performance of the network. However, the primary challenge is to identify which switch causes the performance problem. In this paper, we propose two solutions that include data collection, analysis, and decision assistance to reduce the impact of malicious behavior in SDN. The proposed algorithms may be included in the MSDP module shown in Figure 1, with the corresponding threshold values defined in the form of rules in the Policy block.

4.1 Detection of packet droppers

Our anomaly detection strategy for packet droppers includes the following two components:

- **link anomaly detection** — indicates that the reports show abnormal data loss or data gain on a specific link;
- **switch anomaly detection** — indicates that the reports show abnormal data volume entering and leaving a specific switch.

According to the OpenFlow Switch Specification (version 1.4.0), the port statistics that are returned by switch devices in response to the *OFPMP_PORT_STATS* request sent by the controller contain two fields, *tx_packets* and *rx_packets*, which provide the total number of successfully transmitted and received packets on each port. Therefore, the controller can request switches to send the reports with the same intervals. The controller can then detect anomalies through examining port statistics from all switches.

4.1.1 Packet dropping in switches

When a malicious switch drops packets and does not provide false information in statistics reports, then the controller can infer abnormal switch behaviors from statistics reports. For example, in the port statistics report, the number of received packets and the number of transmitted packets from all ports should be approximately equal. Other indicators, such as the number of receiving/transmitting errors and the receiving/transmitting drop rate, can also reflect the health status of a switch.

We can use a simple threshold-based formula to detect abnormality of behavior of switch k:

$$\left| \frac{\sum_{\forall i \in P_k} T_k^i - \sum_{\forall i \in P_k} R_k^i}{\sum_{\forall i \in P_k} T_k^i + \sum_{\forall i \in P_k} R_k^i} \right| > \theta \quad ? \tag{1}$$

where T_k^i is the number of transmitted packets from switch k port i in the last time window, and R_k^i is the number of received packets from switch k port i in the last time window. P_k is the set of all ports on switch k. If the total packet dropping rate is higher than the threshold θ, then the switch is perceived as malicious or malfunctioning.

4.1.2 Packet dropping on links

One type of behavior of malicious switches is to drop packets it should forward. The lost packets can be reflected by missing packets on links. To detect abnormality on data transmission through links, the controller can analyze port statistics reports by comparing the number of transmitted and received packets between adjacent switches in each time window. If some packets are dropped on a link (e.g., at the destination network interface), then the number of transmitted packets and the number of received packets reported by switches at the two ends of the link will be different.

We can assume that during the normal network operation the packet dropping rate does not exceed a certain percentage of the total number of exchanged packets. We propose to use the following test for each pair of connected switches on any selected link, S_i and S_j, respectively:

$$|T_{ij} + T_{ji} - R_{ij} - R_{ji}| > \alpha\,|T_{ij} + T_{ji}| \quad ? \qquad (2)$$

where T_{ij} is the number of transmitted packets from switch S_i to S_j in the last time window, R_{ij} is the number of received packets by switch S_i from S_j in the last time window, and α is the chosen threshold value. The left side of the test denotes the difference between the total number of transmitted packets and total number of received packets from both ends during the time window. The right side of the test denotes the total transmitted packets from both ends during the time window. If Condition (2) is true, then the related packet dropping is perceived as abnormal.

4.1.3 Packet dropper detection algorithm

Packet dropping can be detected with the aid of two tests defined in Sections 4.1.1-4.1.2. In this section, we propose an algorithm (Algorithm 1) that can be used to detect packet droppers based on information collected from port statistics from switches.

Once abnormal packet dropping is detected on a switch or a link during the last time window, the corresponding anomaly scores are updated. If a switch is an endpoint of multiple links with abnormal packet dropping, then the anomaly score of such a switch will be higher. If the condition in Line 5 is true, then the abnormal packet dropping inside the switch is detected and its anomaly score is increased. Further, if the condition in Line 12 is true, then abnormal packet dropping is detected and the anomaly scores of both link endpoints are increased. In this case, it is possible that only one of the adjacent switches is dropping packets.

The α and θ parameters in Algorithm 1 should be set to values that provide a reasonable trade-off between false positives and false negatives, considering the usual differences in the reported numbers of transmitted and received packets in case of a trusted network (i.e., a network which is known not to contain compromised devices at the time of the measurements). Note that the time window may also be modified accordingly. Note that the computation of anomaly scores for all switches in Algorithm 1 is triggered after every reporting time. The computation complexity for each time cycle is $O(V+E)$, where V is the total number of switches and E is the total number of connected pairs of switches. To reduce computation overhead on the controller side, reducing the statistics reports frequency can be a feasible solution.

Algorithm 1: Computing anomaly scores of all switches based on port statistics

Input : $\mathbb{S}$ denotes the set of all switches to be monitored and $\mathbb{L}$ denotes the set of links to be monitored; B_{ij} denotes the number of bytes sent from switch S_i to switch S_j during the last time window; T_{ij} denotes the number of packets sent from switch S_i to switch S_j during the last time window; R_{ij} denotes the number of received packets by switch S_j from switch S_i during the last time window; T_i^j denotes the number of packets sent for transmission by switch S_i on port j; R_i^j denotes the number of packets received by switch S_i on port j; α, θ are the anomaly thresholds to identify anomalies; β is the discounting factor; C_{ij} is the capacity of the link between switch S_i and S_j (in bits per second); TW is the length of the fixed time window (in seconds).

Output: Set H containing packet dropping scores of all switches.

1 Initialize set H to be all 0s for all switches;
2 **Event e:** to be triggered after every reporting time cycle TW.
3 $H = \beta H$; `// discount scores after each cycle.`
4 **foreach** k ***s.t.*** $S_k \in \mathbb{S}$ **do**
5 **if** $\left|\frac{\sum_{\forall i \in P_k} T_k^i - \sum_{\forall i \in P_k} R_k^i}{\sum_{\forall i \in P_k} T_k^i + \sum_{\forall i \in P_k} R_k^i}\right| > \theta$ **then**
6 $H_k = H_k + 1$
7 **end**
8 **end**
9 **foreach** pair of connected switches $(S_i, S_j) \in \mathbb{L}$ **do**
10 $\Delta = |T_{ij} + T_{ji} - R_{ij} - R_{ji}|$
11 $\Sigma = |T_{ij} + T_{ji}|$
12 **if** $\Delta > \alpha\Sigma$ *and* $\frac{8 \cdot B_{ij}}{TW} \leq C_{ij}$ *and* $\frac{8 \cdot B_{ji}}{TW} \leq C_{ji}$ **then**
13 $H_i = H_i + 1$
14 $H_j = H_j + 1$
15 **end**
16 **end**
17 return H;

4.2 Detection of packet swappers

A malicious switch can also be a packet swapper, i.e., forward packets to wrong destinations. In this section, we propose an algorithm to detect such a behavior.

The principle of the algorithm is described as follows: we investigate the reports regarding each unknown flow i and compare the indices of the expected output network interface (according to the routing table R) with the actually observed network interface. The function $GetPrecedingSwitchIndex$ is to find the preceding switch index m where packet p_i^k was directed to the switch S_k. This can be found by inspecting the port index of the packet that arrives at switch k and find the corresponding switch connected to that port. $GetExpectedOutputInterface$ and $GetRealOutputInterface$ are the corresponding functions to find the expected output port index and the actual output port index of the packet. If the indices are different, it means that switch S_m forwarded the flow via a wrong network interface. Therefore, switch S_m is likely to be a packet swapper and its binary decision variable is set to 1.

Algorithm 2: Detecting packet swappers in an SDN

Input : $\mathbb{S}$ denotes the set of all switches to be monitored; T denotes the network topology; R denotes the current flow routing table maintained by the controller; p_i^k denotes the first packet of an unknown flow i received at switch $S_k \in \mathbb{S}$.

Output: Set W containing binary decision variables for all switches in $\mathbb{S}$.

Initialize set W to be all 0s for all switches;

Event e: to be triggered whenever a switch reports a packet representing an unknown flow to the controller.

$m \leftarrow$ GetPrecedingSwitchIndex(p_i^k)

$n_{\text{out,exp}}$ = GetExpectedOutputInterface(R, S_m, i)

$n_{\text{out,real}}$ = GetRealOutputInterface(T, S_m, S_k, p_i^k)

if $n_{\text{out,exp}} \neq n_{\text{out,real}}$ **then**

  $W_m = 1$

end

return W;

5. EVALUATION

To verify the performance of the proposed solutions, we have implemented them in the widely-used ns-3 simulator, and then we conducted a simulation-based evaluation for an example SDN network (Figure 2). We set up a network with the following features:

- the network contains 10 SDN-enabled switches;
- each switch is directly connected to 2 client devices (e.g., personal computers or servers) which may generate and receive network traffic;
- there are always 3 malicious switches in the network and they are selected at random;
- for packet droppers: the probability that a packet is dropped is constant and equal to 0.25, whereas $\beta = 0.6$;
- packet swappers forward all received packets of a given flow via the network interface which has the highest index, and at the same time, which is not the source or the expected destination interface for this flow;
- in each experiment, 10 UDP CBR[3] flows are set up between pairs of client devices (the source and destination nodes are selected at random, and the bit rate of each flow equals 80 kbit/s);
- each simulation case is repeated 10 times (based on that, we compute the average values and 95% confidence intervals using the Student's t-distribution).

The effectiveness of the proposed algorithms was assessed based on the following two metrics:

- **detection rate** — the ratio of the number of detected malicious switches to the number of all malicious switches in the network;
- **false alarm rate** — the ratio of the number of switches misclassified as malicious to the number of all non-malicious switches in the network.

[3]Constant Bit Rate

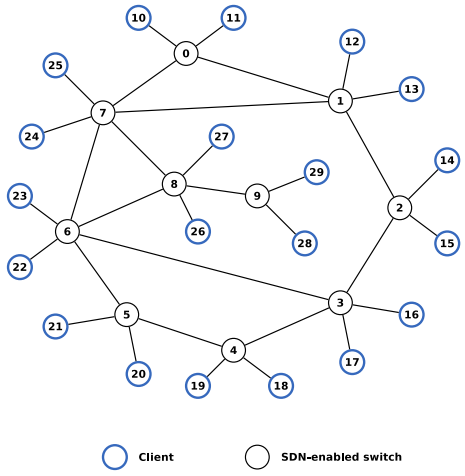

Figure 2: Network topology used in the evaluation.

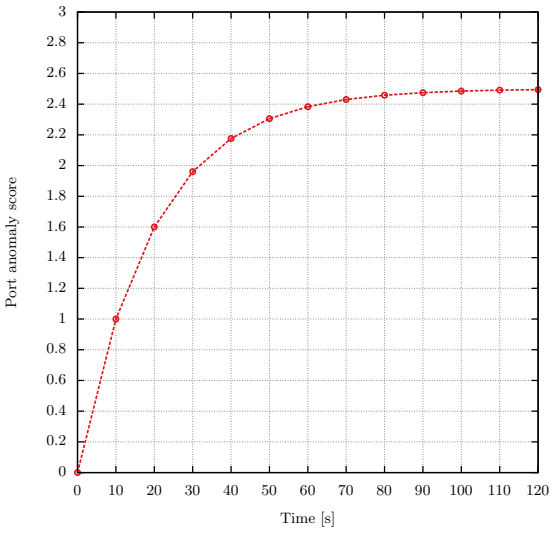

Figure 3: An example relation between the port anomaly score and time for a packet dropper.

We considered three different simulation scenarios with respect to the type of malicious behavior:

- all malicious switches in the network are packet droppers;
- all malicious switches in the network are packet swappers;
- the type of malicious behavior is selected at random (either packet dropper or packet swapper).

In addition, each scenario consisted of 12 different cases in terms of the set of values for the following parameters: α, β, and θ (see Table 1). The results for each of the scenarios are discussed in the following sections.

5.1 Scenario A — three packet droppers

In the first scenario, three different switches in the network were selected at random as packet droppers. It was

Table 1: Simulation cases within each scenario

Case no	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII
α	0.01	0.01	0.01	0.010	0.10	0.10	0.10	0.10	0.30	0.30	0.30	0.30
β	0.60	0.60	0.60	0.60	0.60	0.60	0.60	0.60	0.60	0.60	0.60	0.60
θ	0.01	0.05	0.10	0.50	0.01	0.05	0.10	0.50	0.01	0.05	0.10	0.50

expected that they will be detected as such by any of the proposed algorithms. The results are summarized in Table 2, while Figure 3 shows an example relation between the port anomaly score and time for a real packet dropper.

Based on the results, the port-related detection method achieved 100% accuracy in all cases except IV, VIII, and XII, in which the θ parameter was assigned the highest value of 0.5 (it means that the difference between the number of transmitted and received packets on all ports of a switch in the last time window has never exceeded 50% of the sum of all transmitted and received packets in this time window — see Algorithm 1). The selection of α and θ values leads to the trade-off between the sensitivity of the mechanism and the corresponding false alarm rate (some packets may be dropped due to transmission errors, for instance). At the same time, the flow-related detection method was confirmed to be ineffective with respect to the detection of packet droppers.

5.2 Scenario B — three packet swappers

The second scenario is similar to the previous one, but this time all malicious switches are packet swappers. The results for this case are shown in Table 3.

As expected, the flow-related detection method performed better than the port-related one. However, note that the detection rate is 77% in all considered cases[4], which suggests that some malicious switches remained undetected. The reason for the observed decrease in performance is that some packets have been forwarded by malicious switches to client devices, while client nodes are not aware of SDN and they are not even connected to the SDN controller. Thus, they do not provide any statistics to the controller and by definition are not analyzed by our algorithms.

5.3 Scenario C — three malicious switches of both types

The last scenario combines both types of malicious behavior — three switches, as well as types of their abnormal behavior, were selected at random. It was expected that the proposed solutions might be used together to detect the compromised switches in the network. Table 4 shows the evaluation results.

According to the results, both algorithms have demonstrated their ability to detect malicious switches in a mixed environment (i.e., containing packet droppers and packet swappers). It is worth to note that no false alarms were observed, which proves that the proposed solutions are accurate. However, the detection rates reflect the previous conclusions regarding the limited applicability of each method (the algorithms detected distinct malicious switches, according to the related type of malicious behavior). Thus, when combined together, the algorithms can detect malicious switches more effectively.

[4]The flow-related method does not depend on α, β, and θ.

6. CONCLUSION AND FUTURE WORK

In this paper, we investigate the problem of malicious switches in SDN networks and propose a malicious switches monitoring and detection system that uses information contained in the OpenFlow protocol messages to detect abnormal behavior of SDN-enabled switches. Two algorithms are presented that recognize packet droppers and packet swappers in a network. The simulation results have confirmed that each of the two algorithms is able to detect one of the considered types of abnormal behavior, depending on the selected threshold values. The proposed concept may be developed further to detect other signs of malicious behavior, such as an unauthorized modification of the TTL value included in the IP header.

However, we realize that there are some limitations to our proposed solution. In our future work, we would like to address the following problems: (1) what is the impact from dishonest switches which may provide false information in their statistics reports? (2) what if the compromised switches collude together? (3) what are the other types of malicious behavior of compromised switches and what are the corresponding solutions? In addition, we also plan to use a real SDN testbed to evaluate our proposed solutions.

7. REFERENCES

[1] The Open Networking Foundation. https://www.opennetworking.org/about.

[2] M. Aminian and F. Aminian. Neural-network based analog-circuit fault diagnosis using wavelet transform as preprocessor. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 47(2):151–156, Feb 2000.

[3] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. PayLess: A Low Cost Network Monitoring Framework for Software Defined Networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9, May 2014.

[4] Cisco. Introduction to Cisco IOS NetFlow, 2012. http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.pdf (last accessed: July 4, 2015).

[5] X. Du, M.-Z. Wang, X. Zhang, and L. Zhu. Traffic-based Malicious Switch Detection in SDN. *International Journal of Security and Its Applications*, 8(5):119–130, 2014.

[6] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro. Analysis of operating system diversity for intrusion tolerance. *Software: Practice and Experience*, 44(6):735–770, 2014.

[7] R. Isermann. Supervision, fault-detection and fault-diagnosis methods — An introduction. *Control Engineering Practice*, 5(5):639–652, 1997.

Table 2: Evaluation results for Scenario A

Case no	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII
Port-related detection method												
Detection rate	1	1	1	0	1	1	1	0	1	1	1	0
Error	0	0	0	0	0	0	0	0	0	0	0	0
False alarm rate	0	0	0	0	0	0	0	0	0	0	0	0
Error	0	0	0	0	0	0	0	0	0	0	0	0
Flow-related detection method												
Detection rate	0 ± 0											
False alarm rate	0 ± 0											

Table 3: Evaluation results for Scenario B

Case no	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII
Port-related detection method												
Detection rate	0.00 ± 0.00											
False alarm rate	0.00 ± 0.00											
Flow-related detection method												
Detection rate	0.77 ± 0.16											
False alarm rate	0.00 ± 0.00											

Table 4: Evaluation results for Scenario C

Case no	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII
Port-related detection method												
Detection rate	0.60	0.60	0.57	0.00	0.60	0.60	0.57	0.00	0.60	0.60	0.57	0.00
Error	0.19	0.19	0.20	0.00	0.19	0.19	0.20	0.00	0.19	0.19	0.20	0.00
False alarm rate	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Error	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Flow-related detection method												
Detection rate	0.30 ± 0.18											
False alarm rate	0.00 ± 0.00											

[8] R. Isermann. *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. Springer Berlin Heidelberg, 2006.

[9] I. Katzela and M. Schwartz. Schemes for fault identification in communication networks. *Networking, IEEE/ACM Transactions on*, 3(6):753–764, Dec 1995.

[10] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards Secure and Dependable Software-defined Networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 55–60, New York, NY, USA, 2013. ACM.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[12] S. Neti, A. Somayaji, and M. E. Locasto. Software Diversity: Security, Entropy and Game Theory. In *7th USENIX Workshop on Hot Topics in Security*, Berkeley, CA, 2012. USENIX.

[13] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-Defined Networks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 413–424, New York, NY, USA, 2013. ACM.

[14] M. Steinder and A. S. Sethi. Probabilistic fault localization in communication systems using belief networks. *Networking, IEEE/ACM Transactions on*, 12(5):809–822, Oct 2004.

[15] The Open Networking Foundation. OpenFlow Switch Specification, 2013. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf (last accessed: July 4, 2015).

[16] L. Wei and C. Fung. FlowRanger: A Request Prioritizing Algorithm for Controller DoS Attacks in Software Defined Networks. In *IEEE International Conference on Communications (ICC 2015)*. IEEE, 2015.