Anomalies detection method for non-determinist SDN control

Loïc Desgeorges, Jean-Philippe Georges, Thierry Divoux

Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 55-8 (2022) 57–63

Anomalies detection method for non-determinist SDN control

Loïc Desgeorges, Jean-Philippe Georges, Thierry Divoux

Université de Lorraine, CNRS, CRAN, F-54000 Nancy, France (e-mail: firstname.name@univ-lorraine.fr).

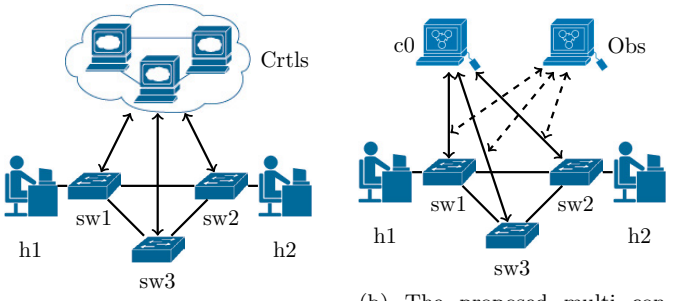
Abstract: Software Defined Networking (SDN) is a networking architecture within the control is centralized through a software-based controller. Thus, being a single point of attack makes it the preferred target in case of attack. Multi controller architecture has been considered to reinforce the control plane. However, the communication interface between the controller is a security threat. We already propose a dual controller architecture, one nominal controller which is in charge of the data plane computation plus a second one which is in charge of the detection of anomalies in the decisions taken by the first controller. Previous work considered a deterministic control and this paper extends to the case of a non-determinist algorithm. In this objective we introduce a multi-criteria detection approach and we developed two approaches: verifying the consistency of the performance of the decisions taken and verifying the consistency of the sequence of decisions of the controller. We tested the proposition on a study case.

Keywords: Software Defined Networking, Safety, Security, Multi-Controllers, Observability, Hidden Markov Models

1. INTRODUCTION

During the last decade, a huge activity concerns the Software-Defined Networking (SDN), presented by Farhady et al. (2015), architecture within the networking field. Fundamentally, SDN separates the control from the network infrastructure. This control is then centralized through a software-based controller, which takes all the decisions related to the network. Thus, from a security point of view, the controller is a preferential target as resumed in Scott-Hayward et al. (2013) Chica et al. (2020). In case of an attack of the controller, the attacker has access to the whole network and can damage for example thought a Denial of Service Attack by flooding the tables of the switches or by modifying the content of the commands sent by the controller as developed in Lee et al. (2017) or Alharbi et al. (2017).

The topic of this paper is the security of the network control using a multi-controller architecture. Multi controller is already widespread in the literature for several reasons as presented in Li et al. (2017) Zhang et al. (2018). One of these reasons concerns the safety aspect because, a failure of the controller inhibits the network service Vizarreta et al. (2017), a multi-controller architecture permits to provide a redundancy of the main controller like in the work of Fonseca et al. (2012). Also, the use of a set of controllers presents some benefits in terms of security. First, there is the possibility to set up a decision-making security architecture as in Qi et al. (2016): to determine if rules coming from one controller are valid, a vote is launched between all the other controllers which limits the influence of a controller attack. Also, more recently, Blockchain has also been considered as an option to secure the control layer and especially the communication interface between the controllers as in Derhab et al. (2021) or Yang et al. (2020). From another perspective, the use of the Moving Target Defense such as in Hyder and Ismail (2021) with the introduction of the notion of shadow controllers is also used to secure the control layer. Indeed, if malicious traffic is detected, a part of the shadow controllers are selected, randomly, as substitute of the infected controller.



(a) Classical mutli controller architecture

(b) The proposed multi controller architecture

Fig. 1. Differences between the classical multi controller architecture and our proposal

Regarding the related literature, it has to be noted that there is only multi controller approaches with East-West interfaces. These interfaces are a threat in terms of security, described by Kreutz et al. (2013) and some work proposed to secure them Shang et al. (2018). However, the decisions taken by one controller depend on the information given by another which might be malicious. That's why we proposed to develop a multi-controller architecture without communication between the controllers. The proposal consists of one observer which observes the main activity of the controller and detects anomalies through this activity. It has already been proposed in Desgeorges et al. (2021a) and Desgeorges et al. (2021b). However,

10.1016/j.ifacol.2022.08.010

these papers are based on the assumption that the behaviour of the controller is deterministic (such as routing using Dijkstra or Belleman-Ford algorithms). Thus, when a decision was sent on the network, the observer expected the same as the one observed previously. As a consequence, the proposed algorithms cannot be used to detect anomalies of non-determinist control (such as the one based on machine learning or greedy algorithms). The current work is an extension of the method by considering a non-deterministic control.

The first part aims to introduce the detection problem. Then an anomaly-based detection method is proposed in section 3. This proposal is illustrated and discussed on a case study in section 4 and finally, a conclusion presenting the perspectives concludes the work.

2. DETECTION PROBLEM

2.1 Recovering a forwarding plane from the capture

The OpenFlow packets (version 1.3, ONF (June, 2012)) exchanged, through the southbound interface, between the controller and the switches, are representative of the real behaviour of the controller. The whole activity (i.e. the trace Σ) of the control is defined hence accordingly as the set of packets:

$$\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}, \qquad \sigma_i = (t, \text{type}, \ldots)$$

and a packet, noted as a n-tuple, consists of the date of the capture t, the OpenFlow type and specific additional fields (related to the type) as described in the following.

Among all types of OpenFlow packets, the activity of the control is mainly determined according to four types: inputs (transfer the control of a packet to the controller), commands (send the packet out of a specified port, modify the state of the switch), status (changes of port/link status of a switch) and statistics (read-state information from a switch about flow, table, queue, etc.).

When a switch receives a message of a given flow for the first time, it sends a `Packet_In` message to the controller in order to know the action to be executed for this message. We note hence $\mathcal{I} \subset \Sigma$ the set of `Packet_In` packets such that:

$$\forall \sigma \in \mathcal{I},\ \sigma = (t, \texttt{"Packet\_In"}, \omega, \rho, s, d)$$

where ω corresponds to the switch identifier, ρ to the in-port of the switch, s and d the IP source and destination addresses of the packets.

The actions $\mathcal{A}$ (for instance delete the message or send it out of a specified port) can be sent through two types of commands. First, `Packet_Out` which is used only once (the switch does not retain the information and will have to ask again to the controller if a similar message is received).

$$\forall \sigma \in \mathcal{A}_p,\ \sigma = (t, \texttt{"Packet\_Out"}, \omega, \rho, s, d)$$

where ω is the identifier of the switch on which the action needs to be applied, ρ is the port(s) identifier(s) on which the message (from s to d) needs to be forwarded.

The second command is `Flow_Mod`, which is permanent (the switch adds this command to its flow table and will directly apply the action for any matching message).

$$\forall \sigma \in \mathcal{A}_f,\ \sigma = (t, \texttt{"Flow\_Mod"}, \omega, \rho, s, d, \delta, \text{type})$$

where ω, ρ, s and d are identical as for $\mathcal{A}_p$ and δ is the time to live of the action followed by the type of the action (i.e. add, modify or delete a rule).

The status packets consist of notifications from the switches about the state of their ports (packet `Port_Status`), the links or the switch itself. We focus here on the ports' status, such that we note $\mathcal{P}$ the set of

$$\forall \sigma \in \mathcal{P},\ \sigma = (t, \texttt{"Port\_Status"}, \omega, \rho, \text{type})$$

where the type corresponds to the reason for the packet (`add` to notify about a new port, `delete` and `modify` a change of the state of the given port ρ.

The last type of packets gathers the statistics sent by the switches (`"MultipartReply"`) in response to the request of the controller (through `"MultipartRequest"`). According to ONF (June, 2012) there are several kinds of statistics given by the switch and in this work, we will only consider the statistics related to the flow.

$$\forall \sigma \in \mathcal{S},\ \sigma = (t, \texttt{"MultipartReply"}, \omega, s, d, b)$$

such that b is the total number of bytes, regarding the flow from s to d, forwarded by the switch ω.

At the end, it means that the trace (i.e. the set of the captured packets) consists of the set of inputs, actions, status and statistics.

$$\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\} = \mathcal{I} \cup \mathcal{A} \cup \mathcal{P} \cup \mathcal{S}$$

A particular attention is paid on the actions since they are consecutive of the controller logic. A faulty or hacked controller will lead to missing or unexpected actions. The role of an observer will be hence to determine if the captured actions are, considering the context (the demands, the topology, the inputs, the status and the statistics) consistent with an unfaulty and non hacked control (and hence if the trace is consistent or malicious).

Such analysis depends on the control function. In this paper, we consider a routing algorithm. It means that the action consists of forwarding decisions (to which port(s) a switch need to forward a message?). Considering a demand from s to d, it means that the installed route consists of the actions defined by the controller and may be identified by the (surjective) function:

$$\mu(t, s, d) = \{\sigma \in \mathcal{A} \mid t_\sigma \in [t - \delta, t], s_\sigma = s, d_\sigma = d\}$$

where δ corresponds to the time to live of the actions.

Obviously (and as shown in the following), a valid route need to verify specific properties (like no loop) such that a first set of faults/attacks might be detected. However, such investigation needs to be enlarged to the whole set of routes defined by the controller, i.e. the forwarding plane. Considering the list of demands $\mathcal{D}$ for which the controller decided to install active routes at time t such that:

$$\mathcal{D}(t) = \{(s, d) \mid \exists \sigma \in \mathcal{A},\ t_\sigma \in [t - \delta, t],\ s_\sigma = s,\ d_\sigma = d\}$$

it is important to take into account the time to live of such actions (δ). Indeed, an action set up at $t - \delta - \epsilon$ is still valid at t.

It enables finally to define the active forwarding plane $\mathcal{P}$ (i.e. the set of active routes) at a given time t as:

$$\mathcal{P}(t) = \bigcup_{\forall (s,d) \in \mathcal{D}(t)}^{*} \mu(t, s, d)$$

where $\overset{*}{\bigcup}$ consists of a special union operator dealing with the different types of actions. If the type of the `Flow_Mod` is `add`, the (forwarding) action is directly added to the set; if the type is `delete`, the relevant previous action is removed from the set and if the type is `modify`, the relevant previous action in the set is substituted by the new rule.

In a way, this plane $\mathcal{P}$ can be seen as the internal state of the controller. The purpose of the observer consists therefore of determining online if it this state is coherent and representative of an unfaulty and unattack control.

2.2 Controller activity

A first detection of a faulty control is related to missing actions. In case of reactive routing, an input message (a `Packet_In` packet) has to be followed by at least one action (i.e. `Packet_Out` or `Flow_Mod` message). In case of proactive routing (for which all possible routes are installed whatever the demands), actions should be sent by the controller in a periodic scheme. More generally speaking, it means that it is necessary to check at any time if all demands are supported by actions. In other words, a plane $\mathcal{P}(t)$ has to address all real demands $\mathcal{D}'(t)$, i.e. $\mathcal{D}'(t) \in \mathcal{D}(t)$. If this condition would not be satisfied, we would detect a faulty control and generate an alarm.

To note also that if one considered that a faulty route is not a problem if it is not related to a real demand (or a very slight as determined thankfully the statistics messages), the plane $P(t)$ could be reduced to the list of the routes supporting the real demands. This could be particularly interesting in the case of a proactive routing (for which the routing algorithm will install routes from any node to any other), especially limiting the combinatorial explosion (for example, in the GÉANT network, there are more than 2^{506} possible data plane).

2.3 Routes coherence

Another necessary condition about the consistency of the control deals with the connectivity of the routes. To verify this criterion there are the three (structural properties) rules introduced in Desgeorges et al. (2021b). We note $\mathcal{G} = (\Omega, V)$: the topology composed of the set of switches, Ω and the links between the switches, V and we introduce the function $\nu := i \rightarrow \omega$ which returns the switch identifier ω on which a node i is connected and $\xi := (\omega, \rho) \rightarrow \omega'$ the function which returns the neighbour switch ω' along a given port ρ of a given switch ω. Hence, each route from s to d included in the plane at time t will be considered consistent with the topology only if the three following rules are verified.

(1) There is no loop:
$\forall \sigma \in \mu(t,s,d), \quad \nexists \sigma' \in \mu(t,s,d), \ \sigma' \neq \sigma \mid \omega_\sigma = \omega_{\sigma'}$
(2) There is no dead node:
$\forall \sigma \in \mu(t,s,d), \quad \exists \sigma' \in \mu(t,s,d) \mid \xi(\omega_\sigma, \rho_\sigma) = \omega_{\sigma'}$
(3) The destination is reached:
$\exists \sigma \in \mu(t,s,d) \mid \omega_\sigma = \nu(d)$

By extension, a plane is coherent only if all routes are coherent. However, since an attack may lead to consistent but non desired routes, the problem of the detection of a faulty/malicious controller is hence more complex.

2.4 Routes likelihood

The problem consists now to determine if the plane implemented by the controller is efficient and has not been modified (for instance due to an attack). For example, a plane in which all routes are visiting a given common switch could be the symptom of a man-in-the-middle attack. However, the likelihood of a route depends on the nature of the algorithm. For determinist routing algorithm, the answer might be easily obtained. For instance, Desgeorges et al. (2021b) showed the efficiency of a method checking that the data plane is not changing while the context remains the same (i.e.same demands, same topology). However, for non-determinist routing algorithms (like in Casas-Velasco et al. (2020)), different results are possible (even for the same context), such that we need to introduce new metrics.

More generally, the performance of the decisions of the controller might be considered. Indeed, according to the control and/or the considered network, a hypothesis could be added such that the evolution of some metrics might remain limited. However, this might not be sufficient. For instance, the delay performance can be degraded not due to a fault or an anomaly but due to the appearance of a large demand for example. An idea would then consist of comparing the plane to a set of previous observed and known planes as introduced in the following section.

3. ANOMALY-BASED DETECTION

We propose to compare the running behaviour of the control to the unfaulty commands observed. In particular, we propose to determine the likelihood of a data plane $\mathscr{L}(P)$ according to a multi-criteria approach:

$$\mathscr{L}(\mathcal{P}) = \Sigma_{i=1}^{n} \alpha_i \times p_i$$

with:

- $(\alpha_i)_{i\in[1,n]}$: the criteria weights such that: $\Sigma_{i=1}^{n}\alpha_i = 1$
- $(p_i)_{i\in[1,n]}$: the likelihood of the criteria i

Hence, a data plane implemented by the controller is assumed to be consistent iff its likelihood is lower than a threshold noted TD, i.e. $\mathscr{L}(P) < TD$.

Here, we propose two metrics:

(1) compare the impact of the plans by assessing the performance of each plan.
(2) compare the routes of the observed plans to the ones known as consistent.

such that it gives:

$$\mathscr{L}(\mathcal{P}) = \alpha_1 \times p_1 + \alpha_2 \times p_2$$

The aim of what follows is to introduce p_1 and p_2.

3.1 Criterion 1: planes performance

The first considered criteria concerns the performance. The evaluation of the performance of a decision depends on the considered metrics and the stability sought the control.

In this work, we propose to focus on the bandwidth offered compared to the demand throughput. For each flow (s,d), the difference between the traffic sent N_{Sent} and the

received one $N_{Received}$ is then computed. For a given flow (s, d), the plane efficiency is defined as follows:

$$m(\mathcal{P}, (s,d)) = \frac{N_{Sent} - N_{Received}}{N_{Sent}}$$

Obviously, such efficiency depends on the considered demand. Since a plane gathers all routes, we introduce a function φ which corresponds to the distance between two planes $\mathcal{P}$ and $\mathcal{P}'$ (whatever the demand) such that:

$$\varphi : (\mathcal{P}, \mathcal{P}') \mapsto \max_{\forall (s,d) \in D(t)} |m(\mathcal{P}, (s,d)) - m(\mathcal{P}', (s,d))|$$

This definition permits to measure the gap between the current plan $\mathcal{P}$ and a previous plan observed $\mathcal{P}'$ by considering the maximum impact observed. To note that it is possible to be more tolerant by considering the *min* function instead of the *max*.

As a consequence, the likelihood of the n-th data plane, $\mathcal{P}_n$ corresponds to the distance between $\mathcal{P}_n$ and the set of previous planes observed:

$$p_1 = \max_{i=0 \ldots n-1} \varphi(\mathcal{P}_n, \mathcal{P}_i)$$

Such equation means that a plane (in definitive the activity of the control) might be detected as faulty/attacked as long as it is different to at least one single previous observed plane. Obviously, this statement can be slightly changed if we considered the *min* operator (looking for a minimal difference along all previous observed planes) or even the median or the average (looking here for the disparity between the planes).

This criterion has, however, still some weaknesses. Indeed, in case of an attack that aims to degrade performance slowly, the disparity between the planes will remain low such that the threshold TD will not be reached and the fault/attack not detected. We then introduce a second criterion that will focus more on the routes diversity themselves.

3.2 Criterion 2: planes structures

In fact, the second criterion consists into analysing the routes. The proposal is to determine the likelihood of the observed sequence of decisions of the control W.

$$p_2 = \mathscr{L}(W) \tag{1}$$

There are several possibilities to determine $\mathscr{L}(W)$ (it depends on the considered routing algorithm). However, there is a constant: the decisions taken by the controller depend on its interns variables. Here, as there is no East-West interface between the controller and the observer, we do not have access to the evolution of these interns variables. We just have access to the observation of the activity of the controller (i.e. the OpenFlow messages), resulting itself of the evolution of the interns variables. Such evolution is represented in Fig. 2.

We assumed that the evolution follows a Markov Process. Thus, we propose to use the Hidden Markov Model (HMM) formalism introduced in Baum and Petrie (1966). This formalism has already been used in the context of the security of the SDN controller as in Wang et al. (2018). The objective is to re-estimate the interns variables of the controller based on the observation and to estimate the

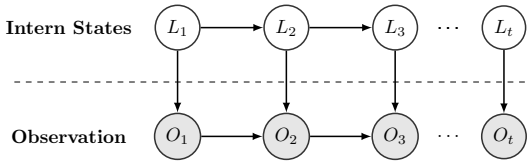


Fig. 2. Inference process

parameters of the model. To subsequently determine the likelihood of an observation by inference over the internal states, probabilistic theory such as the Bayes' theorem is used.

3.3 Definition of HMM

A Hidden Markov Models (HMM) is defined by 3-tuple HMM $= (\pi, A, B)$ defined as follows:

- N: the number of states
- $S = \{s_1, \ldots s_N\}$: the set of states.
- M: the number of observations
- $O = \{o_1, \ldots o_N\}$: the set of observations.
- $A \in M_{N,N}$: a transition probability matrix A
  $A = (a_{i,j} = p(q_t = s_j | q_{t-1} = s_i)$: represents the probability of moving from state s_i to state s_j
  $\forall i : \Sigma_{j=1}^{N} a_{i,j} = 1$
- $B \in M_{N,M}$: observation probability matrix,
  $B = (b_{i,j} = p(o = o_i | s = s_j))$: each expressing the probability of an observation o_i being generated from a state s_j
  $\forall i : \Sigma_{j=1}^{N} b_{i,j} = 1$
- $\pi \in M_{1,N}$: an initial probability distribution over states

Here, the HMM is designed from the observer point of view and so the set of observations $W_{Obs} \in O^n$ corresponds to a sequence of events of the activity of the control. HMM introduced then three problems presented in Rabiner (1989) and resumed as follows:

(1) Given a HMM $\lambda = (\pi, A, B)$ and an observation sequence W_{Obs}, determine the likelihood $P(W_{Obs}|\lambda)$.
(2) Given an observation sequence W_{Obs} and an HMM $\lambda = (\pi, A, B)$, determine the best hidden state sequence.
  $Q^* = argmax_Q p(Q|W_{Obs}, \lambda)$
(3) Given an observation sequence W_{Obs}, the set of states S, determine the HMM parameters π, A and B.
  $\lambda^* = argmax_\lambda p(W_{Obs}|\lambda)$

In the following, both criteria are analysed on real experiments and the limits are discussed.

4. CASE STUDY

4.1 Scenario

As a case study, the proposed observer is applied for the network controller introduced in Casas-Velasco et al. (2020). Here, the control function is a multi-objective proactive routing through reinforcement learning technique. The metrics used are delay, packet loss and the available link bandwidth. The topology consists of 23 nodes and 37 links as shown in Fig. 3. To tackle Mininet limitations, we scaled the 10 Gpbs, 2.5 Gpbs, and 155

Mbps link capacities of GÉANT to 100 Mbps, 25 Mbps, and 1.55 Mbps, respectively.

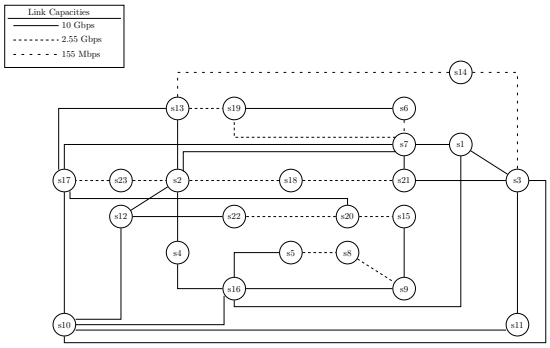


Fig. 3. Topology of the network GÉANT with 23 nodes.

The considered traffic is the dataset TOTEM (January, 2006) which provide intra-domain traffic matrices for the GÉANT topology. The traffic load was also scaled to the same proportion. This control is proactive, which means that periodically a new data plane is set up. The code of the controller is given in Casas-Velasco (2020) and is implemented through a Ryu controller Team (2012).

The considered attacks lead to a degradation of the service without sending any traffic on the network. Indeed, the first step is to take control of the controller using Metasploit, a library of Kali Linux, and then to modify the result of the control algorithm in order to redirect the traffic through the link with the less capacity. This corresponds to a `Flow_Mod` modification as presented in Lee et al. (2017).

4.2 Focus on the performance: $\alpha_1 = 1$ *and* $\alpha_2 = 0$

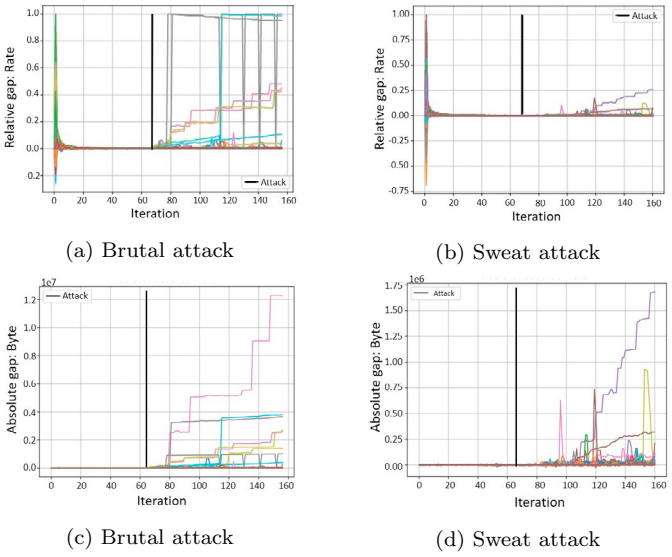
In this part we assume that $\alpha_1 = 1$ and $\alpha_2 = 0$ which implies that $\mathscr{L}(\mathcal{P}) = p_1$ and so the computation of the likelihood is based only on the first criteria previously defined: the evolution of the performance of the decisions of the controller. The analysis is achieved according to two cases.

Particular case. Two attacks are tested: the first one is to change all the routes in the same time to have a brutal effect and the second is to change one single route at a time to observe a slow effect as in low and slow DDoS attack. Fig. 4a and Fig. 4b deal with $m(\mathcal{P}, (s, d))$ while Fig. 4c and Fig. 4d show $|N_{Send} - N_{Received}|$ for each flow.

It can be observed on Fig. 4a and Fig. 4b that there is packet loss at the beginning. This is due to the fact that initially there are some packets which are sent but not yet arrived. During the 70th first iterations, there is no modification of the routes, the attack begins at the 71st iteration. The attack impact is visible around the 80th iteration for the brutal attack by a significant rise loss while for the derived of the sweat attack, it stays relatively constant.

The evolution of the criterion p_1 is finally given in Fig. 5.

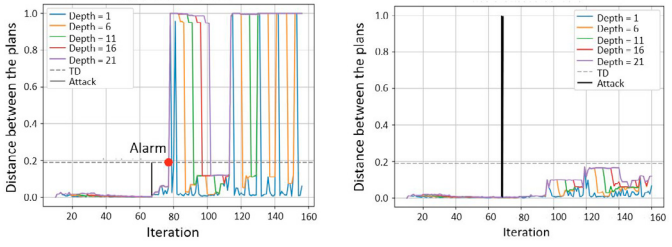
In case of a sweat attack, the criterion does not evolve significantly and stay constant under the threshold and no alarm is raised. However, in case of brutal attack, an alarm is raised after the 20 measures of the statistics



(a) Brutal attack (b) Sweat attack

(c) Brutal attack (d) Sweat attack

Fig. 4. Impact of the attacks



(a) Brutal attack (b) Sweat attack

Fig. 5. p_1 evolution in case of the particular case.

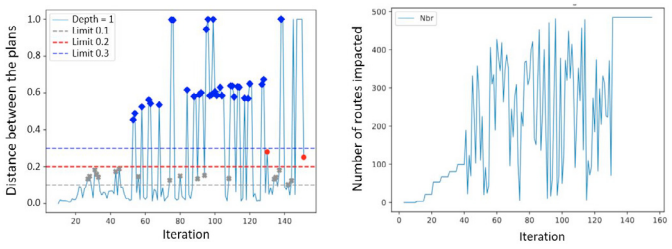
($\sim$ 400 seconds). All depends on the tolerance fixed but considering only the performance is sensitive to the case of a sweat attack.

To note also that it is possible to change the *depth* of the previous observed planes in order to speed up the calculations by updating the criterion calculation by:

$$p_1 = \max_{i=n-depth\ldots n-1} \varphi(\mathcal{P}_n, \mathcal{P}_i)$$

Fig. 5 highlights that increasing the value of *depth* lead to raise more alarms since it can be observed that for $i < j$ the criterion for $depth = i$ is lower than for $depth = j$.

General Case. A random distribution is now considered to determine the routes to change. The results are given in Fig. 6 where the dots stand for the alarm rising.



(a) Distance (b) Number of evolution

Fig. 6. p_1 evolution in case of a random routes attack.

The first graph shows that the criterion between the planes while the second represents the number of routes'

changes. Three thresholds ($TD = 0.1$, 0.2 and 0.3) are considered. Regarding the threshold $TD = 0.1$, there are obviously more alarms than the other thresholds. However, considering a higher threshold has an impact over the reactivity. For $TD = 0.3$, the first alarms are raised after 60 iterations which means that during all the time long the service is damaged. A worst case would have been such as in Fig. 5b, where no attack is detected. Thus, the threshold represents a compromised between the reactivity and the accuracy of the detection.

4.3 Focus on the planes structures: $\alpha_1 = 0$ and $\alpha_2 = 1$

In this part we assume that $\alpha_1 = 0$ and $\alpha_2 = 1$ which implies that $\mathscr{L}(\mathcal{P}) = p_2$ and so the computation of the likelihood is based only on the second criteria, i.e. the likelihood of the sequence observed.

Here the analysis of the routes is based on the HMM defined in section 3. First, let us describe the learning phase. We consider a sequence of 1000 planes randomly selected and we used the Baum-Welch algorithm to determine the parameter of the HMM. The number of internal states is fixed to $N = 3$. Secondly, the HMM has been launched over nominal and attacked sequences, both based on 125 planes.

Since the efficiency of the HMM may depend on the length of the observation sequence (noted *depth*), we considered four depth values. To support the comparison, we then introduce the normalized likelihood as follows.

$$P_N(O|\lambda) = \frac{P(O|\lambda)}{max_{Obs} p(Obs|\lambda)} \tag{2}$$

The threshold TD has been determined regarding the likelihood worst case L_{WC} observed experimentally during the learning phase such that $TD = L_{WC} - 1$. Values of TD and *depth* are reported in the table 1.

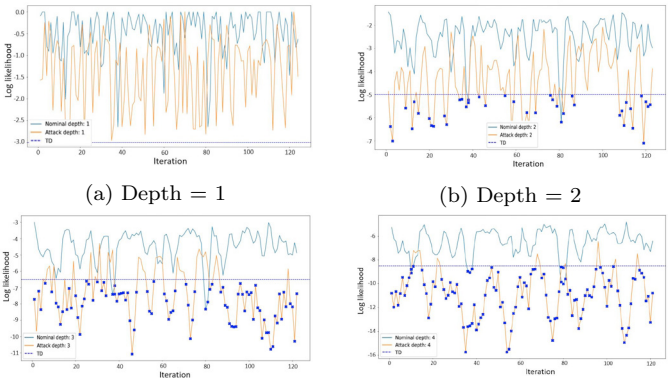
Table 1. The used thresholds

Depth	TD
1	-3
2	-5
3	-6.5
4	-8.5

Fig. 7 shows the normalized log-likelihood depending on the *depth* and the type of sequence (nominal or attacked).

If $depth = 1$, only the likelihood of the observed plan (whatever the sequence) is evaluated. Thus, each plane is considered consistent as far as we have already observed them. Indeed, even if it is rare, it is possible. This means that this is not an issue to observe it again and no alarms are raised.

Considering higher *depth* values makes the analysis more precise. Even if the observed plan is consistent and one of the most frequently seen, the sequence does not correspond to the distribution expected. And so, the likelihood decreases significantly when we consider at least 2 observations in the sequence.



(a) Depth = 1 (b) Depth = 2

(c) Depth = 3 (d) Depth = 4

Fig. 7. Log-likelihood of a nominal sequence compared to an attacking one.

It has to be mentioned that the complexity of the algorithm used to determine the likelihood (named forward algorithm) is $O(N^T)$ with N the number of states and T the depth of the sequence. As a consequence, $depth = 2$ seems to be the best compromise.

4.4 Discussion

In this paper, we introduced only two weights' configurations: either the performance or the planes' structures. Obviously, mixed cases for which both criteria are considered may improve the efficiency of the observer. Fixing the weights in such case will depend on the network topology, the demands' evolution and the network controller function. Indeed, for the second criterion, a network control function generating a large set of solutions will lead to a combinatory explosion and difficulty for the HMM to determine if the plane is representative of a consistent control or not. To note that, however, if the solutions have similar performances, the first criterion will not be able to detect attacks. On the other hand, the first criterion might not be sensitive to a sweat attack as mentioned early. Finally, if planes remains homogeneous in terms of routes but subjects to specific patterns (like typical man-in-the-middle attack), it would be only detected only by the first criterion (longer paths with more congestion). To resume, the final determination of the weights would depend on the hypothesis regarding the type of network, the control function stability and the attack variety.

5. CONCLUSION

In conclusion, this work consists into an alternative solution to detect threats on the control plane in a SDN architecture. It relies on an observer checking the activity of the controller. Also, we assumed that there is not communication between the observer and the main controller to avoid the threats of the East-West interface. This paper introduces an observer logic to detect anomalies in case of a non-deterministic network control function. We proposed a two criteria approach to analyse the consistency of the control: focusing on the performance of the decisions or on the decisions itself. Experiments showed the efficiency of the proposed method and the limits of each criterion.

Also, as soon as an attack or a failure is detected by the observer, it has to take the lead over the main

controller in order to assure a correct performance in the control. Hence, techniques to take the lead will be further applied in future works. Moreover, we aim to extend the detection algorithm. Firstly, by comparing and evaluating the performance of HMM to other well-known machine learning techniques as Support Vector Machine or k-Nearest Neighbours in terms of precision, recall and accuracy. Also, new techniques to detect anomalies in the context of an encrypted communication between the controller and the switches is part of the perspectives.

ACKNOWLEDGEMENTS

This work was supported partly by the French PIA project "Lorraine Université d'Excellence", reference ANR-15-IDEX-04-LUE.

REFERENCES

Alharbi, T., Layeghy, S., and Portmann, M. (2017). Experimental evaluation of the impact of dos attacks in sdn. In *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, 1–6. IEEE.

Baum, L.E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6), 1554–1563.

Casas-Velasco, D.M. (2020). *https://github.com/danielaCasasv/RSIR-Reinforcement-Learning-and-SDN-Intelligent-Routing.git , last visited the 10/01/2022.*

Casas-Velasco, D.M., Rendon, O.M.C., and da Fonseca, N.L. (2020). Intelligent routing based on reinforcement learning for software-defined networking. *IEEE Transactions on Network and Service Management*, 18(1), 870–881.

Chica, J.C.C., Imbachi, J.C., and Vega, J.F.B. (2020). Security in sdn: A comprehensive survey. *Journal of Network and Computer Applications*, 159, 102595.

Derhab, A., Guerroumi, M., Belaoued, M., and Cheikhrouhou, O. (2021). Bmc-sdn: blockchain-based multicontroller architecture for secure software-defined networks. *Wireless Communications and Mobile Computing*, 2021.

Desgeorges, L., Georges, J.P., and Divoux, T. (2021a). Detection of security and safety threats related to the control of a sdn architecture. In *4th IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control, CESCIT 2021.*

Desgeorges, L., Georges, J.P., and Divoux, T. (2021b). A technique to monitor threats in sdn data plane computation. In *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, 1–6. IEEE.

Farhady, H., Lee, H., and Nakao, A. (2015). Software-defined networking: A survey. *Computer Networks*, 81, 79–95.

Fonseca, P., Bennesby, R., Mota, E., and Passito, A. (2012). A replication component for resilient openflow-based networking. In *2012 IEEE Network operations and management symposium*, 933–939. IEEE.

Hyder, M.F. and Ismail, M.A. (2021). Securing control and data planes from reconnaissance attacks using distributed shadow controllers, reactive and proactive approaches. *IEEE Access*, 9, 21881–21894.

Kreutz, D., Ramos, F.M., and Verissimo, P. (2013). Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 55–60.

Lee, S., Yoon, C., Lee, C., Shin, S., Yegneswaran, V., and Porras, P.A. (2017). Delta: A security assessment framework for software-defined networks. In *NDSS*.

Li, D., Wang, S., Zhu, K., and Xia, S. (2017). A survey of network update in sdn. *Frontiers of Computer Science*, 11(1), 4–12.

ONF (June, 2012). *OpenFlow Specification v1.3 https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf, last visited the 14/09/2021.*

Qi, C., Wu, J., Hu, H., Cheng, G., Liu, W., Ai, J., and Yang, C. (2016). An intensive security architecture with multi-controller for sdn. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 401–402. IEEE.

Rabiner, L.R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 257–286.

Scott-Hayward, S., O'Callaghan, G., and Sezer, S. (2013). Sdn security: A survey. In *2013 IEEE SDN For Future Networks and Services (SDN4FNS)*, 1–7. IEEE.

Shang, F., Li, Y., Fu, Q., Wang, W., Feng, J., and He, L. (2018). Distributed controllers multi-granularity security communication mechanism for software-defined networking. *Computers & Electrical Engineering*, 66, 388–406.

Team, R.P. (2012). *"Ryu application API," Available: https://ryu.readthedocs.io/en/latest/ryu_app_api.html , last visited the 10/01/2022.*

TOTEM (January, 2006). *https://totem.info.ucl.ac.be/index.html , last visited the 10/01/2022.*

Vizarreta, P., Heegaard, P., Helvik, B., Kellerer, W., and Machuca, C.M. (2017). Characterization of failure dynamics in sdn controllers. In *2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM)*, 1–7. IEEE.

Wang, W., Ke, X., and Wang, L. (2018). A hmm-r approach to detect l-ddos attack adaptively on sdn controller. *Future Internet*, 10(9), 83.

Yang, H., Liang, Y., Yuan, J., Yao, Q., Yu, A., and Zhang, J. (2020). Distributed blockchain-based trusted multidomain collaboration for mobile edge computing in 5g and beyond. *IEEE Transactions on Industrial Informatics*, 16(11), 7094–7104.

Zhang, Y., Cui, L., Wang, W., and Zhang, Y. (2018). A survey on software defined networking with multiple controllers. *Journal of Network and Computer Applications*, 103, 101–118.