International Journal of Future Generation Communication and Networking
Vol. 11, No. 5 (2018), pp.11-26
http://dx.doi.org/10.14257/ijfgcn.2018.11.5.02

# Isolating Malicious Controller(s) In Distributed Software-Defined Networks with Centralized Reputation Management

Bilal Karim Mughal[1], Sufian Hameed[2] and Bilal Hameed[3]

*[1,3]Bahria University, Pakistan*
*[2]IT Security Labs, NUCES, Pakistan*
*[1]bilalkmughal@gmail.com, [2]sufian.hameed@nu.edu.pk,*
*[3]bilalhameed.bukc@bahria.edu.pk*

***Abstract***

*Although software-defined networks have seen a sharp increase in their deployment around the world, with big tech companies including Microsoft and Google, to name a few, tapping into the enormous potential that these networks offer, there are still various security loopholes that need to be plugged. One such security-related issues is that of a rogue controller bringing down an entire network. As we shall see in this paper, this problem is still short of any definitive solutions, especially when it comes to distributed software-defined networks. We attempt to resolve this issue by developing a centrally managed trust and reputation scheme. By proactively comparing the policies/flow rules that need to be installed in the switches with those that are actually installed, our scheme singles out a malicious controller. We have evaluated the scheme for scalability, message overhead, and for bad-mouthing attacks. Our results suggest that using trust and reputation system can greatly enhance the network security in this scenario as demonstrated by rigorous evaluations in Emulab network emulation testbed.*

***Keywords:*** *Rogue Controller, Software Defined Networking, Software Defined Security, SDN*

## 1. Introduction

Network traffic volumes have soared in the last decade by an enormous amount because of the diversity of applications for which the networks are used today. Faster, always-on, and on-the-go network access are primary requirements of today's network users. With the advent of bandwidth-intensive applications such as big data and video-on-demand, it becomes imperative that the data handling capacity of routers be increased, so that they can route the packets to their destinations faster. This is not possible with traditional network devices because they are short of the flexibility which is required to handle different types of packets because routing rules are hardwired in them [13].

Conventional networks are dependent on standalone hardware components for different network functions, which include but are not limited to managing networks data flow, performing routing functions, *etc.*, [12]. When a routing device receives a packet in a conventional network, it determines the path to its destination device by using the routing algorithm configured by a network administrator. In traditional networks, the routing plane and the forwarding plane are housed in a single device [13]. These routing devices usually make use of proprietary, closed-source algorithms [9], which leave little room for a network administrator to configure the network beyond the liberty given to him by the vendor.

Apart from this, in a traditional network, each network needs to be programmed individually which is time-consuming and becomes a tedious task if the network is too

Received (April 25, 2018), Review Result (June 28, 2018), Accepted (July 21, 2018)

ISSN: 2233-7857 IJFGCN
Copyright © 2018 SERSC Australia

large. In case of a security breach, a policy change, or simply a change in network architecture, the traditional network devices cannot be reprogrammed easily [4].

This poses a limit on the performance of traditional networks and is an impediment to the ever-increasing requirement of scalability, resilience, and above all, security. A solution to this problem is separation of control (routing) plane and data (forwarding) plane, and making the control plane fully programmable. This approach grants network administrators with more control over the network behavior because now they can dynamically reprogram their network keeping in view the situation.

As will be seen, Software-Defined Networking (SDN) is a technology that provides this framework and has seen an enormous increase in its usage in the past few years [1]. SDNs are aimed at providing increased flexibility and control over network functions, and they achieve this by separating control and data plane from each other in routers and switches [8]. Separating the control plane from data plane enables the controlling entity to have a global overview of the entire network, in which the control plane can pass on commands to data plane, to all devices at once [7, 32].

In SDN, a network administrator has to implement the policy only in the controller, which is then replicated across the network's data forwarding devices [10]. Although the terminologies involved do imply that SDN is a physically centralized network architecture, it need not be so. In reality, large-scale software-defined networks make use of multiple controllers for managing the network, which appears to subordinate network devices as a single entity, or at least not in conflict with each other. This simplifies the network administration enhances network resilience and reliability, and also, (logically) centralized point of administration ensures that all network devices have a consistent, up-to-date state of the network. Despite immense popularity and ever-increasing growth in deployment of SDNs, much research is needed as to whether SDN can be deployed on a large scale, that too with all the security [24]. Various researchers including [16, 17] acknowledge control layer as a highly vulnerable section of the SDN, which, if compromised, can result in losing the entire network to the malicious entity.

This is a crucial security vulnerability keeping in view the ever-increasing growth of SDNs, and their deployment in large-scale, enterprise and corporate networks, where distributed SDN architecture is used. However, the models proposed for deploying SDNs so far do not answer one basic question: how to identify malicious or rogue controllers within a network, and how to prevent them from causing damage [23].

This paper proposes a scheme to enhance controller security in a distributed environment. The proposed framework identifies malicious/rogue controllers by finding out if a mismatch exists between the flows which should be installed in the switches by the controllers and those which are actually installed.

The proposed framework achieves this by making use of a centralized trust and reputation scheme inspired by Personalized Trust Model (PET Model) [18], in which controllers are rated positive or negative by other controllers according to their performance. The results are then reported to a central entity called the Trust Collector which aggregates the results and passes them on to the network administrator. Earliest detection of rogue controllers through such reputation management will ensure the isolation of rogue controllers before they can damage the network.

The scheme of collecting ratings and aggregating trust and reputation using a Trust Collector component works more robustly and accurately than delegating trust and reputation management entirely to an individual controller in a distributed environment. This is because a central entity is always needed that can aggregate the ratings generated by all the controllers that are part of the distributed environment, and the central entity can then make a decision of whether a given controller is malicious by looking at what the majority of ratings say about that controller.

Alternatively, the central entity can also output the result of the ratings to a human operator who can decide whether a given controller is malicious based on both their

 Copyright © 2018 SERSC Australia

domain knowledge about the network and also based on the majority of ratings that were received for that controller.

A fully functional prototype was developed and was tested by deploying on Emulab (http://emulab.net/) network evaluation testbed, powered by the University of Utah for the evaluation of this scheme. As such, the results, as will be seen later in this paper, prove that the scheme serves its purpose and successfully singles out malicious controllers which have installed policies other than those asked by the network administrator to do so.

The scheme was evaluated on the following three parameters.

1. Scalability: Up to 15 controllers with 15 switches and 3 malicious controllers were deployed to test the time taken by the scheme to complete one trust rating. As the number of controller increases, the time taken also increases linearly, but not exponentially. Results are reported in Figure 4.
2. Message overhead: Number of messages exchanged during one complete cycle of trust ratings. Up to 10 controllers with 20 switches and 3 malicious controllers were deployed to evaluate the overhead of the scheme. Results are reported in Figure 5.
3. Bad mouthing attacks: The scheme was tested for identifying bad-mouthing attacks. Up to 6 controllers with 12 switches and 4 malicious controllers were deployed and tested for such attacks. The results are reported in Table. 3.

To restate, the contributions include the following:

– Developed a framework for singling out a malicious controller in distributed SDN.

– Deployed the framework in Emulab network emulator to prove that the framework is successful in serving its purpose.

An early version of this work appeared as a short paper in [19]. The extended version contains extensive evaluations with bigger and more complex configurations. The configurations used for this extended version are an insight into the scheme's usability for production networks. As discussed in 'Discussions' section, the maximum number of controllers used in the topologies is 15, which may be too large for some networks. Therefore, testing against a large number of controllers provides us a fair evaluation of how the scheme would perform in environments smaller than those simulated for this paper.

The rest of the paper is organized as follows. Section 2 discusses security problems in the SDN, along with the need for this paper and our scheme. Section 3 presents an overview of our scheme architecture and the components introduced along with their working. Section 4 is where we present the evaluation results. Section 5 discusses different points related to this scheme and finally conclude the paper in Section 6.

## 2. Security Threats to SDNs

There are several vulnerabilities that are easy doorways for hackers who want to get into the network. The attacks can be on resources, and as well as on the assets. For example, a hacker can compromise a switches flow table such that the traffic for a particular destination is forwarded instead of being dropped. Such kind of attack will have major consequences on network bandwidth and can also be used to launch a denial-of-service attack. Similarly, an attacker can make SDN controllers a target. This is a lucrative option because controllers in SDN are centralized entities, and hijacking them or bringing them down can be disastrous for any organization. The communication channel between controllers and switches in SDNs is a major vulnerability, which if targeted, can impart an attacker with absolute control over the network. Similarly, the high-level

interfaces through which a controller communicates with the applications can be attacked which may lead to rogue application gaining control over the sensitive network information.

An SDN can come under attack through several attack types. An attacker can compromise the data plane, manipulate the flow rules, and divert the traffic to another unintended destination or simply towards a black hole route. In another kind of attack, the attacker can take advantage of vulnerabilities in the data plane devices.

However, the most devastating attack types are the ones which are on the control plane, because of the fact that the control plane is the central point of the SDN; bringing down an SDN controller means total hijacking of the network, and consequently exploiting it to launch attacks, steal information, or simply shut down the network as part of a DoS attack.

If this kind of attack takes place in a corporate data-center, financial institution, government database, then it can not only cause loss of billions of dollar but also compromise sensitive information which cannot be afforded. Securing the control plane is a necessary element of SDN deployment, however, as will be seen later in this document, this is the area which has several security loopholes, and there is no definitive security arrangement that tackles this problem. Apart from this, vulnerabilities in nodes that are connected to the SDN controller, for example, a network administrator's workstation, can enable an adversary to manipulate the network in an easy manner.

The SDNs also suffer from several security and dependability problems which include but are not limited to repudiation, spoofing, information disclosure, tampering, elevation of privileges, and denial of service [5]. Researchers have found out that several kinds of attacks can be launched against network based on OpenFlow [14]. These include stealing information during the flow installation. Fingerprinting [26] is a specialized kind of attack which can be launched as a precursor to the DoS on SDN. For launching such an attack, an attacker keeps a check on the time delay between the first packet of a flow and installation of flow by the SDN controller.

Various researches have highlighted numerous security concerns which arise when one talks about deploying SDN on a large-scale. These include but are not limited to deficiency of documentation guiding developers to enable security. Moreover, TLS deployment is made optional and to date, there are many controller and switch platforms which do not implement TLS [30]. Similarly, those platforms which have TLS implemented do not provide any other inherent transport security mechanisms. There are some switches who by default are in listening mode thereby can enable a vicious TCP connection to form a connection [5]. It has already been mentioned that SDNs are centralized networks, and by virtue of this centralization, DoS attacks can wreak a havoc which can bring an entire network down in an instant [28]. For example, some researchers have demonstrated that a single vulnerable application can be exploited by an attacker to take over the control plane by launching a DoS attack and making the resources unavailable for legit users [26, 5].

Similarly, there have been demonstrations that various popular controller platforms including POX, Floodlight, Beacon, and OpenDaylight have their own vulnerabilities due to which they cannot be deemed as completely secure and dependent controllers [27]. These controllers can crash as well because of bugs in applications that bloat the memory. Other than being a victim of DDoS attacks, recent advents in SDN also bring us new approaches to deal with DDoS attacks in a collaborative manner. SDN controllers lying in different autonomous systems (AS) can securely communicate and transfer attack information with each other. A third party detection engine (similar to HADEC [35]) can feed the attack information into the destination network, which then forwards them to the neighboring network and this process continues until the definitions reach the source, thus saving valuable time and network resources [33, 34].

 Copyright © 2018 SERSC Australia

In SDN paradigm, attention to the overall security of network architecture has not been pondered upon, rather, due to its modular nature, each layer of SDN has its own vulnerabilities, and hence its specific security requirements [2]. An in-depth study of security solutions proposed so far reveals that the scientific community is still short of a definite SDN architecture which is completely secure and makes use of a standardized set of security solutions and practices for increasing its reliability and dependency. The frameworks proposed so far have targeted specific security problems such as FRESCO [25] which facilitates the development of security applications for OpenFlow Networks. The authors in [28] have proposed a framework for verifying whether any flow policies are in conflict with security policies. Similar to these problems and their solutions, the authors in [3] identify a bucket-load of security vulnerabilities across the SDN architecture from top to bottom and discuss the solutions. Since this paper is specifically related to controller security in SDN, therefore, only the security issues in control layer will be discussed, and then the focus will be shifted to specific problems that were tackled through the proposed scheme.

There are several studies which have tried to resolve controller security problems in SDN. For example, the Security Enhanced Floodlight (SE-Floodlight) controller provides a mechanism for authentication of applications, role-based authorization for avoiding conflicts in flow-rule insertion, and conflict detection and resolution [22]. It does not, however, address one core problem, that is, isolating a compromised controller in a distributed environment. SDNs are logically centralized networks in which a single controller maintains multiple switches and other network devices, but in case of a man-made or technical mishap, this proves to be a single point of failure too [20].

To overcome this, distributed architectures like DISCO [21] have been proposed in which multiple controllers manage the network for better resilience and faster network management. Some network architectures such as HyperFlow [29] and Onix [15] distribute the control plane physically but keep it logically centralized. The distributed systems described above, however, do not take into account the security aspects. For example, they do not provide a comprehensive framework for identifying and isolating a malicious controller out of several others. On the other hand, so-far proposed schemes for securing the control layer do not discuss the feasibility of their solutions in the distributed environments. To the best of our knowledge, no concrete work has been done to resolve this problem, and therefore this is an open challenge for research.

## 3. Centralize Reputation Management Architecture for SDN

The objective of this work is to develop a framework for singling out a malicious controller in distributed SDN. It was achieved by employing a trust and reputation scheme among controllers. The scenario was that of a distributed controller environment in which the secondary controllers are deployed not as a dormant backup but as active load-balancers. However, for either use case, the controllers need to have access to all switches, so that in case one controller goes down due to an act of sabotage or for any other reason, the other controllers can prevent disruptions in the network environment.

### 3.1. Components

In this architecture, controllers rate each other after verifying the policies installed by them in switches against the policies that are dictated by a central entity called the Policy Distributor. The Policy Distributor is a component introduced by us for consistent policy enforcement throughout the distributed SDN. The second component specific to this scheme is the Trust Collector, which asks controllers to rate their peer controllers and takes ratings from them. The code for both the components was written in Python and they were deployed as separate components. The working of individual components is described below.

**3.1.1. Third-order Headings:** It contains all of the policies that are to be installed by the controllers. Conventionally, a network administrator defines the policies directly into the controller, but in this scheme, a network administrator defines the policies in the Policy Distributor. These policies are then periodically pushed to all of the controllers in the network. This ensures network-wide consistency as there is only one place where the policies need to be defined, thereby centralizing the administration of a distributed SDN. A HashMap was used for policy assignments which take arguments (Controller, Policy). Copy of this HashMap can be retrieved by all controllers when needed, but every controller installs it in only the switches directly under its control.

This helps them later in verifying whether other controllers have installed correct policies or not, and is also good for fault tolerance; in case a controller goes down, other controllers will automatically know which flow rules were in effect in the affected controller. It is assumed that the Policy Distributor is secure and protected from hijacking, and any changes made to it are purely intentional.

**3.1.2. Trust Collector:** It is another central entity which is responsible for trust management. After the Policy Distributor has pushed the policies to the controllers, the Trust Collector, after a specific time, asks all the controllers of the network for their opinion about their peer controllers. Specifically, it asks other controllers to check whether their peer controllers have installed the policies in switches as dictated by the Policy Distributor, or they have (maliciously) installed different policies. The controllers then initiate their respective Policy Checkers (discussed in next section) and fetch the flow tables from the switches. If a controller finds any discrepancy between the flow tables fetched from switches and the policies sent by the Policy Distributor, it reports the results to the Trust Collector. We use the flow tuple format to specify and compare policies, *e.g.*,

$$policy1 : srcIP = 8.8.8.8, action = drop. \quad (1)$$

**3.1.3. Policy Checker:** Another simple component called the Policy Checker was introduced by integrating it into the Ryu SDN Framework (https://osrg.github.io/ryu/) controller. The primary purpose of Policy Checker is to simply probe the switches to fetch the installed policies so that they can be compared with the policies sent out by the Policy Distributor.

## 3.2. Trust Collection

The mechanism of trust collection in this scheme is based on Personalized Trust Model (PET Model) [49], however, necessary changes were made to their method to suit the environment that was in focus. The PET model is designed for strict P2P environments where there is no central entity, and the nodes are dependent on ratings obtained from each other to calculate trustworthiness. In this scheme, however, a central entity called the Trust Collector collects individually calculated trustworthiness values from all controllers and presents it to the operator for review.

When the Trust Collector asks controllers to find out any mismatch between policies installed and policies that had to be installed, the controllers start probing the switches. At this point, all the controllers simultaneously act as recommender and recommendee. A recommender who finds out a mismatch flags the recommendee based on the following function.

$$h(x) = \begin{cases} S_1, x = G, S_1 > 0 \\ S_2, x = B, S_2 < 0 \quad and \quad |S_2| > S_1 \end{cases}$$

 Copyright © 2018 SERSC Australia

Where G and B are the constants used for match and mismatch, respectively. In case of a match, a score of S1 is output, whereas, in case of a mismatch, S2 is given as output. The rating output by the hash function is then used in calculating the recommendation Er. Note that G is used to represent good behavior, similar to PET model, but B is used to represent bad behavior while the PET model uses it to represent Byzantine behavior. Figure 3 shows the different parameters that go into the calculation of the trustworthiness value. The recommendation value Er for a controller A is the average value of recommendations that other controllers have given to A. Therefore, in order to calculate Er for, let's say, controller A, controller B will need access to recommendations that other peers have given to A.

The Trust Collector helps here by allowing all controllers to send their calculated recommendations about other controllers to itself. Once all recommendations are at the Trust Collector, each controller can then retrieve the (global) accumulation of all recommendations about any given controller from the Trust Collector.

The second thing the controllers need to calculate is the interaction-derived information Ir. In the PET model, Ir is a special recommendation given by a peer A to other peers based on how much good or bad service those other peers have provided to peer A, that is, unlike Er, Ir does not take into account the recommendations from other peers.

The controller environment in this paper is slightly different from the pure P2P environment assumed by the PET model, since in this environment no controllers directly provide any services to other controllers as in a P2P system, so the meaning of Ir was changed, such that Ir is now each controller's individual recommendation about its peer controllers based on whether they have installed policies in the switches correctly or not. Thus Ir is an individual controller's own opinion about a given controller A and it does not take into account what other controllers say about A. This saves Ir from getting overwhelmed if a majority of controllers (maliciously) rate controller A as negative. The Er and Ir values are finally used to calculate the reputation Re in a weighted fashion such that,

$$W(Er) = 0.2 \text{ and } W(Ir) = 0.8 \quad (2)$$

The values are based on suggestions from the PET model. A higher value for W(Er) would mean that a lot of trust is put in the environment but since the environment had to be considered risky for being realistic, a very high value was not set. The purpose of reputation Re is to accumulate the past and current values of a controller's performance. That is, the reputation value is the historical accumulation for a recommendee's past behavior from the recommender's viewpoint. It will reflect the overall quality of the recommendee for a long time period.

For example, if a controller which is being rated has installed 99 correct policies but 1 incorrect policy due to, let's say, a software bug, then it shouldn't mean that the controller's reputation immediately becomes completely negative. Rather, the final reputation value is calculated through a combination of current and past recommendations from both individual and a collective group of controllers. Since Ir gives us the personalized view of a node for its peers, therefore the PET model uses only Ir to calculate the risk value Ri for the network. This results in each controller having its own Ri value that represents its own view of the risk in the network.

The reputation Re and risk Ri values are used by the controllers to calculate trustworthiness T values. Each controller thus generates one trustworthiness value that gets collected by the Trust Collector. On PET model's suggestions, the weight of reputation and weight of risk was set to 0.5 in all controllers for calculating the T value such that:

$$W(Re) = 0.5 \text{ and } W(Ri) = 0.5 \quad (3)$$

Copyright © 2018 SERSC Australia

The Trust Collector accumulates all these trust values it receives from controllers. It then averages all the trustworthiness values and notifies the network administrator as to which controllers are malicious since their trustworthiness value was very low or which controllers are good since their trustworthiness value was high.

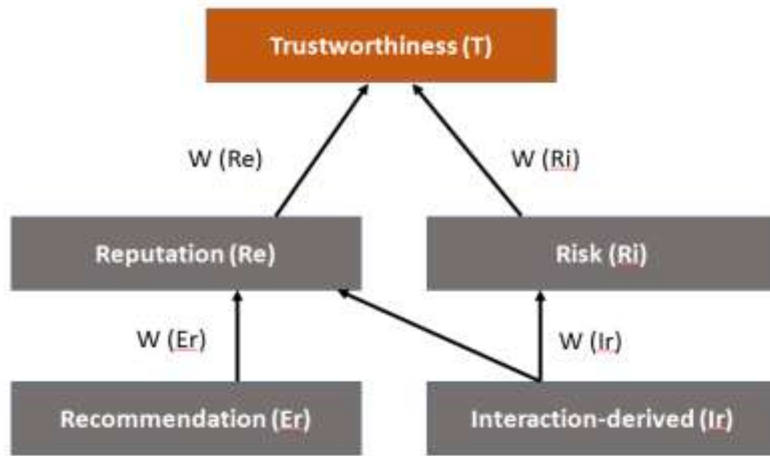


**Figure 1. Trust Calculation Model. Nodes Collect Final Trustworthiness Value Based Upon a Number of Factors**

### 3.3. Scheme Overview

In presence of the Policy Distributor, Trust Collector, and Policy Checker integrated within the controllers, this scheme progresses as follows:

A network has three controllers and three switches, such that each controller directly administers two switches. The network is in a full mesh setting so that all the controllers have access to all switches for backup. Assuming that the network has just booted, and the switches do not have any flow rules as of now. A network administrator defines a policy in the Policy Distributor that all traffic originating from IP address 8.8.8.8 is to be dropped.

After some time, the Trust Collector asks controllers to probe all the switches to find out if there is a mismatch between installed policies and those dictated by the Policy Distributor. The controllers then run their respective Policy Checkers over the network.

As shown in Figure 2, each controller probes switches managed by other controllers too. The Policy Distributor pushes flow rules to the controllers.

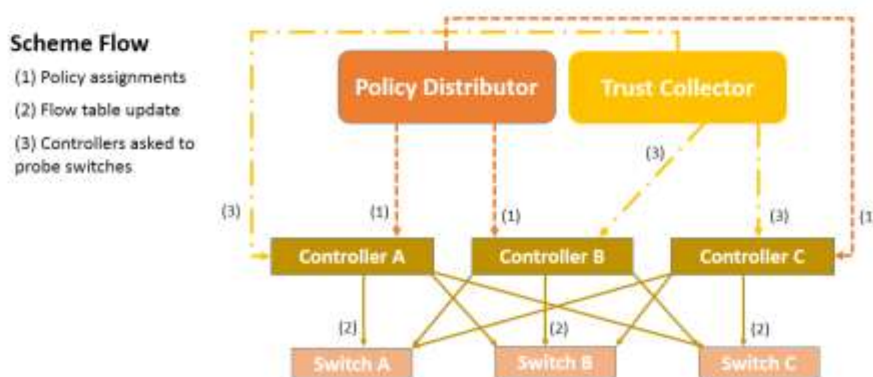


**Figure 2. Flow Diagram for the first Three Steps of the Scheme. Shows Message Exchange flow between Switches, Controllers, Trust Collector, and Policy Distributor**

 Copyright © 2018 SERSC Australia

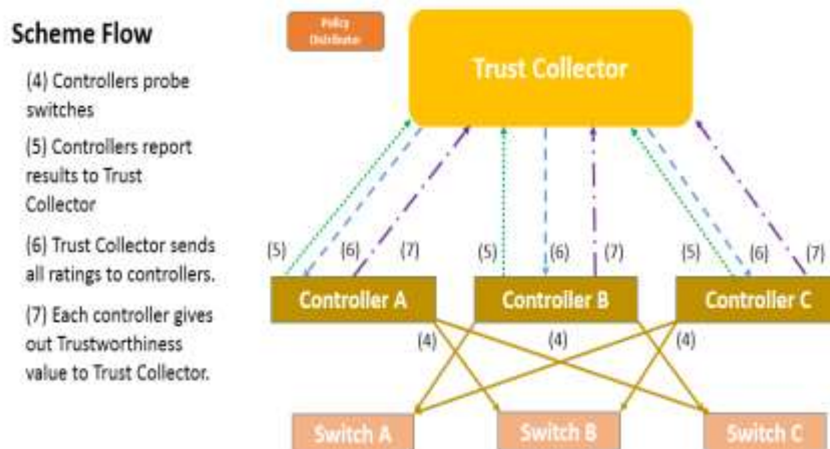


**Figure 3. Flow Diagram for the other Four Steps of the Scheme. Shows Message Exchange Flow between Switches, Controllers, Trust Collector, and Policy Distributor**

Every controller receives all flow-rules, but only the intended controller acts upon it and installs them in its slave switch.

After some time, the Trust Collector instructs the controllers to inspect the switches for installed flow-rules. This is where the benefit of providing every controller with all flow-rules comes in. As seen in Figure 2 and 3, every controller inspects the slave switches of other controllers.

When the probe has finished and matches/mismatches have been found, each controller gives out a rating map for every other controller, which contains good or bad scores for them. Three controllers will generate three such maps, such that in case of three controllers A, B, and C, controller A will report about B and C, controller B will do it for A and C, and controller C will do it for A and B. All of these ratings are sent to the Trust Collector.

Once the Trust Collector has received the reports from all of the controllers, it combines all of them and sends back to all of the controllers, so that A will receive reports of B and C about each other, B will receive reports of A and C, and C will receive reports of A and B. Each of the controllers now has information about what its peer controllers think about other controllers. This information helps a controller in calculating the average value of recommendation (Er) for other controllers.

Controllers then output final trust value about other controllers to the Trust Collector. Er combined with Ir are used to calculate reputation Re. Combined with risk Ri, the Re is used to calculate final trustworthiness as:

$$T = Reputation(Re) * WeightofReputation[W(Re)] + Risk(Ri) * WeightofRisk[W(Ri)] \quad (4)$$

Each controller outputs trustworthiness values for other controllers. The results are fed to the Trust Collector, which aggregates the results from all the controllers and shows it to the network administrator for review.

## 4. Evaluations

A prototype implementation was developed in Python for Trust Collector, Policy Distributor, Policy Checker, and rating mechanism of the controllers. The Policy Checker and rating mechanism were integrated into Ryu controller, whereas the Policy Distributor and Trust Collector were deployed as separate modules. A small number of controllers and OpenFlow switches were also deployed in the Emulab network evaluation testbed.

Copyright © 2018 SERSC Australia

In the topology, one Policy Distributor, one Trust Collector, and a varying number of controllers and switches were deployed for different evaluations. For the scalability tests, simulated switches were used, and the number of controllers was increased to up to 15, and the number of switches to up to 20. The results of scalability evaluation that calculates the time taken for the trust collection process is shown in Table 1. The corresponding results for the time taken by the trust collection process for each of the configurations from Table 1 are shown in Figure 9. The graph is labeled from C1, ..., C12 which correspond to configuration numbers from Table 1.

For correctness evaluations, which were performed alongside the scalability evaluations mentioned above, one or more of the controllers were deliberately triggered to randomly install a malicious policy and then ran the rating mechanism in the controllers. All other controllers were able to detect the controller which installed the wrong policy and rated it negatively. The Trust Collector aggregated the ratings from all these controllers. For all the tests conducted, the scheme was always able to find the malicious controller(s) with zero false positives or false negatives.

Figure 9 shows the time taken to perform the entire process of rating and trust collection as the number of controllers involved in the process is increased. As seen from the graph, the time shows a linear pattern of increase and our scheme is able to work fast in finding out the malicious controller.

The scheme defines a specific number of message exchanges (as shown in earlier sections) between the different components in the system, *i.e.*, the controllers, switches, Trust Collector, and Policy Distributor. A centralized graph database, Neo4j [31], which serves as a 'noticeboard' for communication using the publish-subscribe mechanism, was used. This saves network bandwidth since the Policy Distributor or Trust Collector do not have to broadcast messages containing commands such as 'startTrustCalculation', a command meant to be sent to all controllers to start the trust calculation process, to all controllers. Instead, the Trust Collector can publish this command by writing it in the centralized database and the controllers can read it from there. Thus only one message exchange has to be used instead of a broadcast of messages to all controllers.

**Table 1. Different Network Configurations Created of Controllers and Switches for Scalability Evaluation of Time Taken. In each Configuration, Number of Switches Controlled by One Controller is Equal to (Number of switches / Number of controllers)**

| Configuration | Controllers | Switches | Malicious controllers |
|---|---|---|---|
| Config 1 | 4 | 4 | 1 |
| Config 2 | 5 | 5 | 1 |
| Config 3 | 6 | 6 | 1 |
| Config 4 | 7 | 7 | 1 |
| Config 5 | 8 | 8 | 2 |
| Config 6 | 9 | 9 | 2 |
| Config 7 | 10 | 10 | 2 |
| Config 8 | 11 | 11 | 2 |
| Config 9 | 12 | 12 | 3 |
| Config 10 | 13 | 13 | 3 |
| Config 11 | 14 | 14 | 3 |
| Config 12 | 15 | 15 | 3 |

 Copyright © 2018 SERSC Australia

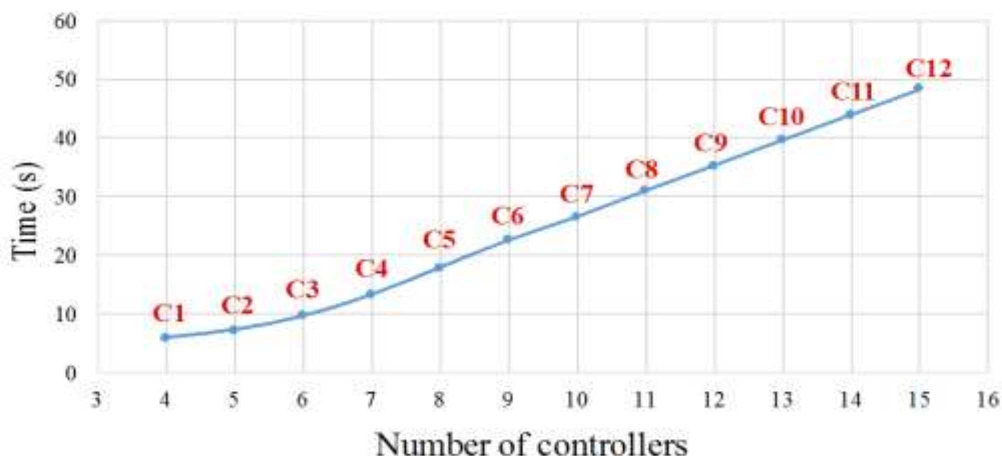


**Figure 4. Scalability of the Scheme: Shows the Time Taken in Seconds for Entire Rating and Trust Collection Scheme to Fnish as the Number of Controller is Increased. C1, .. , C12 refer to Config1, , Config12 from Table 1**

Neo4j has a Python library that handles the lower level network communication code and provides a RESTful web API which is invoked from the code to perform publish or subscribe functions. Note that each node in evaluation setup has an IP address, this includes the node running Neo4j, and so the REST API can be invoked on the Neo4j database from any of the controllers and Trust Collector or Policy Distributor components by using the IP of the Neo4j node. The number of messages that need to be used for one complete process of trust calculation is:

$$No.ofmessages = O(N * M) \quad (5)$$

Where N is the number of controllers involved in calculating the trust and M is the number of switches in the network, and there exists one instance of the Policy Distributor component and one instance of the Trust Collector component. While the Neo4j database based communication scheme described earlier helps get rid of broadcast messages, each controller (from N number of controllers) has to communicate with each of the switches (from the M number of total switches).

Table 2 shows the various configurations of controllers and switches which were created in the evaluation setup for scalability evaluations for the number of messages used for the entire trust calculation and collection process. Note that this set of configurations created are different than those created for the earlier evaluation and were shown in Table 1.

Figure 10 shows the number of messages that were used to perform one complete process of trust collections for the various configurations mentioned in Table 2. Note that here a message from a component A to component B is defined as one write of a message from a component A and its corresponding read by a component B. As can be seen from the graph, the scheme scales smoothly as the number of controllers and switches involved in the process is increased. For the highest configuration, Config 10, with 10 controllers and 20 switches, the scheme uses less than 250 messages to finish the entire process of trust calculation. Note that in each configuration, the switches are equally divided between the controllers. That is,

$$No.ofswitchescontrolledbyacontroller = No.ofswitches/No.ofcontrollers \quad (6)$$

Finally, Table 3 shows the different network configurations of controllers and switches for doing bad mouthing evaluations to show that the trust-based scheme provides defense against bad mouthing attacks [6]. Bad mouthing is a technique where a malicious party (a

Copyright © 2018 SERSC Australia 

malicious controller in our case) provides dishonest recommendations for one or more other parties (controllers in our case) to malign them.

The scheme guarantees to catch bad mouthing attacks as long as the number of bad or malicious controllers in the network is less than n/2 where n is the number of total controllers in the network. If the number of malicious controllers is exactly equal to n/2 then the scheme can output an unsure result which can be shown to a human operator for final decision making. For example, if n=4, we have 4 controllers C1, C2, C3, and C4 and lets say that there are 2 malicious controllers C1 and C2 in the network, and lets say both C1 and C2 rate C3 as a bad controller but C3 and C4 rate C3 as a good controller then our scheme will output an unsure result since the votes for C3 are equally divided between good and bad. In such a scenario, a human operator will look at the results and will have to make the final decision based on his expertise and knowledge about the network.

**Table 2. Different Network Configurations Created of Controllers and Switches for Scalability Evaluation of Number of Messages. In each Configuration, Number of Switches Controlled by One Controller is Equal to (Number of switches / Number of controllers)**

| Configuration | Controllers | Switches | Malicious Controllers |
|---|---|---|---|
| Config 1 | 1 | 2 | 1 |
| Config 2 | 2 | 4 | 1 |
| Config 3 | 3 | 6 | 1 |
| Config 4 | 4 | 8 | 2 |
| Config 5 | 5 | 10 | 2 |
| Config 6 | 6 | 12 | 2 |
| Config 7 | 7 | 14 | 3 |
| Config 8 | 8 | 16 | 3 |
| Config 9 | 9 | 18 | 3 |
| Config 10 | 10 | 20 | 3 |

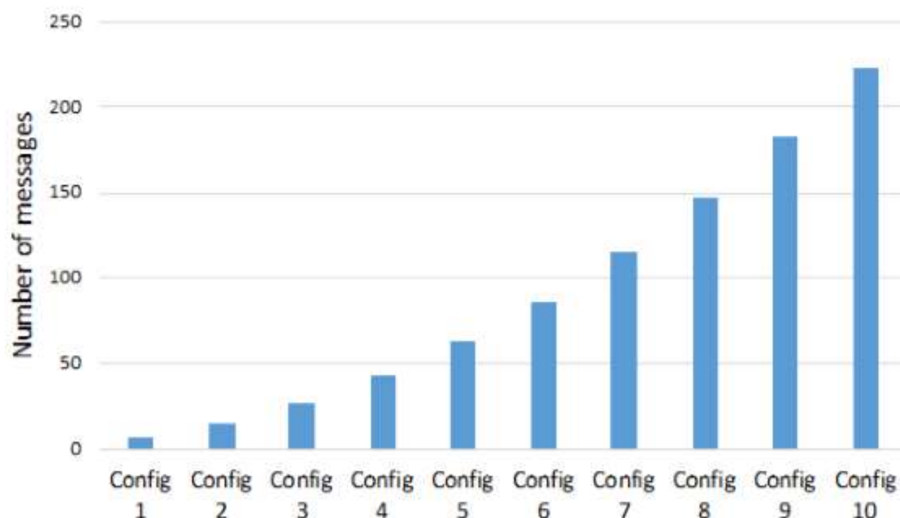


**Figure 5. Shows the Number of Message Exchanges that Take Place for Different Network Configurations of Controllers and Switches, the Names of the Configurations on the X-axis (e.g. Config1, Config2, , Config10) Refer to the Configurations in Table 2.**

 Copyright © 2018 SERSC Australia

**Table 3. Evaluations for Bad Mouthing Attacks. In each Configuration, Number of Switches Controlled by one Controller is Equal to (No. Of switches / No. Of controllers). And CX Refers to the Controller Number, e.g. C1, C2, etc. The Result Column Shows the final Result after the Trust Calculation Round which Aggregates Trust Values from all Controllers in the Network**

| Configuration | Controllers | Switches | Malicious Controllers | Bad Mouthing | Result |
|---|---|---|---|---|---|
| Config 1 | 2 | 4 | 0 | None | All found trusted |
| Config 2 | 3 | 6 | 1 | C1 bad mouthed C2 | C2 found trusted |
| Config 3 | 3 | 6 | 1 | C2 bad mouthed C3 | C3 found trusted |
| Config 4 | 3 | 6 | 2 | C1, C2 bad mouthed C3 | C3 found untrusted |
| Config 5 | 3 | 6 | 2 | C1, C3 bad mouthed C2 | C2 found untrusted |
| Config 6 | 4 | 8 | 1 | C1 bad mouthed C2 and C3 | C2 and C3 found trusted |
| Config 7 | 4 | 8 | 2 | C1, C2 bad mouthed C3 | C3s status found unsure (need human operator to make decision since 50% votes good, 50% bad) |
| Config 8 | 4 | 8 | 3 | C1, C2, C3 bad mouthed C4 | C4 found untrusted |
| Config 9 | 6 | 12 | 1 | C1 bad mouthed C6 | C6 found trusted |
| Config 10 | 6 | 12 | 1 | C2 bad mouthed C1, C3, C4, C5, C6 | C1, C3, C4, C5, C6 all found trusted |
| Config 11 | 6 | 12 | 2 | C1, C2 bad mouthed C3 and C4 | C3 and C4 found trusted |
| Config 12 | 6 | 12 | 4 | C1, C2, C3, C4 bad mouthed C3 and C4 | C3 and C4 found untrusted |

## 5. Discussion

The test results presented earlier show that the scheme works correctly and efficiently in weeding out malicious controllers. Since the Trust Collector decides whether a controller is malicious based on an aggregate of recommendations from all other controllers, therefore the scheme provides defense against bad mouthing attacks [6].

In bad mouthing attacks, a malicious party provides dishonest recommendations for another good party to malign the name of the good party. But since the scheme does not make a decision of whether a controller is malicious based on the recommendation from just one other controller, therefore it can provide defense against bad mouthing as long as malicious controllers are not the majority in the total number of deployed controllers. This assumption is reasonable since the number of controllers which would need to become malicious before the network collapses can be guaranteed.

That is, in a network with N controllers, the scheme is guaranteed to work correctly and identify malicious controllers as long as (N/2)+1 controllers are uncompromised. This assumption is realistic since a majority of controllers is unlikely to become malicious in an instant and if they become malicious one by one over time, then the scheme will identify the malicious controllers at all times when (N/2)+1 controllers are still uncompromised.

This scheme of collecting ratings and aggregating trust and reputation using a Trust Collector component works more robustly and accurately than delegating trust and reputation management entirely to an individual controller in a distributed environment. This is because a central entity is always needed to aggregate the ratings generated by all the controllers that are part of the distributed environment, and the central entity can then make a decision of whether a given controller is malicious by looking at what the majority of ratings say about that controller.

Alternatively, the central entity can also output the result of the ratings to a human operator who can decide whether a given controller is malicious based on both his domain knowledge about the network and also based on the majority of ratings that were received for that controller.

Introducing a central trust managing entity also helps in solving an important dilemma, which is, what happens if a majority of rogue controllers vote against a controller which otherwise has installed correct policies? Let us examine a case of distributed trust

Copyright © 2018 SERSC Australia 

management, in which the controllers find the policy mismatches themselves, and there is no central entity for managing the trust and reputation. There are three controllers in a network, A, B, and C, each managing one switch under them, and connected to other switches too. A network administrator defines one flow rule, i.e. block any traffic originating from IP address 200.0.0.1. The controllers install the flow rules in their respective switches. After some time, controllers probe the switches to find out whether other controllers installed correct policies in their respective switches. A finds out that B and C have (maliciously) installed flow rules in their switches which allow traffic originating from 200.0.0.1. It rates B and C negative. B and C on the other hand rate A as negative. In presence of an automated solution of shutting down or restricting a malicious controller, this will prove to be disastrous. If, however, a human operator has to approve the shutting down or restricting of a malicious controller, then it will be a burden for him to sort through the conflicting ratings of controllers against each other. Using the Trust Collector for aggregating the opinions about other controllers from each controller not only helps us in ascertaining the validity of recommendations with surety, but it also helps in eliminating broadcasts.

If, for example, there are three controllers A, B, and C, then all of the nodes will have to send their reports to each other so that they can perform final trust calculation (since the final step in the trust calculation process inside a controller needs input from other controllers too). However, by introducing the Trust Collector in between, all controllers send their reports to this central entity, which simply forwards it to individual controllers.

While it is true that the scheme introduces this one central point of compromise, the Trust Collector, but it is much easier to guard and protect one component if it can help us have a safe distributed environment of controllers where each controller does not have to be guarded very well. As long as the majority of controllers are not compromised, the scheme guarantees that the network will keep functioning correctly. Extensive evaluations conducted by varying the number of controllers and switches ensures that the scheme still holds itself together when different topologies are used. The maximum number of controllers were used during evaluations is 15, which may be too large a number for most network topologies. Researchers have used the Internet2 (http://www.internet2.edu/) OS3E topology for determining the ideal number of controllers, and their placement across the networks. The Internet2 is a research network for collaboration between different universities across the United States and is an SDN comprising of 34 nodes.

Researchers in [26] have demonstrated that adding more controllers to the network does increase resilience and provides defense against complete network failures, however, proper placement of controllers within the network is an ongoing research problem. In case a network controller goes down, the switches should connect to next possible controller with minimum latency, therefore placing the controllers in such a fashion that they provide minimum latency to all switches is desirable, but not always possible [11].

Authors in [26] have selected up to 5 controllers and authors in [11] have performed their evaluations with up to 8 controllers and found promising results (with some tradeoffs, of course). This means that the choice of using up to 15 controllers is not in any way insufficient in demonstrating the feasibility of the scheme.

## 6. Conclusion

Securing the controllers in SDN is an open problem for research. Researches carried so far do not address the problem of identifying malicious controllers, especially in a distributed environment. In our work, we have focused on this problem and have tackled it by employing a trust and reputation management scheme in which controllers rate each other for the services they provide. This was done by introducing a centralized entity which keeps a record of policies to be installed so that controllers can compare them with installed policies for rating purpose.


Copyright © 2018 SERSC Australia

The scheme was coded in Python, and implemented into Ryu controller as a prototype and demonstrated promising results. Emulab network emulation testbed was used for evaluations. In the evaluations, the number of controllers, switches, and malicious controllers was varied to reflect different network topologies. Metrics that were evaluated included the number of message exchanges that take place for different network configurations of controllers and switches, time taken in seconds for our entire rating and trust collection scheme to finish as the number of controllers is increased, scalability evaluation, and finally, the scheme was tested against bad mouthing attacks common in trust management systems.

Overall it was found out that the scheme works well in isolating a malicious controller by probing the switches for any rogue flow rules thus completing the objective of this work.

## References

[1] S. Agarwal, M. Kodialam and T. V. Lakshman, "Traffic Engineering in Software Defined Networks", In INFOCOM, 2013 Proceedings IEEE, **(2013)**, pp. 2211-2219.

[2] A. Akhunzada, E. Ahmed, A. Gani, M. K. Khan, M. Imran and S. Guizani, "Securing Software Defined Networks: Taxonomy, Requirements, and Open Issues", IEEE Communications Magazine, vol. 53, no. 4, **(2015)**, pp. 36-44.

[3] I. Alsmadi and D. Xu, "Security of Software Defined Networks: A Survey", Computers & security, vol. 53, **(2015)**, pp. 79-108.

[4] K. Bakshi, "Considerations for Software Defined Networking (SDN): Approaches and Use Cases", In Aerospace Conference, 2013 IEEE, **(2013)**, pp. 1-9.

[5] K. Benton, L. J. Camp and C. Small, "Openflow Vulnerability Assessment", In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, ACM, **(2013)**, pp. 151-152.

[6] S. Buchegger and J.-Y. Le Boudec, "Coping with False Accusations in Misbehavior Reputation Systems for Mobile Ad-Hoc Networks", Technical report, **(2003)**.

[7] S. Das, G. Parulkar, N. McKeown, P. Singh, D. Getachew and L. Ong, "Packet and Circuit Network Convergence with Openflow", In Optical Fiber Communication Conference, page OTuG1. Optical Society of America, **(2010)**.

[8] S. Fang, Y. Yu, C. H. Foh and K. M. M. Aung, "A Loss-Free Multipathing Solution for Data Center Network using Software-Defined Networking Approach", In APMRC, 2012 Digest, IEEE, **(2012)**, pp. 1-8.

[9] N. Feamster, J. Rexford and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks", ACM SIGCOMM Computer Communication Review, vol. 44, no. 2, **(2014)**, pp. 87-98.

[10] A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt and T. Gayraud, "Software-Defined Networking: Challenges and Research Opportunities for Future Internet", Computer Networks, vol. 75, **(2014)**, pp. 453-471.

[11] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner and P. T.-Gia, "Pareto-optimal Resilient Controller Placement in SDN-Based Core Networks", In Teletraffic Congress (ITC), 2013 25th International, IEEE, **(2013)**, pp. 1-9.

[12] F. Hu, Q. Hao and K Bao, "A Survey on Software-Defined Network and Openflow: From oncept to Implementation", IEEE Communications Surveys & Tutorials, vol. 16, no. 4, **(2014)**, pp. 2181-2206.

[13] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking", IEEE Communications Magazine, vol. 51, no. 2, **(2013)**, pp. 114-119.

[14] R. Kloti, V. Kotronis and P. Smith, "Openflow: A Security Analysis", In Network Protocols (ICNP), 2013 21st IEEE International Conference on IEEE, **(2013)**, pp. 1-6.

[15] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue and T. Hama, "Onix: A Distributed Control Platform for Large-Scale Production Networks", In OSDI, vol. 10, **(2010)**, pp. 1-6.

[16] D. Kreutz, F. 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, ACM, **(2013)**, pp. 55-60.

[17] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey", Proceedings of the IEEE, vol. 103, no. 1, **(2015)**, pp. 14-76.

[18] Z. Liang and W. Shi, "Pet: A Personalized Trust Model with Reputation and Risk Evaluation for P2P Resource Sharing", In System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on IEEE, **(2005)**, pp. 201b-201b.

Copyright © 2018 SERSC Australia 

[19] B. Karim Mughal, S. Hameed and G. Muhammad Shaikh, "A Centralized Reputation Management Scheme for Isolating Malicious Controller(s) in Distributed Software-Defined Networks", International Journal of Advanced Computer Science and Applications (IJACSA), vol. 7, no. 12, **(2016)**.
[20] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks", IEEE Communications Surveys & Tutorials, vol. 16, no. 3, **(2014)**, pp. 1617-1634.
[21] K. Phemius, M. Bouet and J. Leguay, "Disco: Distributed Multi-Domain SDN Controllers", In Network Operations and Management Symposium (NOMS), 2014 IEEE, **(2014)**, pp. 1-4.
[22] P. A. Porras, S. Cheung, M. W. Fong, K. Skinner and V. Yegneswaran, "Securing the Software Defined Network Control Layer", In NDSS, **(2015)**.
[23] A. S. Prasad, D. Koll and X. Fu, "On the Security of Software-Defined Networks", In Software Defined Networks (EWSDN), 2015 Fourth European Workshop on IEEE, **(2015)**, pp. 105-106.
[24] S. Sezer, S. S. 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", IEEE Communications Magazine, vol. 51, no. 7, **(2013)**, pp. 36-43.
[25] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu and M. Tyson, "Fresco: Modular Composable Security Services for Software-Defined Networks", In NDSS Symposium, **(2013)**.
[26] S. Shin and G. Gu, "Attacking Software-Defined Networks: A First Feasibility Study", In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, ACM, **(2013)**, pp. 165-166.
[27] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh and B. B. Kang, "Rosemary: A Robust, Secure, and High-Performance Network Operating System", In Proceedings of the 2014 ACM SIGSAC conference on computer and communications security, ACM, **(2014)**, pp. 78-89.
[28] S. Son, S. Shin, V. Yegneswaran, P. Porras and G. Gu, "Model Checking Invariant Security Properties in Openflow", In Communications (ICC), 2013 IEEE International Conference on IEEE, **(2013)**, pp. 1974-1979.
[29] A. Tootoonchian and Y. Ganjali, "Hyperflow: A Distributed Control Plane for Openflow", In Proceedings of the 2010 internet network management conference on Research on enterprise networking, **(2010)**, pp. 3-3.
[30] M. Wasserman and S. Hartman, "Security Analysis of the Open Networking Foundation (ONF) Openflow Switch Specification", **(2013)**.
[31] J. Webber, "A Programmatic Introduction to NEO4J", In Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity, ACM, **(2012)**, pp. 217-218. ACM.
[32] S. H. Yeganeh, A. Tootoonchian and Y. Ganjali, "On Scalability of Software-Defined Networking", IEEE Communications Magazine, vol. 51, no. 2, **(2013)**, pp. 136-141.
[33] S. Hameed and H. Ahmed Khan, "SDN Based Collaborative Scheme for Mitigation of DDoS Attacks", Future Internet, vol. 10, no. 3, **(2018)**, pp. 23.
[34] S. Hameed and H. Ahmed Khan, "Leveraging SDN for collaborative DDoS mitigation", Networked Systems (NetSys), 2017 International Conference on IEEE, **(2017)**.
[35] Sufian Hameed and Usman Ali. "Efficacy of live ddos detection with Hadoop", Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP, IEEE, **(2016)**.

 Copyright © 2018 SERSC Australia