Detection of security and safety threats related to the control of a SDN architecture

Loïc Desgeorges, Jean-Philippe Georges, Thierry Divoux

Detection of Security and Safety Threats related to the Control of a SDN Architecture

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 is a concept within the networking field which proposed a centralized control considering the control and data planes. To overcome the safety and security threats, solutions might be divided into two categories: enforcing the controller to make it more robust or the architecture using a multi-controller approach. This work aims to pave the way for a multi-controller architecture without East-West interface to avoid the spreading of an attack. There is one nominal controller in charge of the control while the second observes the traffic at the Southbound interface to detect anomalies of control. A detection method is introduced theoretically and relies on Intrusion Detection System theory, more precisely the specification-based. Here, the specification is a template determined through a projection function of the control logic. The template is compared to the activity of the command observed such that any deviation generates an alarm. The method is finally explained in use cases.

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

1. INTRODUCTION

Software-Defined Networking (SDN), McKeown (2009), has been introduced to provide a structured software environment to deal with various application requirements and dynamic networks, see Kreutz et al. (2014). It provides an architecture within the data plane is separated from the control part in a centralized control architecture manner. SDN simplifies network management and facilitates network evolution. The centralized control has two main issues: scalability and robustness. To overcome it, a variant of this architecture has been proposed: a distributed control. A multi-controller architecture permits to balance the load between the controllers while it provides an active redundancy, see Li et al. (2017). On top of that, it has some challenges in terms of consistency, reliability, load balancing and security as developed in Hu et al. (2018).

Indeed, each plane of the SDN architecture has its own weakness, see Abd Elazim et al. (2018). The controller is the heart of the network and has a global view of the network. As a consequence, a security or safety threat of the control plane has consequences on the entire network. Some mechanisms have been proposed in the literature to improve the security and/or safety of the control plane. These methods might be divided into two categories: the one which proposes a more robust controller to the threat considered as in Porras et al. (2012). Merely, this architecture presents a single point of failure. To consider the safety issue, it is mandatory to consider a multi-controller architecture as in Fonseca et al. (2012). Thus, the second category proposes to combine several controllers. The role of the second controller may differ according to the method. Qi et al. (2016) proposed a decision-making security architecture for the control layer. In this architecture, each commands is the result of a vote between all controllers. Similarly, Liu et al. (2011) proposed to set a filter which analyses the command sent by the controller and validates it. However, the information is collected by communication between the controllers through an East-West interface. Regarding a security point of view, this interface is opportune to the spread of attacks. To deal with this issue, Lam et al. (2015) proposed to introduce a Private Key Generator to encrypt the communication between the controllers. In the same way, Shang et al. (2018) proposed to use an indirect way of communication named "inter-domain agent flow". Additionally, they proposed a multi-granularity approach of the security and safety challenges of the controller. However, little attention has been paid for the consideration of both security and safety threats.

The objective of the paper is to pave the way to a safe and secure multi-controller architecture without East-West interface. In this architecture, one controller is in charge of the nominal control of the network while the second is in charge of the detection of a safety or security issue of the first controller and take the lead in case of anomalies. In this work, a novel detection method relying on a deterministic algorithm is introduced. To develop the logic, attention has been paid to Intrusion Detection System (IDS) theory. Compared to Giotis et al. (2014) or Tang et al. (2016), our method is based on the estimation of the intern variables of the nominal controller only by observation of the activity of the command and not on the information from the nominal controller or from traffic on the data plane (since both might be corrupted). We considered also the combination of anomaly and specification based approaches as in Sekar et al. (2002) which proposes to determine a model of the system's recurrent behavior based on a specification and then learn some statistical

properties to detect anomaly. Similarly we defined a specification of the command, formalized as a template as introduced in Pandalai and Holloway (2000) and completed on-line based on the estimation of the controller's interns variables in order to verify that there are no anomaly in the control.

2. THE DETECTION PROBLEM

The multi-controller architecture proposed is represented in Fig. 1. Without East-West interface, the detection method will not be based on the information we get from the other controller but on its capture activity and some *a priori* knowledge of the control logic.

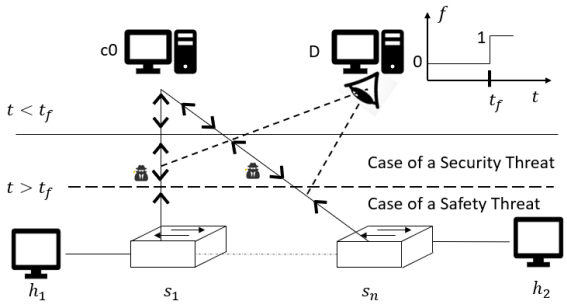

Fig. 1. Architecture proposed ($c0$: nominal controller, D: the detector, h_1 and h_2: two hosts, s_1 and s_n: the switches and t_f: the time of the failure or the beginning of the attack)

2.1 Principle

To develop the detection logic, attention has been paid to IDS. According to Liao et al. (2013), IDS works might be divided into two approaches: focusing on the attack behavior or on the unfaulty behavior of the system. The first approach is named signature-based detection and is focused on the attack signature. Thus, this approach concerns just the detection of attacks known and expected. The second one is composed of Anomaly and Specification-based approaches. They differ in their knowledge base of the system behavior but both aims to compare the unfaulty behavior known to the running behavior. Basically, anomaly detection techniques work on determining a model of the unfaulty behavior of the system. Regarding specification-based, the behavior is determined by a specification directly extracted from documentation.

In our approach, the specification formalism is a template, as introduced in Pandalai and Holloway (2000), which represents the action/reaction of the command. This specification evolved according to intern variables of the controller. Nevertheless, without East-West interface there is no access to these variables. Consequently, we proposed an observation of the controller activity to estimate its interns variables and then set up the specification's template to compare it with the activity of the command as represented in Fig. 2. Again, it is not the relevance of control that is judged but its consistency. If the controller is attacked before the observation has started then the malicious behavior will be considered as the reference and a non-malicious packet inconsistent will be declared faulty.

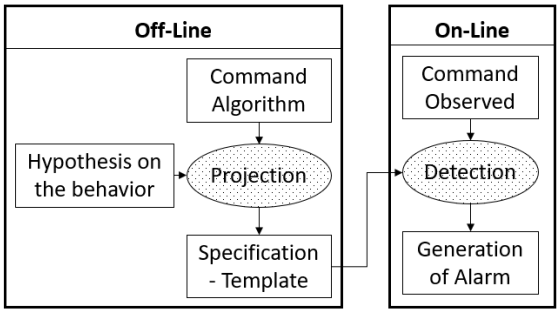

Fig. 2. Representation of the steps of the detection method

2.2 Formalisation

This section aims to formalize the activity of the command. OpenFlow, developed by ONF (June 2012), is the protocol which standardized the Southbound interface's communication and is considered in this study.

Events in the network From a controller's point of view, there are two kinds of events in the network: the incoming messages Σ_{In}, the requests from the switches, and the outgoing messages Σ_{Out}, the commands for the switches: $\Sigma = \Sigma_{In} \cup \Sigma_{Out}$

The first set, Σ_{In}, corresponds to the "Packet_In" which is the packet 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 MAC source address of the packet.
- $dest$: the MAC 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" Σ_{PO} which is single uses: the switch does not retain the information and will have to ask again to the controller what to do. Secondly, "Flow_Mod" Σ_{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 nextly.
- $b \in \mathbb{N}$: the buffer ID of the packet.
- $S \in \mathbb{N}$: the switch destination of the command.

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

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

Actions The set of actions, $Actions$, can be divided into two sets $\{Forward \cup Set\}$. Here, layer 2 forwarding applications will be considered so just $Forward$ is formalised. It corresponds to actions sent by the controller to a switch for the forwarding of a packet above a given output port.

So, an action can be modelled as a vector within each component is a binary value, associated with a port of the switch, representing the transmission or not.

$$Actions = \left(\left\{ \prod_{j=1}^{N} b_j \quad (b_j) \in \mathbb{B} \right\} \right)$$

Bias In case of an attack or a failure, the control algorithm returns a biased command. This bias may have several origins in the architecture as explained in Mubarakali and Alqahtani (2019). It might be a bias b_{Pin} of the Packet_In from the data plane, due to a man in the middle attack for example, or a bias b_{Cgn} of the consign from the application plane, due to an application eviction attack as presented in Shin et al. (2014) or a bias b_{Cmd} of the command from the control plane. As the method does not focus on isolating the fault, only b_{Cmd} will be considered. Let us formalised a biased command $pout' \in \Sigma_{Out}$ 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.
- $+$ is the operator defined as:
  $\forall i \in [1, lengh(pout)]\ pout'[i] = pout[i] + b_{Cmd}[i]$

The challenge consists therefore in detecting the additive bias, i.e. b_{Cmd}.

3. PROJECTION AND DETECTION

3.1 Projection

We assumed that the logic (limited here to layer 2 forwarding applications) embedded in the controller is known. The capabilities of the attacker is to provoke a bias specifically on the controller. Also, the reason of the detected anomaly is not isolated. Regarding the behavior of the controller, whatever the request of a switch, the same number of commands is waited from the controller and the estimation of the computation time is not developed. Moreover, at the starting of the observation of the control, we assume that the controller is not processing a request. We then introduce the projection function which aims to determine the specification template of command algorithms such as Bellman Ford routing, packet filtering, etc.

Causality in the algorithm An hypothesis of this work is that the command algorithms embedded in the controller are known. The action/reaction causality of the algorithm is located in the instructions. The output of the algorithm, which corresponds to a Packet_Out, is returned if the condition of the instruction, which corresponds to a specific Packet_In, has occurred. Consequently, we decided to model the algorithms *Algo* through the instructions $\mathcal{I}$.

$$Algo = \mathcal{I}^m \text{ and } \mathcal{I} = \Sigma_{In} \times Op \times \Sigma_{Out}^n$$

An instruction $istr \in \mathcal{I}$ is defined as:

- $istr = (pin, op, POUT)$
- $cond \in (\Sigma_{In})^{\mathbb{B}}$: a boolean equation which is verified by a set of Packet_In $pin \in \Sigma_{In}$.
- $op \in Op$: a set of operations
- $POUT = \cup_{i=1}^{n} pout_i \in \Sigma_{Out}^n$: the set of output corresponding to the commands.

Template Inspired by Pandalai and Holloway (2000), we define a template as an object which gathers a request from a switch and the answered command. Consider $\mathcal{T}$ the set of templates and $temp \in \mathcal{T}$ defined as:

- $temp = \{pin, \cup_{i=1}^{n}(pout_i, [t_i, t_i'])\}$
- $pin \in \Sigma_{In}$: the triggered event. It might be empty in case of spontaneous command.
- $\cup_{i=1}^{n}(pout_i, [t_i, t_i']) \in (\Sigma_{Out}, \mathbb{R}^2)^n$: the set of command, $pout_i$, expected to answer the event pin. Each command is expected to occur during an interval of time $[t_i, t_i']$.

Indeed, a command cannot be waited indefinitely. The function Est is introduced to return this interval of time but is not developed in this work. Est is defined as follows:

$$\begin{array}{rl} Est : \mathcal{I} \times (\Sigma_{In} \cup \Sigma_{Out}) & \rightarrow \mathbb{R}^{2+} \\ (istr, L_i) & \mapsto it_i \end{array}$$

With:

- $istr \in \mathcal{I}$: the instruction considered.
- $L_i \in \Sigma_{In} \cup \Sigma_{Out}$: the line of the instruction to reach.
- $it_i \in \mathbb{R}^{2+}$: the interval of time within the line is reached.

Projection's tool The projection permits to obtain the template for an algorithm. As an algorithm is modelled as a set of instructions, first let us introduce the function $Proj_{Istr}$ as:

$$\begin{array}{l} Proj_{Istr} : \mathcal{I} \times (\mathcal{I} \times \Sigma)^{\mathbb{R}^{2+}} \rightarrow \mathcal{T} \\ \qquad (istr, Est) \mapsto \{pin, \{(pout, Est(istr, pout))\}\} \end{array}$$

With:

- $istr = (pin, op, POUT)$: an instruction.
- $\{(pout, Est(istr, pout))\} \in (POUT \times \mathbb{R}^{2+})^n$: the set of expected commands

Moreover, to simplify the reading we assumed that for each instruction the same number n_{pout} of commands is waited.

$$\begin{array}{rcl} Proj_{Algo} : Algo & \rightarrow & \mathcal{T} \\ a & \mapsto & \left\{pin, \cup_{i=1}^{n_{pout}}(pout_i, it)\right\} \end{array}$$

With:

- $it = [t_1, t_2]$: fixed interval of time
- $t_1 = min_{istr \in Algo} min_{pout \in istr[3]} Est(istr, pout)$: the lower bound of the intervals of time.
- $t_2 = max_{istr \in Algo} max_{pout \in istr[3]} Est(istr, pout)$: the upper bound of the intervals of time.

For a trigger event, pin, a set of commands $pout_i$ is expected. Besides, the function $Proj_{Algo}$ permits to accomplish the step Projection of the Fig. 2. Intended changes of the algorithms of the nominal controller would have to be notified to the observer through its north interface.

3.2 Detection

As mentioned and developed in the section 2, the principle of the method is to compare the trace of the messages send/received by the controller to the instances of the template. The process is synthesized in Fig. 3 and developed here after. In this work, the intern variable to estimate will be the MAC Table MTN of the controller in case of a routing command defined as $MTN = \{(S, MT)\}$ with:

- S: the switch considered.

- $MT = \{(p, MAC)\}$: the MAC Table of the switch S with p the port to join the MAC Address MAC.

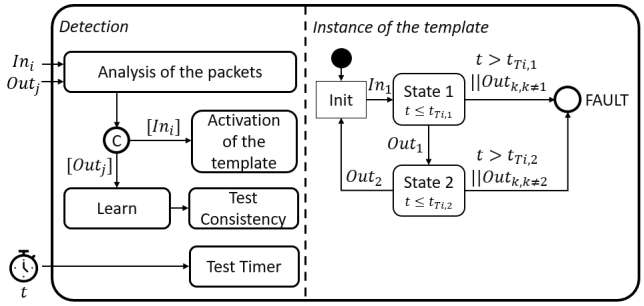

Fig. 3. Representation of the algorithms process

The input of the algorithm is the activity of the command, so the first step is to treat those packets by Algo. 1 which launches the algorithm related to the observed packet. It corresponds to the junction of Fig. 3.

Algorithm 1 Analysis

Input: Packet p, MAC Table Estimated MTN and O
if $p \in \Sigma_{In}$ **then**
  $Instance(p, MTN, O)$
else
  $Match(p, MTN, O)$

In case of a Packet_In, Algo. 2 is launched and the template is instantiated. The expected commands are completed based on the knowledge stored about the controller's intern variables. It corresponds to launch the instance of the template as represented in Fig. 3. The set of the template instantiations at a date t is noted as $O(t)$. We make the hypothesis that initially there are no command expected: $O(t = 0) = \emptyset$.

Algorithm 2 Activation

Input: Packet_In pin, MAC Table Estimated MTN and O
$i = 1$, $Know = 0$, $MTN_{Check} = MTN[1]$
$Temp[1] = pin$, $Temp[2][2] = it$
while $MTN_{Check}[1]! = pin[1][2]$ **do**
  $MTN_{Check} = MTN[i + 1]$, $i = i + 1$
$MT = MTN_{Check}[2]$
for $Dest \in MT[2]$ **do**
  if $Dest = pin[3]$ **then**
    $Temp[2][1] = MT[1]$, $Know = 1$
    $O = O \cup Temp$, $Timer(Temp)$
if $Know = 0$ **then**
  $Temp[2][1] = \emptyset$,$O = O \cup Temp$,$Timer(Temp)$

As soon as the template is instantiated a timer is launched through Algo. 3 to verify that the command is in time. Basically, it consists into waiting the last action before removing the instance from O.

On top of that, when a command is observed, the consistency of the command is checked through Algo. 4 as represented in Fig. 3. For each instance, there is a verification that the command was the one expected by this instance. In addition, there is a formal verification of the safety of the path installed by the controller, which is not developed

Algorithm 3 Timer

Input: An instance $Temp$
$fault = False$, $sleep(min(Temp[2][2]))$
if $Temp \notin O$ **then**
  $fault = True$
$sleep(max(Temp[2][2]) - min(Temp[2][2]))$
if $Temp \in O$ **then**
  $fault = True$
return $fault$

in Algo. 4 to ease the readability. It checks that there are no forwarding loops, no-dead node and the destination reaching. In such a case, the instance is deleted from O. Else, the command was not expected and assumed to be inconsistent. Moreover, when a command is observed it is necessary to update the estimation of the controller's intern variables through Algo. 5.

Algorithm 4 Consistency

Input: Packet_Out $pout$, MAC Table Estimated MTN and O
$fault = True$, $Learn = 0$
for $Temp \in O$ **do**
  if $Temp[2][1] = pout$ **then**
    $fault = False$, $O = O \setminus Temp$
  else if $Temp[2][1] = \emptyset$ *and* $pout[3] = Temp[1][1][2]$ **then**
    $fault = False$, $O = O \setminus Temp$, $Learn = 1$
if $Learn = 1$ **then**
  $Learn(MTN, pout)$
return $fault$

Algorithm 5 Learn

Input: Packet_Out $pout$ and MAC Table Estimated MTN
$MTN_{Check} = MTN[1]$, $newLigne = 1$, $i = 1$
while $MTN_{Check}[1]! = pout[3]$ **do**
  $MTN_{Check} = MTN[i + 1]$, $i = i + 1$
$MT = MTN_{Check}[2]$
for $Dest \in MT[2]$ **do**
  if $Dest = pout[5]$ **then**
    $newLigne = 0$
if $newLigne = 1$ **then**
  $ML_{new} = (pout[5], pout[1])$
  $MT[2] = MT[2] \cup ML_{new}$

In these algorithms, the MAC Table Estimated MTN has been considered as the intern variable of the controller. But it can be changed by other intern variables on the condition that Algo. 5 is updated in consequences.

4. USE CASES

The aim of this section is to illustrate the method presented just above in use cases.

4.1 Scenario

The topology represented in Fig. 4 is simulated using Mininet. An ONOS (ONF (2021)) controller is loaded with

the MAC learning forwarding application. We name MTN the MAC Table Estimated of the controller defined as in section 3. The algorithm is structured in two instructions : $istr_1$ retransmits on a particular port the packet and $istr_2$ floods the packet:

$istr_1 = pin_1 \cup op_1 \cup pout_1$ and $istr_2 = pin_2 \cup op_2 \cup pout_2$

The computation of the time constraints is not further discussed here for the demonstration, we fixed $Est(istr_i, pout_i) = [0.0005, 0.02]$ for $i = 1, 2$. Let's apply $Proj_{Istr}$:

$$Proj_{Istr}(istr_1, Est) = \{pin_1, \{(pout_1, [0.0005, 0.02])\}\}$$
$$Proj_{Istr}(istr_2, Est) = \{pin_2, \{(pout_2, [0.0005, 0.02])\}\}$$

Thus, the algorithm's template $\forall pin \in \Sigma_{In} \forall pout \in \Sigma_{PO}$:

$$Proj_{Algo} = \{pin, \{(pout, [0.0005, 0.02])\}\}$$

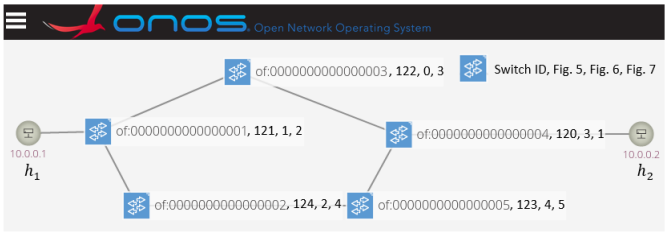

Fig. 4. The topology of the network

The scenario applied is a ping from $h1$ to $h2$. The detection method will be put into practice in three cases: without anomalies, with an attack and with a failure. The frames, captured using wireshark, of the scenario without anomalies is represented in Fig. 5. To ease the readability, the value of the instances of the template and the evolution of O are detailed in Fig. 5. It will be the same for the case of attack and the case of failure.

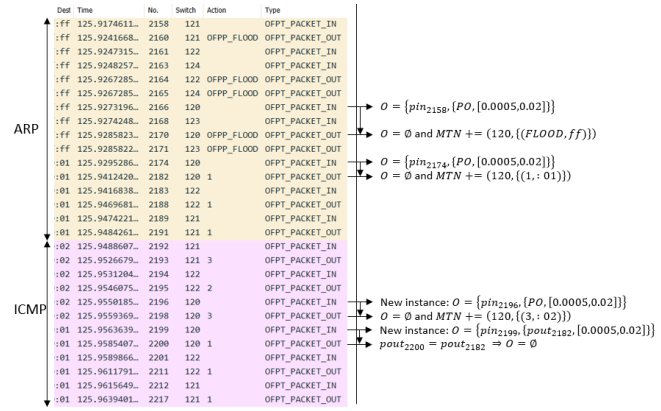

Fig. 5. Packets exchanged during a ping without anomalies

The evolution of $O(t)$ concerning the switch 4 is detailed in Fig. 5. Firstly, the Packet_In pin_{2158} is observed. The main algorithm Algo. 1 launches the instantiation through Algo. 2. The condition line 11 is not fulfilled as pin_{2158} has not yet been observed, so any specific command is expected. The Packet_Out $pout_{2170}$ is observed in the direction of the switch 4. Similarly, Algo. 1 launches the verification through Algo. 4. As an instance expects $\emptyset$ for the switch 4, the condition line 27 is fulfilled and the instance will be deleted from O. Moreover, Algo. 5 is launched to register the command in MTN as represented in Fig. 5.

Secondly, at the observation of requests already seen, as pin_{2199} or pin_{2413}, the template instantiation is based on the knowledge stored previously in MTN, according to the condition line 11 of Algo. 2. Thereafter, commands $pout_{2200}$ and $pout_{2415}$ are observed and are similar to the one learned $pout_{2182}$. As a consequence, the condition line 25 of Algo. 4 is fulfilled so no anomaly is detected.

4.2 Case of an attack

Let's consider the dynamic flow tunneling attack developed in Porras et al. (2012). To model such an attack, a bias b_{Cmd} of a command $pout \in \Sigma_{Out}$ which leads to $pout' \in \Sigma_{Out}$ is introduced:

$\forall i \in [1, lengh(pout)]\ pout'[i] = pout[i]$
$pout'[1] = 10$ if $pout[3] = of4$, $pout'[1] = pout[1]$ else

The attack is simulated by adding on the controller an extra code modifying commands such that all traffic which passes through the switch 4 is in practice retransmitted on the port 10. In this section, two pings are considered: a first which permits the detector to learn the unfaulty behavior of the command and a second under attack. The corresponding frames are represented in Fig. 6.

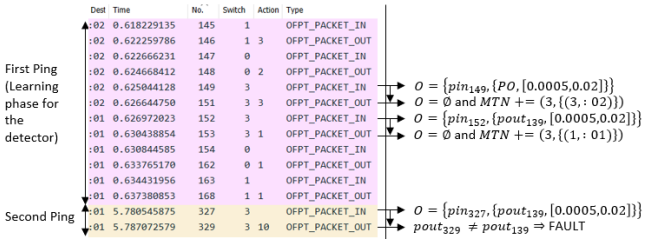

Fig. 6. Packets exchanged during two pings. The first is the learning phase and the second is under the attack

The aim of what follows is to show how an anomaly is detected. First, there is the learning phase with the observation of pin_{149} and $pout_{151}$ as with pin_{152} and $pout_{153}$. Based on those exchange of packets, the estimation of MTN has been completed through Algo. 5. As a consequence, in the observation of pin_{327} the template is instantiated through Algo. 2 with the expectation of the same command as $pout_{139}$. But $pout_{329}$ is different of the command contained in the instance of O. Thus, the condition line 25 of Algo. 4 is not fulfilled and a fault is declared as shown in Fig. 6. In this example, the bias considered was a modification of the port of transmission but any other bias of the command which can be formalized as in section 3 would be detected similarly.

4.3 Case of a failure

The case of link failure between the switch 4 and the controller is considered it corresponds to a command $pout \in \Sigma_{Out}$ biased $pout' \in \Sigma_{Out}$ as follows:

if $pout[3] = of4$: $pout' = \emptyset$ and else: $pout' = pout$

To set up it, the code of ONOS is modified in order to not send the packets in direction to the switch 4. The observed frames are represented in Fig. 7.

At the observation of pin_{589} from the switch 4 the template is instantiated in Algo. 2 and in parallel a timer is launched

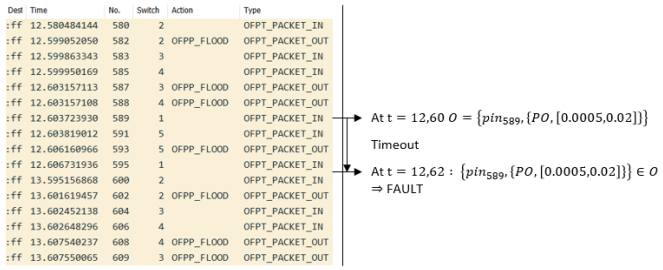

Fig. 7. Packets exchanged during a ping. Case of a failure.

by Algo. 3. After the waiting of 0.02 seconds a fault is declared in Algo. 3 as represented in Fig. 7.

5. CONCLUSION

To conclude, a secure and safe detection method, of both security and safety issue, has been proposed theoretically. In this objective, a multi-controller architecture is proposed, without East-West interface, within a controller is in charge of the control of the network while the second observes the activity of the command and compares it to the template specification. This template is determined using a projection function and corresponds to the expected behavior of the command. The detection process consists into instantiating this template by the observation of the controller behavior to verify the consistency of its control which permits to detect anomalies. To pursue in this objective, this work would have to be experimented in the future and extended to non-deterministic algorithms using machine learning techniques. Moreover, techniques to take the lead, after the detection of a fault, by the second controller will have to be developed.

ACKNOWLEDGEMENTS

This work was supported partly by the French PIA project “Lorraine Université d’Excellence”, referenceANR-15-IDEX-04-LUE.

REFERENCES

Abd Elazim, N.M., Sobh, M.A., and Bahaa-Eldin, A.M. (2018). Software defined networking: attacks and countermeasures. In *2018 13th International Conference on Computer Engineering and Systems (ICCES)*, 555–567. IEEE.

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.

Giotis, K., Argyropoulos, C., Androulidakis, G., Kalogeras, D., and Maglaris, V. (2014). Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments. *Computer Networks*, 62, 122–136.

Hu, T., Guo, Z., Yi, P., Baker, T., and Lan, J. (2018). Multi-controller based software-defined networking: A survey. *IEEE Access*, 6, 15980–15996.

Kreutz, D., Ramos, F.M., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S., and Uhlig, S. (2014). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1), 14–76.

Lam, J.H., Lee, S.G., Lee, H.J., and Oktian, Y.E. (2015). Securing distributed sdn with ibc. In *2015 Seventh International Conference on Ubiquitous and Future Networks*, 921–925. IEEE.

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.

Liao, H.J., Lin, C.H.R., Lin, Y.C., and Tung, K.Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1), 16–24.

Liu, X., Xue, H., Feng, X., and Dai, Y. (2011). Design of the multi-level security network switch system which restricts covert channel. In *2011 IEEE 3rd International Conference on Communication Software and Networks*, 233–237. IEEE.

McKeown, N. (2009). Software-defined networking. *INFOCOM Keynote Talk*, 17, 30–32.

Mubarakali, A. and Alqahtani, A.S. (2019). A survey: Security threats and countermeasures in software defined networking. In *2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT)*, 180–185. IEEE.

ONF (2021). *ONOS source: https://opennetworking.org/onos/*.

ONF (June 2012). *OpenFlow specification version 1.3.0 https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf*.

Pandalai, D.N. and Holloway, L.E. (2000). Template languages for fault monitoring of timed discrete event processes. *IEEE transactions on automatic control*, 45(5), 868–882.

Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M., and Gu, G. (2012). A security enforcement kernel for openflow networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, 121–126.

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.

Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., and Zhou, S. (2002). Specification-based anomaly detection: a new approach for detecting network intrusions. In *Proceedings of the 9th ACM conference on Computer and communications security*, 265–274.

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.

Shin, S., Song, Y., Lee, T., Lee, S., Chung, J., Porras, P., Yegneswaran, V., Noh, J., and Kang, B.B. (2014). Rosemary: A robust, secure, and high-performance network operating system. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 78–89.

Tang, T.A., Mhamdi, L., McLernon, D., Zaidi, S.A.R., and Ghogho, M. (2016). Deep learning approach for network intrusion detection in software defined networking. In *2016 international conference on wireless networks and mobile communications (WINCOM)*, 258–263. IEEE.