Game-Theoretic Approach to Malicious Controller Detection in Software Defined Networks

Vignesh Sridharan and Mohan Gurusamy
Department of Electrical and Computer Engineering
National University of Singapore, Singapore
E-mail: E0009097@u.nus.edu, elegm@nus.edu.sg

***Abstract*—Software Defined Networking (SDN) enables programmability and flexibility in networks through the separation of control and data plane. SDN architecture poses new security threats to the network, especially in the control plane. A compromised controller can exhibit malicious behavior such as fraudulent rule installation while avoiding detection. Existing approaches deal with this issue by broadcasting every flow setup request to multiple controllers to check for forwarding rule consistency. This imposes heavy load on the control plane, leading to longer response time and increased cost. We propose a novel approach that can effectively detect a malicious controller without overloading the control plane. The proposed game-theoretic approach randomly selects switches to check for consistency of forwarding rules. We model the problem as a Stackelberg game and solve the corresponding optimization problem to obtain an effective randomization strategy. We also develop a heuristic algorithm to determine a set of actions for the defender to make the problem tractable. We consider load management at the controller and relative importance of switches while finding an optimal strategy for randomized checking. In comparison with three other heuristic approaches, our approach achieves up to 88% improvement in probability of detecting the malicious controller.**

***Index Terms*—Software defined networking, security, game theory, control plane, controller, Stackelberg game**

I. Introduction

Software Defined Networking (SDN) is a promising approach in networking that introduces flexibility and programmability in networks. This is achieved through the separation of control and data plane. The controller residing in the control plane performs critical network functions such as implementing traffic engineering policies, network monitoring and enforcing security policies. The controller achieves this by maintaining a flow table at each switch that contains the forwarding rules for the incoming flows. When a switch receives a new flow for which it does not have a corresponding forwarding rule, the switch sends a flow setup request in the form of a packet-in message to the controller. The controller computes the forwarding rule based on the traffic engineering policy and sends the rule to the switch in a packet-out message. The switch updates its flow table and the new flow is directed through the network. Due to concerns over scalability and resilience, distributed controller architectures have been proposed in SDN [1]. Each controller manages a part of the network and they coordinate with each other to behave as a logically single controller [1] [2] to manage the entire network.

The separation of control and data plane in SDN introduces new security threats to the network. The control plane and the applications that run on the controllers pose serious threat to the network in the event they get compromised [3] [4]. After an attacker compromises a controller, he can have full access to the switches under its purview and can indulge in behavior such as fraudulent rule installation at switches for malicious purposes. A malicious controller can install flow rules that could cause degradation in network performance, divert traffic along sub-optimal routes or through destinations of its choice, and obtain valuable information about traffic flowing through the network and network traffic engineering and security policies. Security measures such as access control have been proposed to prevent unauthorized access to controllers; however these measures are not foolproof and it is possible for a controller to get compromised, due to both internal and external internal security threats, including applications that run on top of the controller [5]. Once it has been compromised, it is possible for it to behave in an intelligent and stealthy manner to prevent detection while engaging in malicious behavior. Thus, the ability to detect the presence of a malicious controller is of paramount importance.

In literature, detection of a malicious controller in a distributed controller architecture is performed by using replication based approaches [5] [6]. In such approaches, each switch sends flow setup requests to multiple controllers simultaneously and checks the consistency of the forwarding rules received. However, such approaches impose heavy load at the controllers and subsequently lead to higher response times to flow setup requests. They also lead to higher costs for the network operator since more number of controllers are required to handle the load at the control plane due to the replication approach. Hence, it is necessary to develop an effective approach that does not incur such high costs and delays at the control plane.

In this paper, we propose a game-theoretic approach towards the detection of a malicious controller in the network. Instead of replicating every single request to multiple controllers to check for rule consistency, requests are replicated randomly for consistency check. This randomization would alleviate the problem of high load at the control plane and as a consequence, lower the network costs. The objective of our work is to develop an effective strategy based on which the randomization is done, taking into account critical aspects such as load management at the controller, relative importance of switches and attack model of the compromised controller. Our contributions are listed as follows:

- We develop a Stackelberg Game based Randomization Strategy (SGRS), with a non-malicious controller in the defender role and malicious controller in the attacker role. The solution to the optimization problem formulated based on the Stackelberg game model provides a probabilistic randomization strategy that allows the defender to minimize the risk of an attack. Stackelberg game is a suitable model for real life security problems as security measures are taken before the attack takes place [7]. The sequential nature of actions is represented well by a Stackelberg game.
- We also develop a heuristic algorithm to obtain the set of actions for the defender, as obtaining a strategy over all the possible actions could be intractable.
- We compare our proposed approach against three other possible approaches in our performance study to demonstrate the effectiveness of our approach.

The rest of the paper is organized as follows. We discuss related works in Section II. We present our proposed scheme and problem formulations in Section III followed by performance evaluation in Section IV. We conclude the paper in Section V.

II. Related Work

The separation of control and data plane in SDN has led to new threats to the network [3] and control plane security has received attention in the research community. Studies have been conducted that focus on prevention of unauthorized access to controllers [3]. However, they do not discuss detection of controllers in the event they get compromised.

A Byzantine resilient approach is described in [5] where each switch sends its flow setup requests to $3f+1$ controllers simultaneously, where f represents the maximum number of faulty controllers that the system can tolerate. The switch decides the rule to be installed based on $3f+1$ forwarding rules that it receives from the controllers to which it had sent the requests. Such an approach requires many controllers to run the network which increases network costs. It also imposes heavy load at the controllers and the switches can experience higher response time since each and every request is broadcast to $3f+1$ controllers. A prototype SDN controller that is resilient to Byzantine failures is analyzed in [8], which still poses the issue of heavy load on the controllers. In [6], an improved approach is proposed wherein each switch broadcasts its flow setup requests to $f+1$ primary controllers and checks the consistency of the forwarding rules received. If the switch detects an inconsistency, it sends the requests to f backup controllers and determines the forwarding rule based on the rules sent by the backup controllers. However, such an approach also leads to heavy load on the control plane due to broadcast of each and every request leading to slower response times of controllers to switches.

Our proposed approach, SGRS, differs from the above works by adopting a probability based randomization approach to replicating rules to multiple controllers. The probability distribution for randomization is obtained by modeling our problem as a Stackelberg game. We also consider the relative importance of switches in the network in addition to managing load at the controllers.

III. Proposed Scheme

In this section, we first describe an overview of our approach in Section III-A followed by an illustrative example in Section III-B. We then present the problem formulation of SGRS based on the Stackelberg game and our proposed heuristic to obtain the set of actions for SGRS in Section III-C and III-D.

A. An Overview

We consider a network with distributed controller architecture. Each controller can serve a switch in any one of the following three roles: i)primary, ii) inspector and iii) backup role. A controller can serve more than one switch and can play different roles for different switches. Every switch is mapped to three controllers; each controller in a different role for a switch as described below:

- **Primary Role**: A controller in “primary” role for a switch receives all the flow setup requests that the switch generates during the operation of the network and responds with the forwarding rules.
- **Inspector Role**: We consider a time slotted approach with suitable time interval for the operation of the inspector role. A controller in an “inspector” role for a switch makes a decision at the beginning of the time slot, based on the probabilistic randomized strategy generated by SGRS, to determine whether it should accept flow setup requests from the switch for the duration of the time slot. If it decides to accept requests from the switch for that time slot, it instructs the switch accordingly. It then responds with the forwarding rules for the requests during that time slot. Thus, in such a scenario, the switch receives a forwarding rule for a request from both the primary and the inspector. Note that a switch sends requests to its inspector only when instructed by the inspector.
- **Backup Role**: If a switch detects an inconsistency in the forwarding rules that it receives from its primary and inspector controller, the switch sends the flow setup request to its “backup” controller and the malicious controller is detected based on the forwarding rule sent by the backup controller.

We assume that the switch has some limited processing capability to perform some simple operations [9]; therefore it can check the consistency of the rules sent by the controllers. We discuss the major assumptions made below:

- **Switch-controller mapping**: We assume that the switches have already been mapped to the controllers. All the controllers in the network perform primary, inspector and backup roles for three disjoint sets of switches respectively.
- **Malicious controller attack model**: We consider at most one compromised controller in the network. We define an attack as fraudulent rule installation by the compromised controller to a flow setup request from a switch. We assume that a malicious controller will be an intelligent and stealthy attacker and aim to leverage its position. Considering the extent of efforts required to compromise a controller, it would be a valid assumption to make that

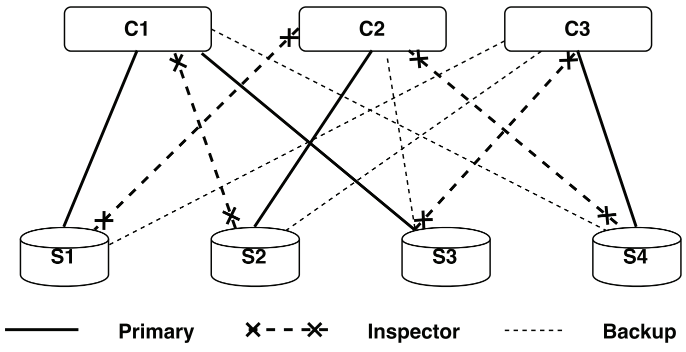

Fig. 1: Illustration of SGRS

Table I: Switch-Controller Mapping

Controller	Primary	Inspector	Backup
C1	S1, S3	S2	S4
C2	S2	S1, S4	S3
C3	S4	S2	S1, S2

the attacker would try to prolong his presence in the network as much as possible. Thus, we assume that the attacker would try to attack only one of the switches in a given time slot and avoid attacking multiple switches as it would increase the chances of his detection. The malicious controller would not respond with fraudulent rules when it receives requests as an inspector or backup controller, since the inconsistency in the rules would be detected and the attacker would be caught.

Each controller has a set of switches for which it is the inspector controller and generates the randomization strategy to check those switches independently. The relative importance of a switch is captured by the criticality parameter. A higher value of criticality for a switch indicates higher importance of that switch. The criticality can be decided based on factors such as the nature of traffic handled by the switch. We introduce the load control parameter, protection level, to allow each controller to manage the load due to the switches for which it is an inspector. The protection level determines the average load in each time slot that would be inspected by the controller. Such an inspection of load helps determine the malicious controller. For example, if protection level is 40%, it means that the controller expects on average 40% of total requests coming from the switches for which it is an inspector in a time slot. We assume that the rate of flow setup requests at each switch is known after observation over a suitable period of time. In SGRS, we first determine the set of actions for a controller through the heuristic described in Section III-D. Each action consists of a subset of switches that the controller would inspect in the time slot if that particular action is chosen according to a probability value. We then solve the Stackelberg game at each controller to obtain their respective mixed strategies that determines the probabilities with which each action is taken.

B. Illustrative Example

The switches and controllers are mapped as shown in Fig. 1 and Table I. If $C1$ is a compromised controller, based on our assumptions, the requests sent by $S1$ and $S3$ could be targeted by $C1$ for manipulation. Let's assume that $S1$ has higher criticality and the attacker receives higher benefit attacking $S1$ as compared to $S3$. The inspector controller for $S1$ is $C2$. Controller $C2$ is the inspector for $S1$ and $S4$ and it can take one of three possible actions every time slot: i) check $S1$, ii) check $S4$ and iii) check $S1$ and $S4$. If all the three actions satisfy the protection level constraint, our approach, SGRS (to be discussed in detail later), generates the optimal mixed strategy for $C2$ as shown in Table II.

Table II: Mixed Strategy for $C2$

Action	Probability
$S1$	0.35
$S4$	0.25
$S1, S4$	0.4

In every time slot, $C2$ selects one of the actions based on the probability distribution obtained through SGRS. Let's assume that the attacker manipulates a request sent by $S1$. If $C2$ selects action 1 or action 3 (Table II), all the flow setup requests from $S1$ are sent to both $C1$ and $C2$ for that time slot. $S1$ receives the forwarding rules sent by both $C1$ and $C2$ and detects an inconsistency. $S1$ then sends the request for which it received inconsistent rules to its backup controller $C3$, whose response helps to determine the misbehaving controller, i.e. $C1$ in this case. In larger networks, each controller plays a primary role for many switches and performs the inspector role for another set of switches. Since inspecting every single request from the switches imposes high load on the controller and increases response times, through SGRS, each controller will choose a subset of switches it has to inspect in each time slot to manage the load, while offering a certain degree of protection for the network.

C. SGRS Problem Formulation

Consider a network with N switches and K controllers. As described in the previous section, each controller has disjoint sets of switches for which it performs the primary, inspector and backup role respectively. We describe the problem formulation for one controller, since each controller solves the same problem independently with the relevant parameter values. Therefore, we consider a controller that has a set of S switches for which it is in the inspector role. The protection level parameter is represented by ρ which also corresponds to the average fraction of traffic sent to the inspector to enable attack detection. The relative importance of a switch S_i is captured by the criticality parameter c_i. The processing capacity of the controller is represented by τ.

We model the problem as a normal form Stackelberg game. A Stackelberg game is a model in which one player, termed as "leader", makes the first move and the other player, termed as "follower", observes the leader's move and then makes the move that maximizes his utility. Thus, the players take their actions sequentially rather than simultaneously. Stackelberg game is suited to model real world security problems such as security patrolling for airports, public infrastructure etc. since security measures are often taken beforehand and the attacker

Table III: Mathematical notations

Notation	Description
N	Number of switches in the network
K	Number of controllers in the network
S	Set of switches for which a controller is in inspector role
c_i	Criticality of switch S_i
τ	Total processing capacity of each controller
ρ	Protection Level
A	Set of actions for attacker
D	Set of actions for defender
$U^{att}_{a,d}$	Utility obtained by attacker for attacker action $a \in A$ and defender action $d \in D$
$U^{def}_{a,d}$	Utility obtained by defender for attacker action $a \in A$ and defender action $d \in D$
P^D_i	Probability of detecting an attack on switch S_i
p_d	Probability of choosing defender action $d \in D$

can observe the defender's strategy before making a move [7]. When the game is played in the security context, the "leader" and "follower" are referred to as "defender" and "attacker".

In the context of our problem, the inspector controller is the defender while the compromised controller is the attacker. Let D denote the set of actions that a defender can take while A is the set of actions that an attacker can execute. D is obtained through the heuristic algorithm described in Section III-D. Each action represents a subset of switches of S, from which it will receive the flow setup requests during that time slot. As explained in Section III-A, we assume that the attacker will not attack multiple switches at a time since it increases the risk of detection. We assume that the attacker will attack only one switch at a time. Thus, the action set of the attacker is the set of S switches where an action in A implies an attack on a particular switch in a time slot. We also assume the game to be a complete information game where both players are aware of the actions that the other player can take and their benefits.

The utility functions for the attacker and defender are represented by U^{att} and U^{def} respectively. If action $a_i \in A$ corresponds to an attack on switch S_i, and the defender happens to play action $d_j \in D$ in that time slot, the utilities obtained by the attacker and defender are shown below:

$$U^{att}_{a_i,d_j} = c_i, \quad U^{def}_{a_i,d_j} = 0, \quad S_i \notin d_j \tag{1}$$

$$U^{att}_{a_i,d_j} = 0, \quad U^{def}_{a_i,d_j} = 1, \quad S_i \in d_j \tag{2}$$

Eq. (1) shows the utility that the defender and attacker obtain if the attacked switch S_i is not present in set of switches of action d_j (inspector controller does not detect attack) while Eq. (2) shows the case where switch S_i is present in d_j (inspector controller detects the attack). In a Stackelberg game, the defender has the first mover advantage over the attacker and aims to deploy a mixed strategy to maximize its utility. After the defender makes the first move, the attacker chooses the action that provides it with the best utility. The utility values have been chosen to reflect the risk and reward that the attacker runs in attacking a switch. The expected utility $U^{att}_{a_i}$ of an attack on switch S_i is shown by Eq. (3) and corresponding expected utility for defender $U^{def}_{a_i}$ is given by Eq. (4):

$$U^{att}_{a_i} = c_i(1 - P^D_i) \tag{3}$$

$$U^{def}_{a_i} = P^D_i = \sum_{d \in D} p_d U^{def}_{a_i,d} \tag{4}$$

As seen in Eq. (3), the utility that an attacker gains on attacking a switch depends on the critical nature of the switch and the probability of detection of the attacker. From the attacker's perspective, an attack would be most beneficial if the attacked switch is highly valuable and has a low probability of detection. The defender's expected utility or probability of detection for a switch is represented by Eq. (4), which is the sum of the probabilities of those actions which include the attacked switch for consistency checking. The optimal strategy for the defender is obtained by backwards induction after assuming an action which would represent the best possible action for the attacker [10]. For every attacker action a_i, the following optimization problem is solved:

$$\text{Maximize:} \quad \sum_{d \in D} p_d U^{def}_{a_i,d} \tag{5}$$

$$\text{subject to:} \tag{6}$$

$$\forall a_j \in A, \sum_{d \in D} p_d U^{att}_{a_i,d} \geq \sum_{d \in D} p_d U^{att}_{a_j,d} \tag{7}$$

$$\sum_{d \in D} p_d = 1 \tag{8}$$

$$p_d > 0, \forall d \in D. \tag{9}$$

The objective function in Eq. (5) represents the maximum utility or probability of detection of an attack that the defender can achieve if action a_i represents the attack on switch S_i that provides the best utility for the attacker. The constraint that utility of action a_i is the highest for the attacker is represented by Eq. (7). Eq. (8) and Eq. (9) represent the constraints for valid probability values. While attacking a_i is the best course of action for the attacker, by solving the above optimization problem, we obtain the mixed strategy that provides maximum utility for the defender. Similarly, for each action in A, we solve the above optimization problem to obtain the best utility for the defender in each case. After obtaining all the solutions for each action in A, we compare the maximum utility values obtained for the defender and select the mixed strategy that generates the highest utility value for the defender and that is chosen as the optimal strategy for the defender. Each controller runs the above game independently to obtain the mixed strategy for randomization for their respective sets of switches for which it is the inspector.

D. Heuristic for Defender Action Space in SGRS

For a given set of S switches for which a controller performs the role of an inspector, the total number of possible actions for the controller is given by:

$$TotalNumberofActions = \sum_{i=1}^{|S|} \binom{|S|}{i} \tag{10}$$

Considering all the actions as part of the controller's action space for the Stackelberg game could make the problem intractable; hence, we develop a heuristic algorithm to limit the

Algorithm 1 Heuristic to determine action space

```
Input: $S$, $r$, $l^l$, $l^u$, $c$,$\rho$, $T$.
Output: $D$.
1: $T^s \leftarrow T$ sorted in descending order
2: Find min. $u$ s.t. $\sum_{i=1}^{u} T_i^s \geq \rho \sum_{i=1}^{|S|} T_i^s - l^l$;
3: Find max. $v$ s.t. $\sum_{i=1}^{v} T_{|S|-i+1}^s \leq \rho \sum_{i=1}^{|S|} T_i^s + l^u$;
4: if $v < u$ then
5:     $v \leftarrow u$;
6: end if
7: $F = \{\binom{S}{i}, i = u...v\}$; /*Reduced set of available actions*/
8: $\forall\ f_i \in F$, weight $w_i = \frac{1}{|f_i|} \sum_{S_j \in f_i}(c_j)$;
9: If available, $D$ is set of $r$ actions with highest weight
   from $F$ that cover set $S$ with load of each action in range
   $[\rho \sum_{i=1}^{|S|} T_i - l^l, \rho \sum_{i=1}^{|S|} T_i + l^u]$.
10: Else, $D$ is set of actions with highest weight from $F$
    until they cover set $S$ with load of each action in range
    $[\rho \sum_{i=1}^{|S|} T_i - l^l, \rho \sum_{i=1}^{|S|} T_i + l^u]$.
```

pool of actions from which the final set of actions is chosen. The pseudo-code of the algorithm is presented in Algorithm 1. The inputs to the algorithm include network parameters such as the set of switches S for which the controller is an inspector, the protection level ρ, the load at each switch given by matrix T, and the criticality of each switch given by matrix c. The range of allowed traffic load for each action is given by $[\rho \sum_{i=1}^{|S|} T_i - l^l, \rho \sum_{i=1}^{|S|} T_i + l^u]$, where l^l and l^u help determine the range of allowed traffic loads in relation to the protection level of the network. The output of the algorithm is the action space of the controller,A. The heuristic reduces the action set by finding u and v as shown in line 2 and 3 of Algorithm 1. Thus, the number of actions in the reduced set is given by:

$$ReducedNumberofActions = \sum_{i=u}^{v} \binom{|S|}{i} \quad (11)$$

The heuristic algorithm then finds the weight of each action in the reduced action set with the weight function signifying the mean criticality of the each action f_i (line 8). The number of actions chosen is limited to r actions (if available) with the highest weight if they cover the entire set of switches, or until they cover the entire set of switches (lines 9 and 10). The complexity of the algorithm is O($\binom{|S|}{\rho|S|}$) when the range of load variation is kept small.

IV. Performance Study

We evaluate the performance of our proposed scheme, SGRS, by comparing it against three other approaches:

- **Naive Uniform Heuristic (NUH)**: In the first approach, if protection level is ρ, in each time slot, the controller checks each switch for which it is an inspector with probability ρ.
- **Naive Weighted Heuristic (NWH)**: We also develop another method for comparison in which the controller checks switch S_i with probability $p_i = \rho c_i / \sum_{j=1}^{|S|} c_j$. The weights are calculated based on the relative criticality of each switch.

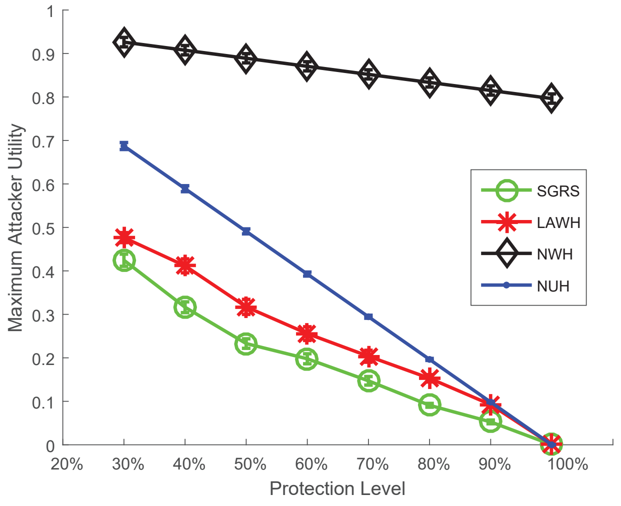

Fig. 2: Maximum Attacker Utility of SGRS vs. Others.

- **Load Aware Weighted Heuristic (LAWH)**: We extend the NWH approach by adding some intelligence to it so that the load inspected by the controller is equal to the protection level. We initialize the probability for each switch by following the NWH. Then, the switches with highest criticality are considered and their remaining load is distributed in equal proportion to ensure the total load at the controller is equal to the protection level. If those switches cannot satisfy the required load level, the next set of switches with highest criticality are selected along with the former and load is distributed in equal proportion again. This continues until the the total load at the controller satisfies the protection level constraint.

The metrics for evaluating the approaches are: i) maximum utility for attacker, ii) detection probability of attacker, and iii) load imposed at the controller.

A. Parameter Settings

We consider a controller with 10 switches for which it functions as an inspector. Each switch has a load in the range of $[100, 200]$ packet-in messages/time slot. The criticality of each switch is in the range of $[0.1, 1]$. A lower value of criticality indicates a switch that handles less sensitive traffic and provides relatively less benefit for an attacker. We vary the protection level in the range of $[30\%, 100\%]$. The processing capacity of the controller is 8000 packet-in messages/time slot. The load due to the switches for which it serves as a primary controller is 6000 packet-in messages/time slot. In our heuristic algorithm to generate the defender's action set, we set the number of actions, r, as 30 and the lower load limit l^l and upper load limit l^u at 100 and 0 respectively.

B. Results and Analysis

In our numerical analysis, for a fixed protection level, we vary criticality and load at the switches randomly in their respective ranges for every run and obtain the results. We perform the analysis one hundred and fifty times and plot the results with 95% confidence interval.

Fig. 2 shows the maximum utility that the attacker can achieve by attacking one of the switches in the network. As

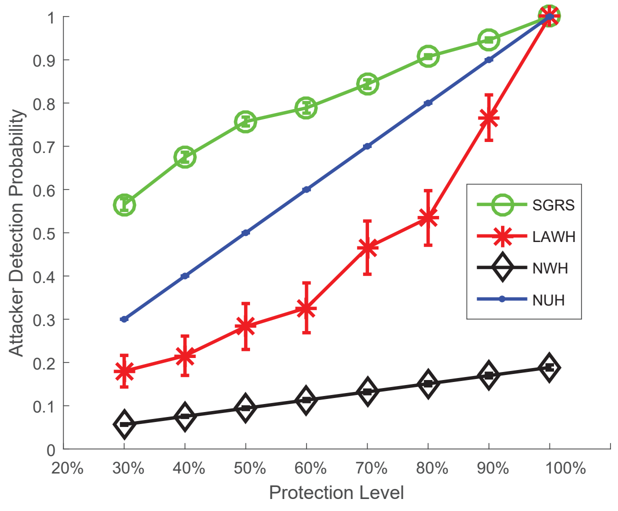

Fig. 3: Attacker Detection Probability of SGRS vs. Others.

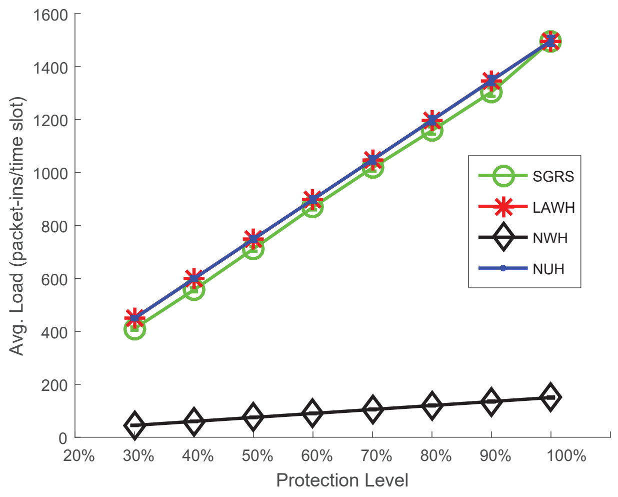

Fig. 4: Average Controller Load of SGRS vs. Others.

the protection level increases, the amount of requests that the inspector controller can check for consistency increases, thereby increasing the probability of detection of the attacker and making an attack riskier. This explains the decrease in attacker utility with increase in protection level (Eq. (3)). SGRS reduces the maximum attacker utility by up to 31% and 60% compared to LAWH and NUH respectively as seen in Fig. 2, and significantly outperforms NWH.

We observe from Fig. 3 that SGRS also achieves higher probability of attacker detection as compared to the other approaches. The probability of attacker detection is the probability of detection of the switch that provides the highest utility for the attacker. SGRS improves attacker detection probability by up to 210% and 88% as compared to LAWH and NUH . SGRS also significantly outperforms NWH in this metric.

The expected load at the controller due to the switches for which it is an inspector is depicted in Fig. 4. The results in Fig. 4 show that SGRS imposes lesser load on the controllers and still outperforms the other heuristics. The poor performance of NWH can be explained by Fig. 4 as the load sent is much lower than the expected load for each protection level. The performance of SGRS, LAWH and NUH converges when protection level is 100% since all the requests are being sent to the inspector controller in every time slot.

Thus, for similar levels of load sent to the controller, SGRS outperforms the other heuristic approaches by reducing the maximum attacker utility by up to 31% and improves the attacker detection probabity by up to 88% while imposing lesser load at the controller.

V. Conclusion

SDN architecture introduces new security threats to the network. A compromised controller in the control plane can engage in malicious behavior such as fraudulent rule installation and cause significant damage while avoiding detection by behaving intelligently and stealthily. Existing approaches towards detection of such a malicious controller lead to high load on the control plane and increased network cost. We developed a game theory based approach to effectively detect the presence of a malicious controller by randomizing the checking of forwarding rules in switches to alleviate the load at the controllers. We modeled our problem based on the Stackelberg game and considered load at the controller and relative importance of switches while obtaining the optimal mixed strategy solution for the game. We also developed a heuristic algorithm to obtain the action space for the defender to make the problem tractable. We compared our proposed approach against three other approaches and the performance results show that our approach outperforms the other approaches as our proposed approach achieves up to 31% improvement in reducing maximum attacker utility and up to 88% improvement in attacker detection probability.

Acknowledgment

This work was supported by Singapore MoE AcRF Tier-2 Grant MOE2015-T2-2-116, NUS WBS R-263-000-C17-112.

References

[1] S. H. Yeganeh *et al.*, "On scalability of Software Defined Networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013.

[2] A. Tootoonchian and Y. Ganjali, "Hyperflow: A Distributed Control Plane for Openflow," in *INM/WREN'10*, San Jose, USA, Apr. 2010.

[3] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2016.

[4] D. B. Rawat and S. R. Reddy, "Software defined networking architecture, security and energy efficiency: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 325–346, 2017.

[5] H. Li, P. Li, S. Guo, and S. Yu, "Byzantine-resilient secure software-defined networks with multiple controllers," in *Communications (ICC), 2014 IEEE International Conference on*. IEEE, 2014, pp. 695–700.

[6] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, "Primary-Backup Controller Mapping for Byzantine Fault Tolerance in Software Defined Networks," in *IEEE GLOBECOM 2017*, 2017, pp. 1–7.

[7] B. An, M. Tambe, and A. Sinha, "Stackelberg security games (ssg): Basics and application overview," *Improving Homeland Security Decisions. Cambridge University Press, forthcoming*, 2015.

[8] K. ElDefrawy and T. Kaczmarek, "Byzantine Fault Tolerant Software-Defined Networking (SDN) Controllers," in *IEEE COMPSAC 2016*, vol. 2, 2016, pp. 208–213.

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

[10] V. Conitzer and T. Sandholm, "Computing the optimal strategy to commit to," in *Proceedings of the 7th ACM conference on Electronic commerce*. ACM, 2006, pp. 82–90.