Using NFV and Reinforcement Learning for Anomalies Detection and Mitigation in SDN

Lauren S. R. Sampaio, Pedro H. A. Faustini, Anderson S. Silva, Lisandro Z. Granville and Alberto Schaeffer-Filho
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
Email: {lsrsampaio, phafaustini, assilva, granville, alberto}@inf.ufrgs.br

***Abstract*—Computer networks are subject to several anomalies, which leads to the necessity of techniques to coordinate detection and mitigation to keep the network operational. In this paper we propose the use of reinforcement learning to promote resilience in Software Defined Networking (SDN). In particular, it is proposed collecting network metrics and grouping them into profiles, each one having a set of actions that handles problems using reinforcement learning, Network Functions Virtualization (NFV), and an SDN controller. Policies for dealing with anomalies are defined based on rewards for each action. Results show that the system obtains mostly positive rewards, but a small increment in the topology size leads to more than four times the number of entries in the state-action table.**

***Index Terms*—software defined networking, network functions virtualization, machine learning, reinforcement learning, anomaly detection**

I. Introduction

Computer networks are impacted by a large range of anomalies. By anomaly it is understood not only the presence of malicious traffic flows, as in denial of service (DoS) attacks, but also the combination of benign flows that may deteriorate the user experience. The goal of this paper is to investigate how Software-Defined Networking (SDN) can benefit from techniques based on reinforcement learning [1] and Network Functions Virtualization (NFV) by selecting mitigation strategies depending on the type of anomaly inflicting the network.

In this context, the main contributions of this paper are: *(i)* a brief review of the state-of-the-art on reinforcement learning in computer networks; *(ii)* the development of an agent that uses reinforcement learning to deal with different types of anomalies; and *(iii)* the implementation of a prototype and evaluation of aspects such as memory consumption and performance.

This paper is organized as follows: Section II presents the theoretical basis of SDN, NFV and reinforcement learning. Section III proposes the model that combines NFV, and reinforcement learning to promote resilience in SDN. Section IV introduces the implemented prototype and results. Section V discusses related work and Section VI presents conclusions and future works.

II. Background

A. Software-Defined Networking

SDN considers the existence of a controller software that centralizes the network routing logic, decoupling the data plane (forwarding packages) from the control plane (routing) [2]. A fundamental element in the control plane is the controller, which has a global view of the network and, therefore, is able to optimize the management of flows in a flexible and scalable manner.

The OpenFlow protocol [3] is the *de facto* communication standard between the data and control planes. The protocol acts as a channel between these two planes, making it possible to add or remove entries in the switch flow tables. In addition, programmable networks can overcome barriers to the adoption of new ideas, promoting innovation in computer networks [4].

B. Network Functions Virtualization

Computer networks usually rely on multiple functions running in the infrastructure, such as firewalls, intrusion detection systems, and load balancers. Typically, these functions are provided by dedicated hardware, which becomes easily outdated [4]. Another obstacle is the low flexibility in the management of these devices, once they are physically deployed. Network Functions Virtualization (NFV) allows these functions to run on general purpose hardware, in a virtualized environment. However, there are challenges, such as the performance in virtualized environments, as well as resilience when facing hardware and software failure [5].

There are three main components in the NFV architecture: infrastructure, service, and orchestration & management [6]. The *Infrastructure*, known as NFVI (NFV Infrastructure), covers hardware and software resources, as well as the virtualization environment. The *Service* component covers the virtualized network functions, or VNFs, which are selected from a repository. They are executed in virtualized environments instead of dedicated hardware (also known as middleboxes) [7]. Finally, *Orchestration & Management*, known as NFV-MANO, focuses on more specialized tasks. For instance, if there is a firewall and an intrusion detection system (IDS) implemented as VNFs, the NFV-MANO determines the order in which packets of a flow should traverse these VNFs.

978-1-5386-6950-1/18/$31.00 ©2018 IEEE

C. Reinforcement learning

Reinforcement learning relies on an agent that analyzes the environment and interacts with it. Given an action taken at time t, at the moment $t+1$ the agent changes state and receives a reward [1]. Reinforcement learning problems can be represented as a Markov Decision Process (MDP), that defines a 4-upla [8] $\langle S, A, R, P \rangle$, where S is a set of states, A is a set of actions, R is a reward function (which returns the reward at the state $s+1$, when the action a is executed at the state s) and P is the transition probability function (which returns the probability of reaching the state $s+1$ when the action a is executed at the state s). The reward and the probability function compose the environment dynamics.

In most problems, the environment dynamics are not available [8], and reinforcement learning problems usually involve a value-function estimative. A state-action table, or Q-Table $Q(s,a)$, defines what will be the received reward when applying action a at state s. This table is updated based on the reward obtained by the agent. There are multiple ways of defining the reward, being the Sarsa algorithm [1] one of the most popular, which is shown at Algorithm 1.

Algorithm 1: SARSA

Input: Q-table
Output: updated Q-table
$\forall s \in S, \forall a \in A, Q(s,a) = 0$ /* Initializes Q(s,a) */
foreach *episode* **do**
  Choose $a \in A$ available in $s \in S$ following a policy (*e.g.* ϵ-*greedy*)
  foreach *time step of the episode* **do**
    Execute the action a, observe s', r
    Choose $a' \in A$ available in $s' \in S$ following a policy (*e.g.* ϵ-*greedy*)
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r - Q(s,a)]$
    $S \leftarrow S', A \leftarrow A'$

The reward is denoted by *r* and α is the learning rate. The higher α is, the higher will be the variance of the updated values in the table (based on a reward). While it learns, the agent may face a dilemma: take an action whose reward is known or explore new possibilities and acquire new knowledge about the consequences of its actions. A popular and effective technique that deals with this dilemma is $\epsilon - greedy$ [1]. In this technique, a ϵ factor indicates the probability of the agent to try new possibilities when facing known options. As an example, if $\epsilon = 0.1$, in 10% of the situations the agent will try an unknown approach.

III. Reinforcement learning in SDN/NFV-Based Infrastructures

In this paper, we propose a model of reinforcement learning integrated into the NFV architecture by means of an agent that resides in the orchestrator. The agent receives network metrics and builds network profiles. Each profile deals with a specific anomaly and consists of a set of actions, as well as a table of states, and a table of state-actions. At each learning cycle (an arbitrary time step) the agent selects a profile, executes an available action and receives a reward – that updates the profile's Q-Table. This method, combining both SDN/NFV and reinforcement learning, guarantees the flexibility necessary to react to network conditions on demand.

At each cycle, the agent executes only one action of a profile, in order to avoid multiple actions affecting one another. Each profile may have its own priority, so that if an anomaly is detected by more than one profile, the agent will prioritize one of these when choosing a mitigation action. Further, profiles may be combined so that the system works harmonically. In the next subsections, we present three of such profiles: *(i)* load balancing, *(ii)* server replication, and *(iii)* denial of service attack.

A. Load balancing

If the links connected to forwarding devices, or switches, present high throughput, the switch input queues (buffers) may overflow, causing packets to be discarded thus deteriorating the user experience. Therefore, to avoid this overload, it may be more appropriate to forward packets through alternative routes. The rationale is that if a longer route is less congested, it may be a better path than a shorter one. The actions of this profile are *(i)* void action, *(ii)* use the shortest route between two hosts, and *(iii)* modify the route between two hosts.

A route is said to be congested (high status) if one of its links has an occupation equal or higher than a maximum threshold – in this study it is arbitrary, once the objective is to measure the efficacy of our model –, *e.g.* 80%. The status of a route is considered average if the load is moderately occupied, say, between 30% and 80%. Otherwise, its status is considered low. A link may be part of multiple routes, which leads to the situation where the traffic of each flow is low but when combined the link is overloaded and all of these routes present a high status.

At each learning cycle the agent collects information from the flow tables of the switches along the routes. In each time step, the agent has access to the current state and the set of possible paths - calculated by the controller for each pair of hosts. Having this information, it can request the controller to modify the flow tables of the switches in order to change the routes. The agent will only modify a route between two hosts if one or more links in this route has a *high* status. There are two objectives: *(i)* avoid link congestion *(ii)* choose always the shortest path possible. The reward given to the agent is 1 if after executing the action there are no links with high status, or -1 otherwise. Formally:

$$R = \begin{cases} -1 & \text{if there is a link with high traffic} \\ 1 & \text{otherwise} \end{cases}$$

Intuitively, the agent will try to load balance in order not to overload any link. To this end, it may be necessary changing the route of certain flows so they will not follow the minimum

path between source and destination, but the less congested one. It can do so prematurely and be punished, or rewarded. Over time, it will learn which states lead to the minimum path.

B. Server replication

A server serving multiple clients may get overloaded if it receives too many requests. This problem can be overcome by using VNFs that replicate this server, which can be instantiated in other places of the topology and absorb part of the requests that overloaded the access link to the main server. The creation of a server replication profile is given by three metrics: *(i)* the traffic in the access link between server and switch, *(ii)* the set of servers or replicated VNFs, and *(iii)* the paths to the servers or replicas.

To simplify the model, we differentiate the traffic volume in a link into three ranges: low (*e.g.*, [0%, 30%) of usage), average (*e.g.*, [30%, 80%)), and high (*e.g.*, [80%, 100%]). The set of VNFs is obtained from the repository in the NFV orchestrator, and we assume that they can be bound only to the switches that are in the path to the server. The number of possible states will depend on the size of the network: the larger it is, the more states may exist based on the combination of the three metrics above. The server replication profile defines three possible actions: *(i)* instantiate a replica of the server, *(ii)* remove an instantiated replica, and *(iii)* void action.

The removal of a replica only occurs when the access link to the server reports low traffic. When there is a high throughput, it is expected a higher number of instantiations than in the case of an average throughput situation, for example. The agent will learn how many replicas it needs to instantiate and where they should reside according to the load of the access link to the server. The agent has two objectives: *(i)* to keep the server access link to the switch in a low state, and *(ii)* to do it using the minimum amount of resources. The reward given to the agent is 1 if after the action the link has a low traffic status, or -1 otherwise. Formally:

$$R = \begin{cases} 1 & \text{if the link has low traffic} \\ -1 & \text{otherwise} \end{cases}$$

Depending on how the traffic volume changes, the agent searches for a minimum configuration of VNFs as to relief the load in the server access links. It is possible that the agent executes an action that brings a link from a low to an average state, for example. It occurs because such link had a low traffic status due to multiple instantiated VNFs, but when exploring a configuration with less VNFs, the link became overloaded and the agent was punished with a negative reward. Over time, it will learn how to react when facing each traffic configuration.

C. Attack mitigation

During an attack there is no option other than mitigating malicious flows. It makes sense that this profile receives a higher priority than the previous profiles. The agent must be able to distinguish malicious flows from legitimate ones, to mitigate the attack and to apply actions from the other profiles on the legitimate traffic. This profile does not use reinforcement learning itself, thus there is no state-action table. This profile has only two states: *(i)* malicious, and *(ii)* benign.

While an attack is not detected, the agent executes actions from lower priority profiles. The controller monitors mean and standard deviation of the number of requests each server receives. It can obtain this information by querying the flow tables of the switches to determine which flows are intended to which servers. The controller also registers which switches each host (identified by its IP) connects to.

An application executing in the SDN application plane collects this information and applies the following heuristic: being r the normal number of requests that the server receives, and c the number of clients that it is connected to, the application assumes that on average each client must send r/c requests. Clients above this threshold plus three deviations must be blocked. In any case, the agent instantiates the VNFs that implement the firewalls to block packets originated from the suspect IPs.

IV. Prototype and results

In this section the implemented prototype is described, which is publicly available[1]. Subsection IV-A elaborates details on the developed prototype, while Subsection IV-B presents experiments and results.

A. Prototype implementation

The developed prototype implements both attack mitigation and load balancing profiles. We used the Mininet platform[2] for emulating network topologies, and POX[3] for the development of the controller. In addition, since Mininet does not provide native support to NFV, network functions where implemented in Docker[4] containers. These are Linux-based containers, that share the kernel with the host system, but isolation is provided between applications, as in virtual machines.

Docker provides the basic infrastructure to host network functions but it does not support orchestration. Thus, our implementation extends Containernet[5], which couples Docker containers with Mininet, by integrating the reinforcement learning agent to the controller. The controller is responsible for determining which packets must pass through instantiated firewalls before arriving at their destination, also collecting metrics and sending them to the agent, that builds the profiles and network states. It is the agent, on the other hand, that makes the necessary calls to instantiate the VNFs defined by Dockerfiles and commands a firewall to stop certain flows. The agent also sends to the controller action requests, such as route changes. The controller also calculates the possible paths between hosts and sends this list to the agent.

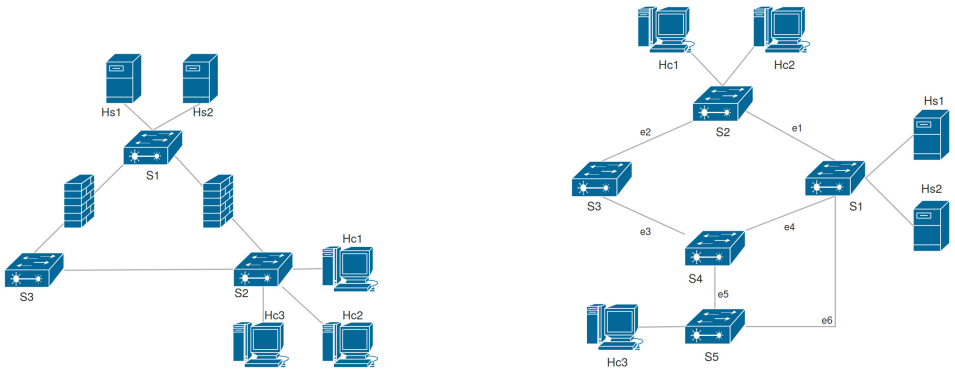
[1] https://github.com/ComputerNetworks-UFRGS/reinforcement-containernet

[2] http://mininet.org/

[3] http://sdnhub.org/tutorials/pox/

[4] https://www.docker.com/what-docker

[5] https://github.com/containernet/containernet



(a) 1st experiment's topology (b) 2nd experiment's topology

Fig. 1. Topologies used in both experiments

B. Experimental analysis

In the following experiments, servers are implemented in Docker containers and send video files to clients so as to generate traffic via TCP connections. Clients are usual hosts of Mininet. The links that connect clients and servers to switches have a maximum throughput of 5 Mbps, and the others have a maximum throughput of 11 Mbps. The bandwidth consumed by the clients is 5 Mbps, which is the limit of the links that connect them to the switches. The experiments were run in a machine with a 3.4 GHz Intel i7 processor and 16 GB RAM, in a virtual machine executing Ubuntu 14.04 (with 10 GB RAM allocated to it). In the experiments, it was adopted a learning rate of $\alpha = 0.1$ (to avoid that an isolated action heavily impacts the agent decision policy), an exploration rate starting in $\epsilon = 0.4$ (as it is expected that the agent explore primarily a known action, but not rarely discover the consequences of other actions) and two seconds for each time step. The topologies used are shown in Fig. 1.

In the first experiment, there are two servers (*Hs1* and *Hs2*) and three clients (*Hc1*, *Hc2* and *Hc3*), as shown in Fig. 1(a). The agent tries to load balance the traffic to the servers and also has to mitigate an attack. In this case, the mitigation profile has a higher priority than the load balancing profile. Two VNFs that implement a firewall have the function of protecting the servers from attack traffic traversing switches *s2* and *s3*. It was also stipulated that after 40 time steps the exploration rate would decrease to $\epsilon = 0.1$. These values were chosen because after starting with a higher exploration rate the agent has enough actions to explore in the Q-table, and thus the exploration rate can be reduced. Clients *Hc1* and *Hc3* connect to server *Hs1*, and client *Hc2* does the same with server *Hs2*. Initially, the exchange of packets happens through a link that connects switches *s1* and *s2*. The file exchanges in each connection have 150 MB. Client *Hc3* is malicious and executes several TCP requests so as to overload server *Hs1*. Quickly the links get overloaded due to multiple files being transmitted, and the agent does not find an alternative route that generates a positive reward. The attack is then detected, and the agent commands firewall VNFs to block the flow from and to *Hc3*. Fig. 2 shows the immediate rewards obtained by the agent for the load balancing profile.

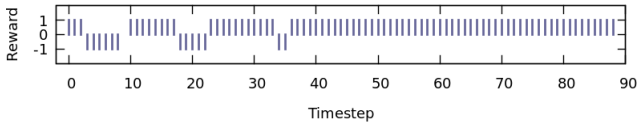


Fig. 2. Immediate rewards obtained.

As can be seen in Fig. 2, during the attack identification stage, at time step 9, there was no reward, as the active network profile was the mitigation, which had a higher priority than the load balancing. After the learning rate was reduced to 0.1, at time step 40, it is possible to observe that the agent only received positive rewards. This shows that it found indeed actions that avoid the overload of the links; otherwise it would receive negative rewards. Regarding the attack itself, Fig. 3 shows the amount of requests sent by the attacker – *Hc3* (red) – and the number of requests that actually arrived at the server – *Hs1* (blue). Until time step 9, all requests sent by the attacker arrived at the victim. However, after that it is possible to see that the firewalls blocked the attacker flow, showing that the victim was shielded from the requests despite the persistent attempts of the attacker.

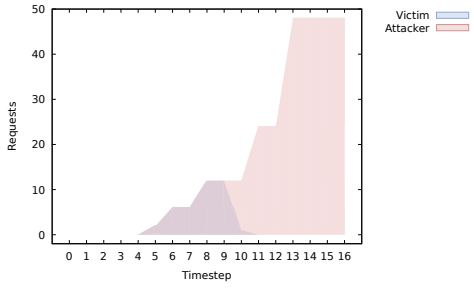


Fig. 3. Amount of requests sent by the attacker and received by the victim.

The second experiment deals only with the load balancing, but in a larger topology, allowing more combinations of alternative routes, as Fig. 1(b) shows. Hosts *Hs1* and *Hs2* are

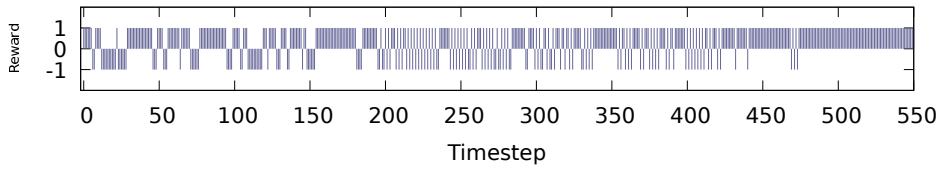


Fig. 4. Immediate rewards obtained over the time.

connected to switch *s1* and act as servers, while *Hc1* and *Hc2* are connected to switch *s2* and are configured as clients.

A 500 MB file is transferred between *Hs1* and *Hc1*, and an identical file is transferred between *Hs2* and *Hc2* at the same time. The initial route adopted by the controller corresponds to the minimal path between the hosts. After 100 time steps, the third client executes the same workload with server *Hs1*. Fig. 4 shows the immediate rewards obtained over time by the agent, during the load balancing. After 400 time steps, the exploration rate was decreased to $\epsilon = 0.1$, and the agent found a path that avoided bottlenecks, obtaining mostly positive rewards, as Fig. 4 shows. In 10% of the cases, it explores unknown actions, which leads to negative rewards, but in a proportion that is lower than in the beginning of the experiment.

By analyzing both experiments, we observed that the reinforcement learning technique presents low spatial performance due to the storage of states in a table. Despite the small increase in the topology size, the number of possible combinations to forward packets is increased in such a way that the agent covered many more states than before (from 31 to 104 states). This is reflected also in the size of the state-action table, that had an expressive increase in the number of added entries (from 39 to 173 entries). The comparison between the number of states and the number of entries in the state-action table of both experiments can be seen in Fig. 5.

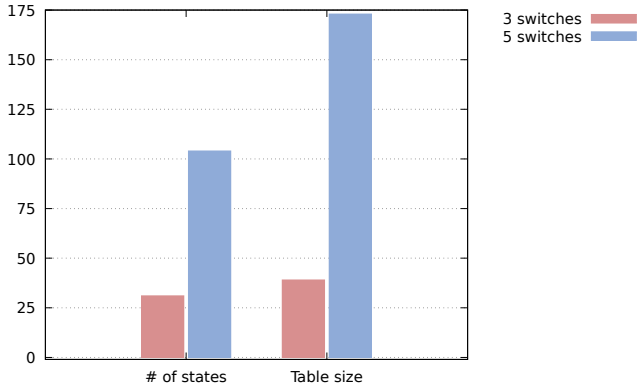


Fig. 5. Number of states and size of table.

The larger the number of collected metrics needed for the creation of a state, the higher is the probability of such phenomenon happening. In the load balancing case, as an example, even if the values that compose the states are discretized in *high*, *average*, and *low*, each state is represented by a matrix, in which each line has two fields (path and status). The higher the number of switches, the higher will be the number of possible paths between hosts, leading to a huge combination of possible states. The storage of such informations and the necessary time for the training of the agent may be costly in larger networks.

In addition, the number of entries in the Q-Table grows quickly as can be seen in Fig. 6. This suggests the need of finding alternatives for the storage of data in the table in order to avoid, or to mitigate, this phenomenon. But, as the exploration rate was reduced at the end of both experiments, the agent stopped exploring unknown actions. As a consequence, new combinations of traffic/action stopped being added to the Q-Table, and there was a stagnation of its growth.

V. Related work

Several research efforts have applied reinforcement learning techniques to address security and resilience issues in computer networks. This section briefly discusses related work combining denial of service attacks, reinforcement learning for load balancing, and anomaly detection in SDN/NFV infrastructures.

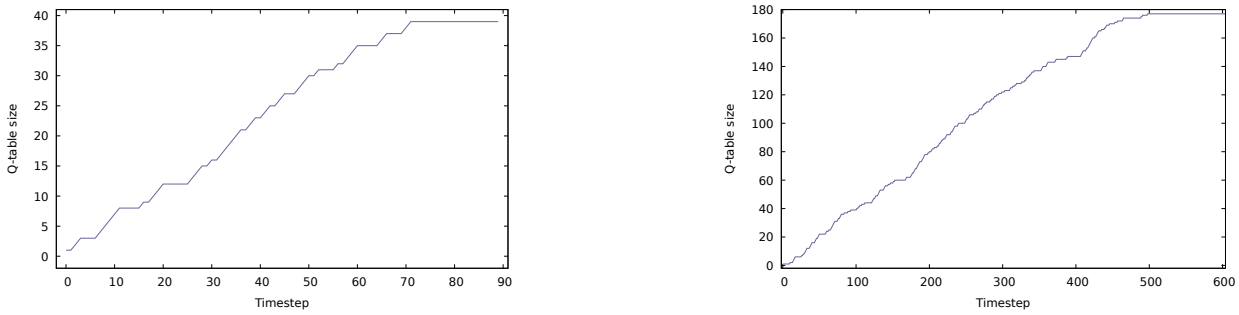
Machado *et al.* [9] propose an architecture that combines SDN and NFV promoting resilience. VNFs and controller monitor and mitigate attacks based on a control-loop that has a system's 'brain' function, but that does not use machine learning.

Belyaev *et al.* [10] use load balancing to deal with DDoS attacks. In this approach, malicious flows are not blocked, but redirected through the network. Our paper, on the other hand, chooses a different approach by dividing the network in profiles and load balancing only the legitimate flows, thus eliminating those considered malicious with the aid of NFV technology.

Malialis [8] proposes an approach to deal with denial of service attacks. His work uses multiple agents in routers, in a decentralized architecture so as to be more resilient to attacks. The agents work together, discarding packets in order to keep an operational service. The agents are placed at fixed points in the topology following the idea of Yau [11].

Hu and Chen [12] adopted supervised reinforcement learning to load balance. Their premise is that an alternative path, despite being longer, could be a more adequate than a shorter (and congested) one. They used the Q-Learning algorithm in a supervised reinforcement learning scenario. In our paper, we adopted reinforcement learning in order to load balance the traffic in an online manner, without the need of a supervisor.

In a similar work, Kim *et al.* [13] also blend SDN and reinforcement learning to deal with flow congestion. The focus



(a) 3-switches topology.

(b) 5-switches topology.

Fig. 6. Table size growth.

of the load balancing is to diminish the stress only on the links, without taking into consideration an occasional host that is the destination of all traffic. Our work proposes an agent that not only distributes traffic, but also - if necessary - instantiates server replicas when load balancing.

VI. Conclusion and future works

In this paper we proposed a detection and mitigation system for SDN with the aid of NFV technology and reinforcement learning-based techniques. It relies on an agent that resides in the orchestrator of the NFV architecture and collects evidences of anomalies from network metrics, building network profiles. The network profiles try to hierarchize the treatment of different threats, keeping the agent concentrated on eliminating first the more urgent anomalies. Such hierarchy also helps avoid that an action directed to eliminate a threat ends up canceling the effects of actions of other profiles.

This paper has as its main contribution the harmonic combination of different mitigation techniques for different anomalies (attacks and network overload). The performed experiments in a virtualized environment showed promising results. On the other hand, through the analysis of the results, it was possible to identify that the state representation has a high storage cost: a small increase in the number of switches causes a large increase in the amount of states that must be visited.

As future improvements, we plan to include additional network functions, including port scanning monitoring, an intrusion system detection (IDS), among others. We also intend to use more sophisticated mechanisms for attack detection, since it was not the focus of this paper, in which a simpler approach was used. In particular, a future possibility would be the adoption of, for example, entropy-based techniques [14]. Besides that, the usage of other techniques for representing state is a topic to be investigated, as it is proposed by neural networks or Tile-Coding [1] [15]. Finally, it is feasible to collect other types of metrics to represent states. For instance, the server replicas could take into account not only the throughput of server access links, but also the CPU and memory usage of the replicas.

Acknowledgement

This work receives financial support from CNPq through research grant ref. 311088/2015-5 and 407899/2016-2.

References

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Massachusetts, EUA: MIT Press, 2 ed., 2012.

[2] M. M. B. A. A. Nunes, 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, pp. 1617–1634, Third 2014.

[3] 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.*, vol. 38, pp. 69–74, Mar. 2008.

[4] J. G. Herrera and J. F. Botero, "Resource Allocation in NFV: A Comprehensive Survey," *IEEE Transactions on Network and Service Management*, vol. PP, no. 99, pp. 1–1, 2016.

[5] ETSI, "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. V. 1, 22-24/10/2012," 2012.

[6] D. King, A. Farrel, and N. Georgalas, "The role of SDN and NFV for flexible optical networks: Current status, challenges and opportunities," in *2015 17th International Conference on Transparent Optical Networks (ICTON)*, pp. 1–6, July 2015.

[7] ETSI, "Network Functions Virtualisation (NFV): Network Operator Perspectives on Industry Progress. V. 3, 14-17/10/2014," 2014.

[8] K. Malialis, *Distributed Reinforcement Learning for Network Intrusion Response. University of York*. PhD thesis, United Kingdom, 9 2014.

[9] C. C. Machado, L. Z. Granville, and A. Schaeffer-Filho, "ANSwer: Combining NFV and SDN features for network resilience strategies," in *2016 IEEE Symposium on Computers and Communication (ISCC)*, pp. 391–396, June 2016.

[10] M. Belyaev and S. Gaivoronski, "Towards load balancing in sdn-networks during ddos-attacks," in *Science and Technology Conference (Modern Networking Technologies) (MoNeTeC), 2014 International*, pp. 1–6, Oct 2014.

[11] D. K. Y. Yau, J. C. S. Lui, F. Liang, and Y. Yam, "Defending Against Distributed Denial-of-service Attacks with Max-min Fair Server-centric Router Throttles," *IEEE/ACM Trans. Netw.*, vol. 13, pp. 29–42, Feb. 2005.

[12] Z. Hu and H. Chen, "Network load balancing strategy based on supervised reinforcement learning with shaping rewards," in *Intelligent Control and Information Processing (ICICIP), 2013 Fourth International Conference on*, pp. 393–397, June 2013.

[13] S. Kim, J. Son, A. Talukder, and C. S. Hong, "Congestion prevention mechanism based on Q-learning for efficient routing in SDN," in *2016 International Conference on Information Networking (ICOIN)*, pp. 124–128, Jan 2016.

[14] A. S. da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN," in *IEEE/IFIP Network Operations and Management Symposium*, pp. 27–35, IEEE, April 2016.

[15] A. Azzouni, R. Boutaba, and G. Pujolle, "NeuRoute: Predictive Dynamic Routing for Software-Defined Networks," *ArXiv e-prints*, Sept. 2017.