A technique to monitor threats in SDN data plane computation

Loïc Desgeorges, Jean-Philippe Georges, Thierry Divoux

A technique to monitor threats in SDN data plane computation

Loïc Desgeorges, Jean-Philippe Georges and Thierry Divoux
Université de Lorraine, CNRS, CRAN, F-54000 Nancy, France
firstname.name@univ-lorraine.fr

***Abstract*—Software Defined Networking (SDN) is a networking paradigm which proposed to decouple the forwarding and the control planes. Security and safety threat challenges at the control level are divided into the reinforcement of the controller, whatever the reason. This work aims to consider both threats and pave the way for a multi-controller architecture without East-West interface. Considering one nominal controller in charge of the data plane computation, we designed a second one in order to control the consistency of the decisions made by the controller, i.e. only through observing the activity of the command (i.e. the management traffic). Compared to related works, no direct exchanges between the controllers are required. The detection logic is introduced theoretically and it mainly relies on two phases: the learning of the decisions and the verification that each decision taken fits with the data plane estimate. The algorithm, implemented on ONOS, is discussed in a case study.**

***Index Terms*—Software-Defined Networking (SDN), Safety, Security, Multi-Controllers, Observability**

I. Introduction

Software-Defined Networking (SDN), [1], has been introduced to provide a structured software environment to deal with various application requirements and dynamic networks [2]. It provides an architecture within the infrastructure are separated from the control part in a centralized control architecture manner. SDN simplifies network management and facilitates network evolution. However, such centralization introduced two main issues: scalability and robustness. To overcome it, a distributed control has been introduced. A multi-controller architecture permits to balance the load between the controllers while it provides an active redundancy [3]. On top of that, it has some challenges in terms of consistency, reliability, load balancing and security as developed in [4].

Indeed, each plane of the SDN architecture has its own weakness [5]. Moreover, due to the global view of the network, a security or safety threat of the control plane has consequences on the entire network.

Classically, the security challenge is resolved by making the controller more robust, as FortNOX [6], while the safety challenge is resolved by the consideration of multi-controller architectures, as in [7]. However, the multi-controller architecture might also be used for security reasons. In such a case, the second controller has a particular task related to the security. As an example, [8] proposed a decision-making security architecture within each controller submits flow rules to a vote between all the others before sending it. In the same idea, [9] proposed to use a second controller as a filter in order to validate the command sent by the nominal controller. The underlying issue is then that such related works need East-West communications in order to assure consistence between the controllers information. This interface might become another primary concern. If solutions like encryption as presented in [10] or advanced authentication as in [11] have been proposed, there is still a channel of communication between the controllers. In this paper, a novel detection algorithm is then designed only based on the messages sent/received by the controller (and not anymore based on its internal states). The objective is to be robust to false messages sent by the controller (i.e. in case of attacked controller). Furthermore, the proposed algorithm aims at considering simultaneously security and safety threats.

The paper focusses on the data plane computation case. In this objective, a multi-controller architecture without East-West interface is introduced within one controller is in charge of the data plane computation while the second monitors the activity of the control with the sole purpose of verifying if anomalies exist concerning the decisions of the main controller. A motivated example is presented in section II. The architecture and the detection algorithm proposed are introduced in section III and IV. Use cases are then developed in section V and section VI concludes the work.

II. Problem Statement

A. Motivating Example

A controller with a deterministic routing application is considered. Here, the controller ONOS [12] is considered. The controller is loaded with applications for the protocol of communication with the switches, Openflow, for the topology discovery, host provider and LLDP, and proxyARP. Moreover, ONOS used an Intent Framework which is a subsystem that allows applications to specify their network control desires in the form of policy rather than mechanisms. And these policy-based directives are referred as intents. Here, the intent's routing will be added by hand.

The topology in Fig. 1 is considered. In what follows, scenarios of threats of the command are presented.

Attack Scenario: There are several security threats on SDN architecture as described in [13]. In this work, attention is paid to the command and only attacks which impacted the decisions are considered. It corresponds as an example to the manipulation by a malicious application of the flow table in the switch. As an example, let's considered the internal storage

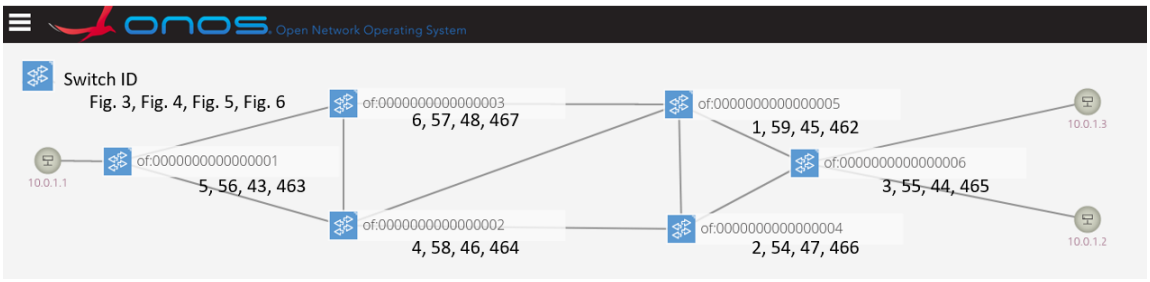

Fig. 1. The studied topology

abuse as described in [14]. A malicious application may access and alter network topology data within the internal storage of the controller. As a consequence all peer applications which use the topology information to derive flows and install a path over the network are impacted. This modification might be used in order to violate security policy install by other applications as developed in [6].

As an example, let us consider an internal storage misuse attack on the ONOS controller of the topology of Fig. 1. Here, a malicious application modifies the network topology data by exchanging the position of host 10.0.1.2 and host 10.0.1.3 in the controller topology storage. In this case, if a host tries to join host 10.0.1.3 then the data plane compute by the controller implies the installation of rules in order to reach 10.0.1.2 and not 10.0.1.3. Also, it is possible to modify the code of the controller and modify the command in the direction of one particular switch, $of6$ here.

Failure Scenario: Obviously, a second threat of the network is a failure of the controller. Such a case would impact the network as no decision will be taken any more. It is also to consider other safety issues of the controller such as an undiagnostiqued port's failure which leads to the interruption of communication with a switch.

B. Modelling of the management traffic

As mentioned, we are considering the computation of the data plane by the controller. In this section, the properties of the transmission of the data plane are developed.

1) Set of exchanges: The packets exchanged for the data plane computation are sent through the Southbound interface, which is the interface between the switches and the controller. This interface is normalized by Openflow [15].

The messages considered at the Southbound interface are: the requests from the switches, the commands for the switches and the port status from the switches: $\Sigma = \Sigma_{In} \cup \Sigma_{Out} \cup \Sigma_{Ps}$ The first set, Σ_{In}, corresponds to the "Packet_In" messages, named pin, which are the packets received by the controller. $\forall pin \in \Sigma_{In}$ there is $pin = (Swp, b, src, dest)$ with:

- $Swp = (p, S) \in \mathbb{N} \times \mathbb{N}$: the in-port p of the switch S.
- $b \in \mathbb{N}$: an identifier named Buffer IDentifier which is tagged to the original packet by the switch.
- src: the IP source address of the packet.
- $dest$: the IP destination address of the packet.

The second type of events is related to the commands sent by the controller. There are two types of commands. First, "Packet_Out", noted $pout$, which is used only once. (the switch does not retain the information and will have to ask again to the controller what to do.) Secondly, "Flow_Mod", noted $fmod$, which is permanent: the switch adds this command to its flow table. Thus: $\Sigma_{Out} = \Sigma_{PO} \cup \Sigma_{FMOD}$ with $\forall pout \in \Sigma_{PO}$, $pout = (act, b, S)$:

- $act \in Actions$: the action ordered defined as following.
- $b \in \mathbb{N}$: the buffer ID of the packet.
- $S \in \mathbb{N}$: the switch destination of the command.

$\forall fmod \in \Sigma_{FMOD}$, $fmod = (act, b, S, src, dest, idle, type)$:

- act, b and S are similar to $pout$.
- src: the IP source address of the packet.
- dst: the IP destination address of the packet.
- $idle \in \mathbb{R}^+$: the storage time of the order by the switch.
- $type \in Add, Delete, Modify$: the type of the instruction.

As we consider only the computation of the data plane, an action can be modelled as a vector within each component is associated with a port of the switch, representing the decision issued by the controller to transmit or not. Moreover, the controller might delete a line of the switch's flow table in case of a data plane evolution and such action will be formalized as a -1 for the corresponding port.

$$Actions = \left(\left\{ \prod_{j=1}^{N} b_j \quad (b_j) \in \{-1, 0, 1\} \right\} \right)$$

Finally, "Port_Status", noted ps, is a notification from the switches about the state of their ports. Then $\forall ps \in \Sigma_{Ps}$, $ps = (reason, p, S)$:

- $reason \in Add, Delete, Modify$: the reason of the message: Add to notify the port was added, $Delete$ if the port was removed and $Modify$ for a modification of the port state.
- p: the considered port.
- S: the switch source of the packet.

2) Path Properties: When a path is set up by the controller, though several packets described just above, the observer will have to verify its consistency according to three criteria defined below. For the analysis of these criteria, the graph topology of the switches infrastructure is needed and noted $\mathcal{G}$ A path, installed by a set of command $(pout_i)_{i \in [1,n]}$, is considered consistent if:

- There is no loop:
  $\forall i \in [1,n] \nexists j \ in[1,n], i \neq j | pout_i[3] = pout_j[3]$
- There is no dead node:
  $\forall i \in [1,n] \exists j \ in[1,n], i \neq j | T(pout_i[3], pout_i[1]) = pout_j[3]$
- The destination is reached:
  $\exists i \in [1,n] \mathcal{G}(pout_i[3], pout_i[1]) = pin[4] \&$
  $\nexists j \in [1,n], i \neq j | pout_i[3] = pout_j[3]$

3) Impacts of the threats: In case of an attack or a failure, the control algorithm returns a biased command. The threat may have several origins in the SDN architecture as explained in [13] but in this work we do not consider the isolation of the fault. Thus, we proposed to synthesize these different threats in one bias, named b_{Cmd}, which leads to an affine biased Packet_Out $pout' \in \Sigma_{Out}$ defined as:

$$pout' = pout + b_{Cmd}$$

With $pout' \in \Sigma_{Out}$: the biased packet; $pout \in \Sigma_{Out}$: the original packet; $b_{Cmd} \in \Sigma_{Out}$: the bias; and $+$ the operator defined as:

$$\forall i \in [1, lengh(pout)] \ pout'[i] = pout[i] + b_{Cmd}[i]$$

4) Temporal notion: A particular case of the bias is the disappearance of the command packets. Which means that there is no command sent by the controller. It corresponds mainly to the case of failure. We choose to use the Student's distribution to statistically learn the interval of time it within the command is expected. Indeed, let us suppose that we have observed k commands. Then, the interval of confidence at the precision $1 - \alpha$ is:

$$it = [t_{Moy} - \Delta t_{Moy}, t_{Moy} + \Delta t_{Moy}] \quad (1)$$

with $t_{Moy} = \frac{\Sigma_{i=1}^{k} t_i}{k}$: the average time; $\sigma = \sqrt{\frac{\Sigma_{i=1}^{k}(t_i - t_{Moy})^2}{k-1}}$: the standard deviation: $\Delta t_{Moy} = t_{\alpha/2}^{k-1} \times \frac{\sigma}{\sqrt{k}}$: expanded uncertainty and $t_{\alpha/2}^{k-1}$: a parameter which permits to fix the confidence of the interval. This parameter depends on the number of observations and the confidence expected.

III. Architecture Proposed

The aim of this work is to propose a detection approach of the bias defined just above.

A. Multi-Controller Architecture

To consider safety threats of the controller, a redundancy is necessary. That's why, a second controller is introduced. Moreover, a security threat might spread over the interface of communication, named East-West, between the controllers. Thus, we proposed to introduce the architecture represented in Fig. 2. There is one controller in charge of the network while the second is an observer which has to detect safety or security threats.

As a consequence of the absence of East-West interface, the detection method will not be based on the internal states we get from the other controller but on the observation of its activity on the network and some *a priori* knowledge of the control logic. Additionally, the topology discovery application also runs on the observer which permits to determine $\mathcal{G}$ the graph of the topology of the infrastructure.

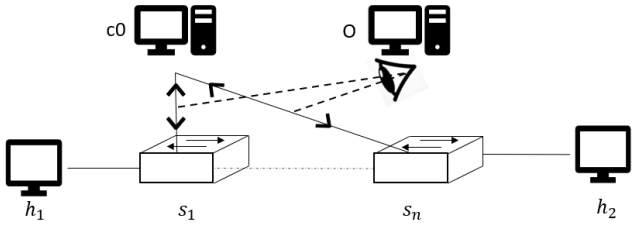

Fig. 2. Architecture proposed ($c0$: nominal controller, O: the observer, h_1 and h_2: two hosts and s_1 and s_n are the switches)

B. Detection principle

To develop the detection logic, Intrusion Detection System (IDS) theory has been considered. According to [16], IDS proposals might be divided into two categories: focusing on the attack behaviour or on the unfaulty behaviour of the system. The first approach is based on the attack signature. This implies to consider particular attacks and detects only the one considered.

The second one is divided into Anomaly and Specification-based approaches. They differ in their knowledge base of the system behaviour but both aim to compare the unfaulty behaviour known to the running behaviour. Basically, anomaly detection techniques are based on a model of the unfaulty behaviour of the system while specification-based is based on a specification directly from documentation.

These two techniques might be combined as proposed in [17]. There are two steps: a specification is determined off-line and then the system is observed on-line to determine a model of its recurrent behaviour and learn some statistical properties linked to the specification model. In this work, a similar approach is introduced. The specification formalism chosen is a template, inspired by [18], which expresses the causality between the requests and the commands. This specification evolves according to intern variables of the controller which are estimated by the observation of the activity of the command.

Formally, let us consider $\mathcal{T}$ the set of templates and a template $temp \in \mathcal{T}$ defined as:

- $temp = \{pin, \{(pout_i)_{i \in [1,n]}, it\}\}$
- $pin \in \Sigma_{In}$: the triggered event. It might be empty in case of spontaneous command.
- $(pout_i)_{i \in [1,n]} \in \Sigma_{Out}^n$: a set of commands in reaction to the event pin. This set of command is expected to be consistent according to section II-B2.
- $it \in \mathbb{R}^2$: the interval of time within the data plane is expected.

More specifically, a template $temp$ specifies that after the capture of a request from a switch pin, the observer expects a set of commands initializing the route over the network $(pout_i)_{i \in [1,n]}$. Moreover, these commands are expected to be set in an interval of time it.

IV. Detection logic

The observer is supposed to declare a fault regarding the decisions of the controller if they are not consistent regarding the constraints defined in section II-B2 and II-B4 or they are subjected to unexpected change (as we considered deterministic algorithm). The logic of the observer is to verify these two criteria. Thus, there is a learning phase in order to store the consistent path observed. The route will be stored, on the form of a template, in the set $\mathcal{R} = \mathcal{T}^n$.

The detection logic is given in the Algorithm 1. The algorithm is executed each time an OpenFlow packet is captured by the observer on the communication network between the controller and the switches. At the observation of a Packet_In, first there is the learning phase which starts line 3 within the data plane set is stored until the end of the timeout, defined in section II-B4, or that the path is assumed to be consistent, according to section II-B2. To simplify, the three constraints are resumed in one boolean variable $cons(r)$ which means that if r satisfied the three constraints then $cons(r) = true$ and else $false$. Otherwise, if the route asked in request is part of $\mathcal{R}$ then the observer verifies that the controller sets the similar route, line 15.

However, as soon as a link evolution between switches is notified, by a Port_Status, the data plane computation evolves. Indeed, the controller uninstalls the previous paths concerned in ordered to install a new one. This implies that the observer has to restart a learning phase. Thus, if there are paths impacted, the observer lets to the controller the time to install a new path. The commands might be the deletion of previous rules, line 29 or the installation of a new one, line 31 or the modification of a previous one, line 33. Anyway, at the end of the timeout, the consistency of the route set up through the observed commands is checked, line 35.

V. Illustration

In this section the detection algorithm is put into practice and discussed. The controller environment is the same as presented in section II-A.

A. Scenario

The network environment for the use case is the same as in section II-A. The topology is represented in Fig. 1 is considered and the traffic applied is an Iperf from host 10.0.1.1 to host 10.0.1.2. The algorithm's template $Temp$ is $\forall pin \in \Sigma_{In} \forall (pout_i)_{i\in[1,n]} \in \Sigma_{PO}^n \; Temp = \{pin, \{((pout_i)_{i\in[1,n]}, it)\}\}$.

The interval it is determined statistically (i.e. through the Student Law) as presented in section II-B4. In fact, the parameters of the model are chosen in order to have $(1-\alpha) = 99\%$ of confidence in the intervals. There are two different intervals, one for a request which is a Packet_In it_{Pin} and another in response to a Port_Status it_{Ps}. Among the parameters of the model, such confidence level is obtained for a value of $k = 10$ time measurements and for a quantile of $t_{\alpha/2}^{k-1} = 2.764$. To note that the time measurements are related to the Iperf probes and that adapting k and $t_{\alpha/2}^{k-1}$ permit to be flexible regarding the reactivity and the precision of the observer. The board of the parameter values for the determination of it are given in (1). In conclusion, the interval of time is $it_{Pin} = [1.879, 8.324]$ and $it_{Ps} = [0, 0.1842]$.

The attack considered is the same as in II-A. Such an attack is then modelled by a bias b_{Cmd} of a command $pout \in \Sigma_{Out}$

Algorithm 1 Observer logic

Input: An OpenFlow packet p.
Data: $\mathcal{R}$: the set of routes and $\mathcal{G}$ the graph of the topology.

```
1  p = wait(packet)
2  if p ∈ Σ_In then
3      if p ∉ R then
4          r = ∅
5          while ¬cons(r) & timeout do
6              f = wait(fmod)
7              r = r ∪ (p, fmod)
8          if cons(r) then
9              R = R ∪ r
10         else
11             return Fault
12     else
13         r_p = r | r ∈ R & pin(r) = p)
14         r'_p = r_p
15         while path(r'_p) ≠ ∅ & timeout do
16             f = wait(fmod)
17             if f ∉ path(r'_p) then
18                 return Fault
19             else
20                 path(r'_p) = path(r'_p) \ fmod
21         if path(r'_p) ≠ ∅ then
22             return Fault
23 else if p ∈ Σ_Ps then
24     for r_p ∈ R | ∃fmod ∈ cmd(r_p) | port(fmod) = port(p)
       do
25         r'_p = r_p
26         while timeout do
27             f = wait(fmod)
28             f' ∈ cmd(r'_p) | src(f') = src(f), dst(f') =
               dst(f), sw(f') = sw(f)
29             if type(f) = delete then
30                 r'_p = r'_p \ f'
31             else if type(f) = add then
32                 r'_p = r'_p ∪ f
33             else if type(f) = modify then
34                 r'_p = r'_p \ f' & r'_p = r'_p ∪ f
35         if ¬cons(r'_p) || r'_p = r_p then
36             return Fault
37         else
38             R = R \ r_p & R = R ∪ r'_p
```

which leads to $pout' \in \Sigma_{Out}$ such as:

$\forall i \in [1, lengh(pout)]$ $pout'[i] = pout[i]$

$pout'[1] = 10$ if $pout[3] = of6$, $pout'[1] = pout[1]$ else

All traffic which passes through the switch 6 is retransmitted on the port 10. The aim of what follows is to show how this anomaly is detected. First, the learning phase is developed and then the running state is presented.

B. Learning Phase

This phase corresponds to the observation of requests without *a priori* information about the controller. The related frames, captured using Wireshark, are represented in Fig. 3. It is important to note here that for the detection, the observer remains silent and do not generate additional traffic.

Time	No.	Switch	Action	Type
3.1944…	464	5		OFPT_PACKET_IN
3.2048…	466	3	3	OFPT_PACKET_OUT
3.2059…	468	3		OFPT_PACKET_IN
3.2105…	469	5	1	OFPT_PACKET_OUT
10.387…	854	2	4,1	OFPT_FLOW_MOD
10.387…	855	3	3,2	OFPT_FLOW_MOD
10.387…	856	5	1,2	OFPT_FLOW_MOD
10.387…	857	4	2,1	OFPT_FLOW_MOD

Fig. 3. Frames exchanged during the learning phase

The evolution of $\mathcal{R}$ according to Algo. 1 is presented in Table. I. To simplify the table, just the observed path related to frame number 464 is considered.

TABLE I
EVOLUTION OF $\mathcal{R}$
$r_1 = \{pin_{464}, \{(pout_i)_{i\in[1,n]}, [1.879, 8.324]\}\}$
$r_2 = \{pin_{468}, \{(pout_i)_{i\in[1,n]}, [1.879, 8.324]\}\}$

Frame	$\mathcal{R}$	Path observed
464	$\emptyset$	$\emptyset$
468	$\emptyset$	$\emptyset$
854	$\emptyset$	$\{fmod_{854}\}$
855	$\emptyset$	$\{fmod_{854}, fmod_{855}\}$
856	$\emptyset$	$\{fmod_{854}, fmod_{855}, fmod_{856}\}$
857	$r_1 \cup r_2$	$\{fmod_{854}, fmod_{855}, fmod_{856}, fmod_{857}\}$

There is the observation of two flows requests, packet 464 for the route from host 10.0.1.1 and 468 for a route from 10.0.1.2 (noted pin_{464} and pin_{468}. The observer launches the Algo. 1. The first action of the controller is to respond to the ARP request by a Packet_Out which are the frames number 466 and 469. Those frame does not install path so are not considered by Algo. 1.

Firstly, the packet number 854 is observed and is a $fmod$, noted $fmod_{854}$. This command is divided in two actions, one regarding pin_{464} and the other regarding pin_{468}. It is added to the current route observed $r_{pin,464}$ and $r_{pin,468}$ through line 7. Then the consistency of these paths are checked in line 5. Here, as the paths are not consistent, no decision is taken and the observer waits for the other commands. It will be the same for the frames number 855, 856 and 857.

Now, for the frame number 857, noted $fmod_{857}$, the same process is done except that the paths satisfied the three conditions introduced in section II-B2 which means that the condition $cons(r_{pin,464})$ is now satisfied. Then, the path is stored in $\mathcal{R}$. Here we assumed that the learning phase was not under attacked. However, the anomalies would also have been detected by an analysis of the consistency of the commands lines 8 and 10.

C. Running State

The three following cases are supposed to be after the learning phase described above.

1) Case of topology evolution: The first case considered is the evolution of the infrastructure topology. Here, a link failure between $of2$ and $of4$ is considered and notified by a Port_Status ps.

Time	No.	Switch	Action	Command			Type	Port
134.990…	6574	54					OFPT_PORT_STATUS	s4-eth1
134.990…	6575	54					OFPT_PORT_STATUS	s4-eth1
134.991…	6576	58					OFPT_PORT_STATUS	s2-eth3
134.991…	6577	58					OFPT_PORT_STATUS	s2-eth3
135.038…	6580	59	4,1	OFPFC_ADD	OFPFC_ADD		OFPT_FLOW_MOD	
135.045…	6582	54		OFPFC_DELETE…	OFPFC_DELETE…		OFPT_FLOW_MOD	
135.047…	6587	55	3,2	OFPFC_DELETE…	OFPFC_ADD	OFPFC_MODIFY	OFPT_FLOW_MOD	
135.047…	6588	58	1,2	OFPFC_ADD	OFPFC_DELETE…	OFPFC_MODIFY	OFPT_FLOW_MOD	

Fig. 4. Frames exchange after the notification of a failure link

The path set up during the learning phase is impacted which means that the observer restarts a learning phase. Thus, during the next 0.01842 seconds the new route will be observed. The evolution of the path set up is shown in the Table. II.

TABLE II
EVOLUTION OF THE PATH INSTALLED.

Frame	Path Observed
6574	$\{fmod_{854}, fmod_{855}, fmod_{856}, fmod_{857}\}$
6580	$\{fmod_{854}, fmod_{855}, fmod_{856}, fmod_{857}, fmod_{6577}\}$
6582	$\{fmod_{854}, fmod_{855}, fmod_{856}, fmod_{6577}\}$
6587	$\{fmod_{854}, fmod_{855}, fmod_{6587}, fmod_{6577}\}$
6588	$\{fmod_{854}, fmod_{6588}, fmod_{6587}, fmod_{6577}\}$

At the end of the timeout, the consistence of the final route $r_{Ps} = \{fmod_{854}, fmod_{6588}, fmod_{6587}, fmod_{6577}\}$ is checked. Here, r_{Ps} is consistent and so, replaced r_1 in $\mathcal{R}$.

2) Case of attack: After this learning phase, at the observation of each similar request, the same path is expected. Let's develop the case with the anomaly described in section II-A. The corresponding frames, captured using Wireshark, are given in Fig. 5.

The evolution of $\mathcal{R}$ is given in Table. III. To simplify, just the path expected in the instance related to the frame number 1569 which is a Packet_In, noted pin_{1569}, is considered.

The Packet_In observed at the frame number 1569, noted pin_{1569}, is similar to pin_{464} which means that the similar data plane $path(r_1)$ is expected. During the running phase, the aim is to verify that the controller takes similar decision than in the learning phase. As $fmod_{1759}$, $fmod_{1762}$ and $fmod_{1766}$ are part of the data plane expected, the commands are assumed to be consistent and thus are deleted from the path

Time	No.	Switch	Action	Type
67.983…	1569	43		OFPT_PACKET_IN
67.993…	1570	44	3	OFPT_PACKET_OUT
67.994…	1571	44		OFPT_PACKET_IN
67.996…	1573	43	1	OFPT_PACKET_OUT
71.189…	1759	46	1,2	OFPT_FLOW_MOD
71.192…	1762	43	2,1	OFPT_FLOW_MOD
71.194…	1766	47	4,1	OFPT_FLOW_MOD
71.195…	1769	44	10,10	OFPT_FLOW_MOD

Fig. 5. Frames exchanged during the steady state with an attack

TABLE III
EVOLUTION OF $\mathcal{R}$ WITH r_1 AND r_2 THE TWO ROUTES LEARNED DURING THE LEARNING PHASE

Frame	$\mathcal{R}$	Path expected
pin_{1569}	$r_1 \cup r_2$	$\{fmod_{854}, fmod_{855}, fmod_{856}, fmod_{857}\}$
pin_{1571}	$r_1 \cup r_2$	$\{fmod_{854}, fmod_{855}, fmod_{856}, fmod_{857}\}$
$fmod_{1759}$	$r_1 \cup r_2$	$\{, fmod_{854}, fmod_{855}, fmod_{856}\}$
$fmod_{1762}$	$r_1 \cup r_2$	$\{fmod_{854}, fmod_{855}\}$
$fmod_{1766}$	$r_1 \cup r_2$	$\{fmod_{855}\}$
$fmod_{1769}$	$r_1 \cup r_2$	$\{fmod_{855}\}$

expected though line 20 of Algo. 1. Regarding $fmod_{1769}$ the command is part of no data plane expected and so is assumed to be inconsistent. As a consequence a fault is declared line 18 of Algo. 1. In this example, the bias considered was a modification of the transmission port but any other bias of the command, which can be formalized as in section IV, would be detected similarly.

3) Case of Failure: The case of the failure of the controller is considered. Formally, it corresponds to a command $pout \in \Sigma_{Out}$ biased $pout' = \emptyset \in \Sigma_{Out}$. The observed frames are represented in Fig. 6.

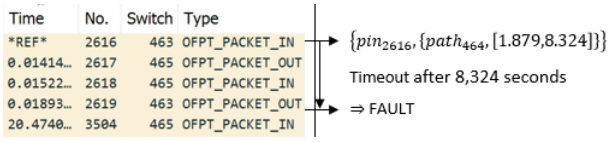

Fig. 6. Frames exchanged during a failure of the nominal controller

At the observation of pin_{2616} the template is instantiated in Algo. 1. After the timeout, no data plane has been observed so a fault is declared, line 22 and as represented in Fig. 6.

VI. CONCLUSION

In this paper, an alternative solution to detect threats at the controller in a SDN architecture has been proposed. It relies on a second "extra" controller, checking if anomalies exist concerning the decisions of the main controller. A novelty is that this detection is not based on exchanges between the controllers as in a classical distributed architecture but it only relies on analysing the traffic between the nominal controller and the switches. (Routing) decisions are hence learned by the observer which verifies any inconsistency representative of a fault or an attach.

In future works, we aim at extending the detection algorithm. Firstly, even if the observer is silent and only fault/attack on the controller are considered, vulnerabilities of the switches (and hence of the south interface captured by the observer) and the observer will be studied. Secondly, in a same manner, we aim at considering other network services (even based on non-deterministic logic). Finally, mechanisms for taking the lead over the nominal controller will be introduced.

ACKNOWLEDGMENT

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

REFERENCES

[1] N. McKeown, "Software-defined networking," *INFOCOM Keynote Talk*, vol. 17, pp. 30–32, 01 2009.
[2] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.
[3] D. Li, S. Wang, K. Zhu, and S. Xia, "A survey of network update in sdn," *Frontiers of Computer Science*, vol. 11, no. 1, pp. 4–12, 2017.
[4] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, "Multi-controller based software-defined networking: A survey," *IEEE Access*, vol. 6, pp. 15 980–15 996, 2018.
[5] N. M. Abd Elazim, M. A. Sobh, and A. M. Bahaa-Eldin, "Software defined networking: attacks and countermeasures," in *2018 13th International Conference on Computer Engineering and Systems (ICCES)*. IEEE, 2018, pp. 555–567.
[6] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for openflow networks," in *First workshop on Hot topics in software defined networks*, 2012, pp. 121–126.
[7] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, "A replication component for resilient openflow-based networking," in *2012 IEEE Network operations and management symposium*, 2012, pp. 933–939.
[8] C. Qi, J. Wu, H. Hu, G. Cheng, W. Liu, J. Ai, and C. Yang, "An intensive security architecture with multi-controller for sdn," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2016, pp. 401–402.
[9] X. Liu, H. Xue, X. Feng, and Y. Dai, "Design of the multi-level security network switch system which restricts covert channel," in *2011 IEEE 3rd International Conference on Communication Software and Networks*. IEEE, 2011, pp. 233–237.
[10] J.-H. Lam, S.-G. Lee, H.-J. Lee, and Y. E. Oktian, "Securing distributed sdn with ibc," in *2015 Seventh International Conference on Ubiquitous and Future Networks*. IEEE, 2015, pp. 921–925.
[11] F. Shang, Y. Li, Q. Fu, W. Wang, J. Feng, and L. He, "Distributed controllers multi-granularity security communication mechanism for software-defined networking," *Computers & Electrical Engineering*, vol. 66, pp. 388–406, 2018.
[12] ONF, *ONOS source: https://opennetworking.org/onos/*, 2021.
[13] A. Mubarakali and A. S. Alqahtani, "A survey: Security threats and countermeasures in software defined networking," in *2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT)*. IEEE, 2019, pp. 180–185.
[14] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang, "Rosemary: A robust, secure, and high-performance network operating system," in *ACM SIGSAC conference on computer and communications security*, 2014, pp. 78–89.
[15] ONF, *OpenFlow specification version 1.3.0 https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf*, June 2012.
[16] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.
[17] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou, "Specification-based anomaly detection: a new approach for detecting network intrusions," in *Proceedings of the 9th ACM conference on Computer and communications security*, 2002, pp. 265–274.
[18] D. N. Pandalai and L. E. Holloway, "Template languages for fault monitoring of timed discrete event processes," *IEEE transactions on automatic control*, vol. 45, no. 5, pp. 868–882, 2000.