Enhanced Solid-Flow: An Enhanced Flow Rules Security Mechanism for SDN

[1]Youssef QASMAOUI, [2]Abdelkrim HAQIQ, IEEE Senior Member

***Abstract*—Software-Defined Networking (SDN) is a new method that aims to virtualize networking by decoupling data plan from the control plan. As this technique impacted many actors in the networking community because of its suppleness, programmability, and central decision management. Many interrogations have been raised about problems that come with it. One of the biggest concerns related to SDN is security. This trait is crucial when determining to implement the SDN as a solution or not. Applications plane contains applications that can generate flow rules, those applications are allowed to access the data store present within the controller to store their rules, which introduces vulnerabilities in the network making the flow rules untrustworthy. Access to data store can only be trusted if the security problems inherent in software-defined networks are resolved. In this paper, we propose an enhanced SOLID-FLOW to improve flow rules database integrity inside the SDN controller. First, we discuss security threats related to SDN mechanisms regarding the control layer. Secondly, we debate appropriate solutions that have been presented in the literature to address security-related issues. Then we present our enhanced mechanism. Finally, we discuss our implementation use case and its results. The last section will provide a conclusion and an overview of future works.**

***Index Terms*— Software-Defined Networking; SDN security; Control plan; Solid-Flow; OpenFlow.**

I. Introduction

THE SDN - Software-Defined Networking - is the hot topic that shakes the world of the network in recent years.

The SDN is recognized today as an architecture for opening the network to the applications and central management. This is far from the original rigid definition in which it was just about to separate the control plane and data. Therefore, OpenFlow [1] is a component of SDN that offers programming and infrastructure simplification. At this level, several models of SDN have been designed in parallel. The work generally focuses on the inherent programmability of equipment (new programmable devices...), the SDN controllers and orchestrators whose mission is to provide a network abstraction layer, network, and virtualization functions that aim to overcome the
complexity of the underlying physical network making the configuration more agile. The challenge for network managers is to follow-up on this new stage to take advantage of these new potentials.

As enterprises choose to implement Software Defined Networking (SDN), one of the main concerns raised is SDN security problems. Enterprises are seeking if SDN products will guarantee that their applications, sensitive data, and infrastructure won't be exposed to any security risk related to SDN. With its introduction, new strategies for securing the SDN application layer, control layer, infrastructure layer are needed. In this paper, we review the attack vectors affecting the control plan in case of SDN implementations and share ways to secure it. After, we propose an enhancement to our approach called “Enhanced Solid-Flow,” which will rely on our previous work [2] to more secure the flow rules database leading to enhance trust in flow rules present in the controller.

In summary, we make the following contributions:

- We propose a new security mechanism based on hashing algorithms and authorization to handle flow rules insertion.
- We design and develop the mechanism that runs in three scenarios for best performance and efficiency and accuracy.
- We evaluate the performance of our solution in deferent scenarios regarding reactivity CPU usage and response time.

The remainder of this paper is organized as follows: Section II presents SDN architecture and principals. Section III security challenges facing the SDN controller. Section IV summary the proposed solutions to dress SDN security issues. Section IV outlines the enhanced Solid flow architecture and its scenarios, and Section V discusses our implementation and performance evaluation. Finally, Section VI concludes with a discussion of our future works.

II. SDN ARCHITECTURE

The purpose of SDN technology is to offer some open interfaces that enable the development of software that can manage the connectivity supplied by a collection of network resources and the flow of network traffic through them, along with possible examination and adjustment of traffic that may transit in the network [3]. The ONF (Open Networking Foundation) defined three main principles of SDN [3]:

- Separating the controller and data planes:
- Decouple control plan from the data plan. However, control must be executed within the data plan system.
- A central control: This aspect will provide a high overview of the network and can enhance the deployment of decision making instead of local control, therefore changes latency are less significant.
- Exposure of abstract network resources and state to outer applications.

Manuscript received August 23, 2019; revised August 23, 2020. This work was supported in part by Computer, Networks, Mobility and Modeling Laboratory at the Faculty of Science and Technology in Settat, Morocco.

[1]Hassan First University, Faculty of Sciences and Techniques, Computer, Networks, Mobility and Modeling laboratory: IR2M, 26000 - Settat, Morocco (Email: qasmaoui@gmail.com).
[2]ORCID ID: https://orcid.org/0000-0002-8857-6586.
Hassan First University, Faculty of Sciences and Techniques, Computer, Networks, Mobility and Modeling laboratory: IR2M, 26000 - Settat, Morocco (Email: abdelkrim.haqiq@uhp.ac.ma).

These principals are clarified by the SDN architecture, which is built around three principal layers: Application plane layer, Controller plane layer and Data plane layer.

A. *Data plane*

The data plane includes the forwarding hardware i.e. switches/routers, and incorporates all software interfaces and hardware elements [4]. The forwarding operations are handled using a forwarding table called flow table, which contains rules that manage forwarding operations. Fig. 1. Shows the Flow Table Entry architecture.

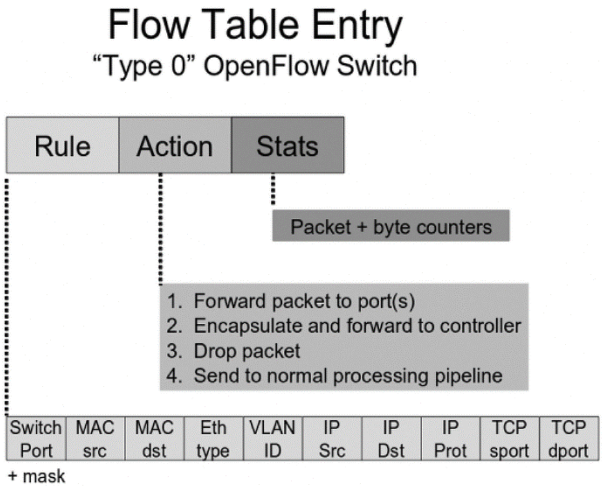

Fig. 1. Open Flow –Table

B. *Control plane*

Network intelligence is installed in the software-based logically centralized SDN controller; this controller communicates through a standard southbound API (OpenFlow). The control layer regulates and manages the forwarding decisions of the hardware [4]. ONF designed the SDN architecture with only a single Controller but new scenarios use multiple distributed controllers [5][6] to enhance the availability and scalability of network resources.

C. *Application plane*

Applications and services explore possibilities offered by the control and infrastructure layer. The abstract application layer is located over the control plane layer and enables the unsophisticated development of network applications [3], such as network virtualization, traffic monitoring, etc. Those applications interact with the control layer through the northbound API.

Application can include SDN specific applications belonging to SDN providers itself, but it can also have third-party application developed to meet specific needs such as applications related to network automation to better align with the needs of the applications running on it, network configuration and management to enable real time configuration and ease of administration , network monitoring, network troubleshooting, network policies and security. Those application often present high-risk rate, specifically third party applications, the threats generally target the control plan because of its high value, hence securing the access to control data provided by the SDN controller is mandatory.

Fig. 2. shows the architecture of the typical SDN network as described by the Open networking foundation.

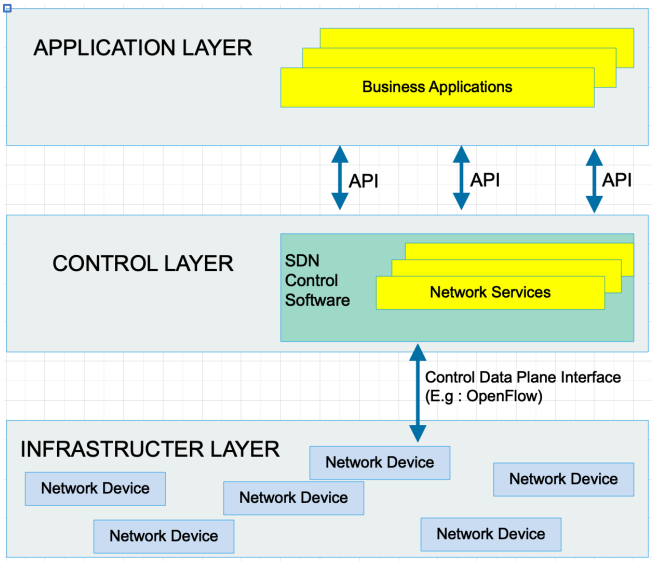

Fig. 2. SDN Architecture showing the tree layers (Application plan, Control plan, Data Plane)

III. SDN SECURITY CHALLENGES

Since the introduction of SDN, security issues have been raised because of the nature of its concept, which consists of separating the control plane from the data plane. The list of security issues will grow with the deployment of SDN technologies.

In this section, we will debate some of the most significant security challenges menacing the control layer.

A. *Security challenges in Control Plane*

The SDN control plane is a central making decision entity because of its particularity. The controller is the most suitable target for hackers. It's the brain of any network based on SDN technology. This entity is exposed to several risks, as we will present below.

1) *Unauthorized controller access*

Threats presented by the application layer affect the control layer; this threat exists because of the lack of any trust and reliable interaction between third party applications and the controller [7]. As the controller is responsible for authorizing applications to use network resources with proper access levels, these applications need to be separated. This separation should ensure that applications with low authorization levels could not handle high privilege. E.g: load balancing applications need to get packet statistics and IDS need header fields of the packet, and such custom access needs custom security policies.

2) *Availability:*

Denial of Service (DoS) attacks and Distributed Denial of Service (DDoS) flooding attacks are the main approaches to terminate the availability of a machine or network resource. DoS and DDoS attacks are one of the significant security challenges[8]. Since the SDN control plane is located in the controller within a physical machine, DDOs attack still a significant risk to SDN, which compromises the availability of the controller and consequently impacts the whole SDN network [8].

3) *Scalability & availability*

Concentrating network intelligence in the controller exposes the network to scalability and availability if a DOS

/ DDOS attacks succeed to paralyze the regular running of the controller, so both application layer and data layer functionalities will hang.

Another issue with the control plan is the adoption of Transport Layer Security (TLS) [9] mechanisms, which was proposed to secure the southbound communication with the transport layer. In the last version of open flow, this mechanism is optional and vendors rarely implement it because of its complexity of implementation, the lack of TLS mechanisms deployment in SDN network exposes it to many security threats such as DOS / DDOS and man-in-the-middle attacks.

In this section, we presented the most relevant security issues that come up with the adoption of the SDN network solution at the control layer level.

In Table I, we review a list of different threats related to the control layer.

TABLE I
SDN SECURITY THREATS

SDN Layer	Threat of nature	Threat Description
Control Layer	Lack of authentication and authorization	No suitable authentication & authorization mechanisms due to the diversity of third party applications.
	Fake flow rules	Malicious or compromised applications can generate false flow rules
	Lack of access control and accountability	Challenging to implement access control and accountability on third-party applications that consume network resources.
	Scalability and availability	Centralizing intelligence in the control plan will expose the network to scalability and availability issues.
	Unauthorized controller access	No suitable mechanisms for enhancing access control on applications.
	DoS attacks	The centralized nature of intelligence in the SDN network is attractive to DDOS attacks
	TCP-Level attacks	TLS is vulnerable to TCP-level threats
	Man-in-the-middle attack	Due to optional implementation of TLS, if not used it's exposed the network to Man-in-the-middle attack

VI. Related work

A. Security Solutions in control Plane

Securing the control plan from various types of threats, which can be resumed as scalability and availability, unauthorized controller access, fake flow rules and DoS attacks, these threats have been a challenge since the introduction of SDN. Many solutions were proposed to solve these issues [10].

The SDN controller performs an intermediate operation between the network hardware and applications by abstracting the network complexity from applications. Thus, the centralized control architecture enabled by SDN makes it easy to deploy new applications that would collect network information through the controller. Therefore, various network programming languages such as Frentic [11] Procera [12] and NetCore [13] were proposed to simplify the development of applications in SDN. Furthermore, FRESCO was introduced to enable the development of OpenFlow security applications, besides language programming. Different frameworks were suggested to solve the compliance between SDN application and network security policies.

1) *Lack of access control and accountability*

SDN applications need to follow these applications to claim the network specifications regarding the information. To dress this issue, one of the critical solutions is permOF [15], which is a well-grained permission system that evolves a set of OpenFlow permission and runtime isolation mechanism applying for permissions. The permission set is sorted into reading, notification, write, and system permissions.

Table II shows these permissions and further sub-categories.

TABLE II
PERMOF PERMISSION SET

Category	Permission
Read	read topology
	read all flow
	read statistics
	read pkt in the payload
Write	flow mod route
	flow mod drop
	flow mod modify hdr
	modify all flows
	set device config
	set flow priority
Notification	pkt in event
	flow removed event
	error event
	topology event
System	network access
	file system access
	process runtime access

The read permission defines what information application can retrieve from the network through the controller. The write explains whether an application can modify or not the state of the controller or switches, notification handle permission of notifying the application if a specific event acquires and the system category manages access to local resources of the operating system.

2) *Fake flow rules*

Fake flow rules remains the first and most critical issue to deal with, any flow rules residing within the controllers is legitimate and expose the controller to propagate false and misleading routing information, hence the importance given to solve this issue among the researcher community.

Many types of research conducted to propose FLOVER, which is a model-checking system that verifies the flow policies against the security policies of a network and make decision which is transferred to the conroller.

In the SDN architecture, FLOVER [16] is executed as an OpenFlow application, but logically it is located in parallel to the controller. The controller is modified to request FLOVER's permission on every new flow rule generation or modification. Primarily, the controller communicates flow tables and security policies to FLOVER which contains critical security information to the network preconfigured by the network administrator, so FLOVER can evaluate requested changes accordingly and respond to them. FLOVER also uses the controller to access the current state and the network information's and statistics, like flow rule tables on the switches. Fig. 3. shows the architecture of FLOVER alongside the communication between Open Flow controller, OV switches and the FLOVER application.

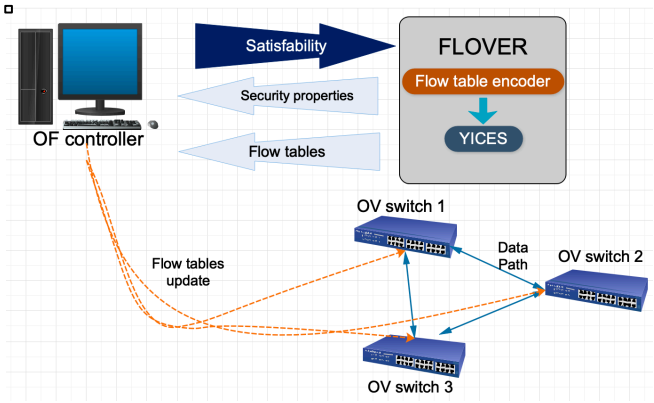

Fig. 3. FLOVER architecture

Another flow rule conflict management system is FortNox. FortNox [17] employs a security enforcement kernel (SEK) to enforce flow controls for active defense against different threats. FortNox is an enhancement engine that is responsible for enforcing and avoiding rule conflicts from separate security authorizations. FortNox uses two protection mechanisms:

- Rule prioritization, which ensures that any new rule that contradicts the rules produced by FRESCO applications is simply overridden because of the highest priority.
- The conflict detection algorithm is applied to each new rule.

In [18] the authors propose PERM-GUARD a system for managing and authorizing flow rules. PERM-GUARD uses an authentication/authorization model to verify the validity of the controller's flow rules through identity-based signature, this solution effectively filters out unauthorized flow rules created by valid applications and traces their creator in a timely and accurate manner.

3) Authentication and Authorization

The lack of the implementation of any authentication and authorization mechanism by third-party applications compromises the normal network behavior. FRESCO [13] a security-specific application development framework for Open Flow networks was proposed, FRESCO is a security application development platform that facilitates the exportation of API scripts, which help security experts develop threat detection logic and security monitoring as programming libraries. Moreover, FRESCO programming framework was presented to help attain rapid design and modular composition of different security mitigation and detection modules using Open Flow.

In [18], the authors proposed a modular SDN security-control communication architecture, KISS, with innovative solutions in the context of key distribution and secure channel support. Besides, they suggest iDVV, the integrated verification value of the device, a code protocol for generating a secret code that is deterministic but indistinguishable from chance. This allows local but synchronized generation/verification of keys at both ends of the channel even by message. IDVV should make a significant contribution to both the robustness and simplicity of authentication and secure communication problems in SDN.

4) Scalability & availability

SDN controller is responsible for handling all network requests and events. As the size of the network keeps growing the controller can reach a state of a bottleneck. Benchmarking of the SDN NOX [19] controller shows that it can support up to 30k demand this rate can be a serious issue to networks with high requests.

To assuage this matter levelling parallelism in multicore systems was proposed by Tootoonchian et al. [20] their approach showed that minor alterations to the NOX controller increase its performance by an order of greatness on a single core. It means that a single controller can support a further larger network, given adequate controller channel bandwidth with acceptable latency. More improvement can be achieved by reducing the number of requests sent to the controller using DIFANE [19] unique switches, called authority switches. These switches are used to discharge the tasks of the controller and handle the packets in the data plane. Most micro-flows are handled in the data-plane to reduce the need to request the control-plane and increase the scalability.

Another way to resolve the scalability issue is to distribute the control plane tasks to multiple controllers. ONIX [20] delivers a distributed control plane platform while preserving the reliable network-wide state (Fig. 4).

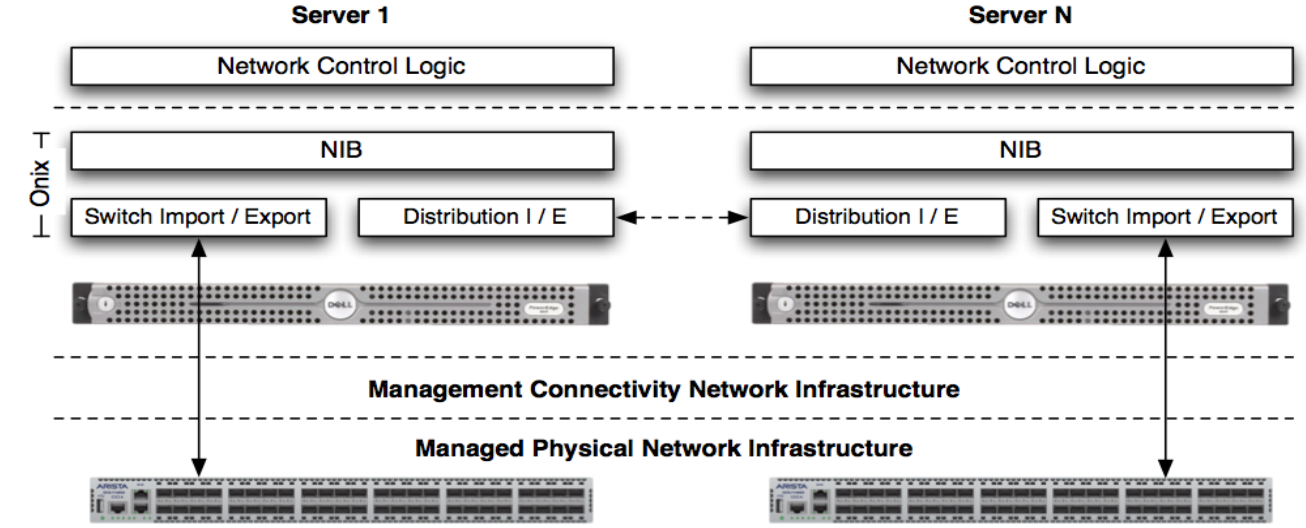

Fig. 4. Two ONIX controller coordinating and their views of the underlying network state

Many other solutions to the scalability issue were presented in the literature such as HyperFlow [21], which consists of a physically distributed and logically centralized control platform. In recent works, Shin et al [22] proposed IRIS-HiSA, a cluster architecture for distributed controller, it's the main objective is to support uninterrupted load balancing and failover with horizontal scalability, as is done in existing work, but one of IRIS-HiSA's distinctive features is to provide transparency between the data plane and the switches. Thus, the switches do not need to know the internal details of the controller cluster, and they simply access the same way a single controller is accessible.

5) Unauthorized controller access

SDN Applications access the controller for a wide range of reasons, it's primordial to ensure that these applications work within their respective perimeter with the legitimate functional requirement. For that, securing the controller from the malicious application must be ensured.

SE-Floodlight [23] controller was proposed as an extended version of the Floodlight controller. SE-Floodlight introduced several security enhancement methods:

- Privilege separation by adding a secure programmable northbound API, this enables SE-Floodlight to work as a truly independent mediator between the application layer and the data plane.
-
- A runtime integrity validator of the modules that generate flow rules. The runtime copy of the local flow-rule producer is compared to the original image installed by the administrator.
- A Rule Reduction (ARR) algorithm that manages inline rule-conflict detection.
- Role-based conflict resolution by comparing the authoritative roles of the producers of the conflicting rule
- PACKET_OUT Control: packet control generated by OpenFlow apps can be blocked by the administrator.
- Security Audit: is a subsystem that tracks all security events

6) DoS attacks

The centralized nature of SDN is revealed to be a single point of breakdown that can be exploited by one of the Internet's most old, high risk and major security threat known as Distributed Denial of Service (DDoS) Attack. A DDoS attack is a dispersed and harmonised attack that begins from multiple network devices. Essentially, the strategy of this attack is to send a huge volume of spoofed IP packets from disparate points in order to make the network resources unattainable to legitimate users. Over recent years, the attackers have got smarter and have been constantly enhancing and using advanced DDoS attack methods to inflict more economical and financial costs.

While it is a standard security issue, DDoS attack mitigation is still a severe threat to almost every technology , Braga et al presented in [24] a detection method that consists of self-organizing maps (SOMs) to identify abnormal/injected flows (Fig. 5).

The proposed solution consists of three modules:

- A flow Collector module that is responsible for gathering flow from switches.
- A feature Extractor module that extracts relevant data distinguishing DDOS attacks.
- A classifier module analyzes extracted data then classifies them as normal or abnormal.

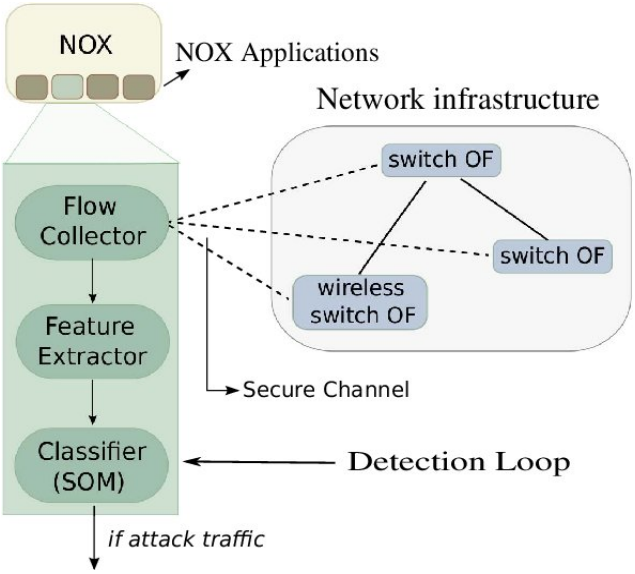

Fig. 5. SOMs detection process

In recent works, Dridi et al. [25] suggested SDN-Guard, a new system that can effectively protect SDN networks against DoS attacks dynamically by potential redirection of malicious traffic, flow timing adjustment and aggregation of flow rules. The solution relays on four module (Flow management module, Rule aggregation module, Monitoring module).

De Assis et al. [26] proposed a stand-alone DoS / DDoS defensive approach for SDN called Game Theory (GT) Game Theory which is an analytical tool that can model negotiation situations and deal with many problems in different areas - Holt-Winters for Digital Signature (HWDS), which combines detection and anomaly identification provided by an HWDS system with a decision-making model based on GT.

Yunhe et al. introduced in [27] a system named Software-Defined Anti-DDoS divided into four modules detection, a Trigger Module, Detection Module, Traceback Module, and Mitigation Module, respectively. These modules correspond to four states:

- **Init State**: It is the primary state of the SD-Anti-DDoS system. A packet_in trigger is used to program attack detection in this state. If no abnormal event was found, it will remain in the Init State. Else, the system will run a Detection State.
- **Detection State**: This state is handling the detection of a DDoS attack. The attack detection will start to verify the existence of a DDoS attack in the network.
- **Traceback State**: If the detection module detects a DDoS attack, the Traceback State will start and the system will try to trace the attack path and the attack origin switch.
- **Mitigation State**: The Mitigation module will try to stop the threat from its source and will clean the infected switch from malicious flow entries.

Fig. 6. Illustrate the four state of the above Anti-DDoS attacks.

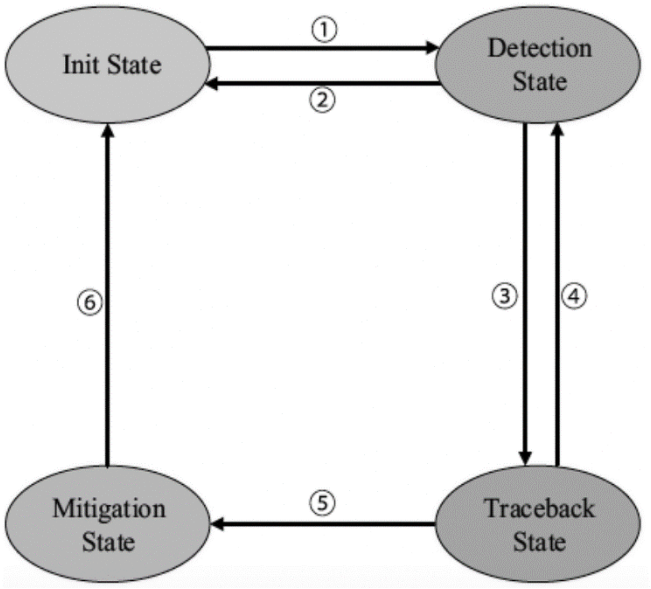

Fig. 6. SD-Anti-DDoS four state diagram

7) Malicious flow rules

Changing the flow rules within switches must be controlled. FortNox platform allows the NOX controller to check flow rules contradiction instantaneously, then decide the generated flow rules. Blocking flow rules insertion request is achieved using a conflict analysis algorithm when a security application adds a flow rule. FortNOX restricts other applications from inserting conflicting flow rules in the same OpenFlow network.

V. OUR PROPOSED MODEL

A. System Model

In previous sections, we presented the most relevant security threats facing the controller from many perspectives. Then we gave some of the solutions proposed in the literature to mitigate these threats. None of the previous solutions has treated the integrity of the controller config data-store. A malicious application or any mischievous person or system that gained access illegitimately can alter the data present within the config data-store. The main goal of the datastore is to hold the actual configuration of the SDN controller and its environment making it a suitable target, losing control over the storage section expose the SDN network-based implementation to several type of risks, from a simple traffic deviation to denial of services.

In this section, we will introduce a significant enhancement to our approach that aims to improve the controller security, especially the flow rules config datastore from false and illicit flow rules modification, insertion and deletion. The Enhanced Solid-Flow module will be implemented in one of the SDN controller and will be designed following this perspective such as the open-daylight [23] controller , Floodlight. Fig 7. shows SDN Floodlight controller architecture. Our module will target the storage section of the Floodlight Controller.

The Enhanced Solid flow was designed using java-programing language and was implemented in two scenarios:

- As module running within the controller
- As a module exposed via a secure rest-api.

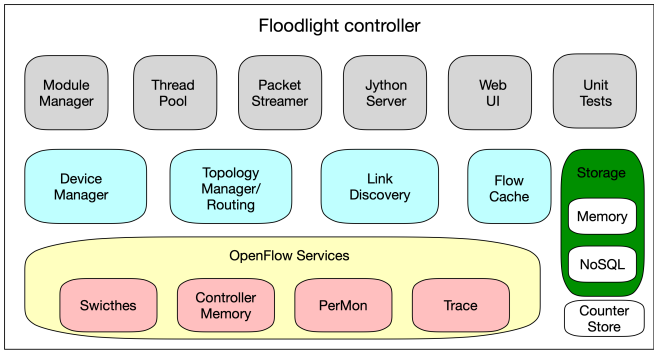

Fig. 7. Floodlight Architecture

As mentioned in the previous paragraph, flow rules are generally generated by two manners, either from SDN application operating on the application plan or by users with system administration rights. To ensure that these flow rules are generated by one of the above trusted methods, we present an enhancement of the "Solid-Flow" module, which will be implemented as a plugin, developed using OSGI framework Equinox. Our plugin will rely on several mechanisms such as SHA 256 or SHA 512 [28] cryptography hashing function to ensure the integrity of the control flow rules present within the controller data-store.

Solid-flow is attached to the controller environment as shown in Fig. 9 (next page), and will run a periodic check after the initiation state on dynamic periodic check calculated on the fly depending on the number of attacks in the previous cycle. Our proposed model exposes its services through a secure rest API, and can run in standalone mode.

Flow rules check will be also triggered in case of newly arrived legitimate flow from any source whether is an SDN application or users with administrative privilege. This event will be triggered to ensure that during the interval between periodic checks, no flow-rule was inserted illegitimately, resulting in more reliable config-data store.

Each flow rule will trigger several process from hash calculation passing through checking the type of flow (periodic or check) then depending on the type the Solid-flow will compare hashes and determine the eligibility of the flow rule then decide what to do with it.

Fig. 8 shows two types of flow rules (legitimate flow-rule and illegitimate flow-rule) sources insertion along the time line and the two types of check, a periodic Solid-flow-check and a check before the insertion in the data-store.

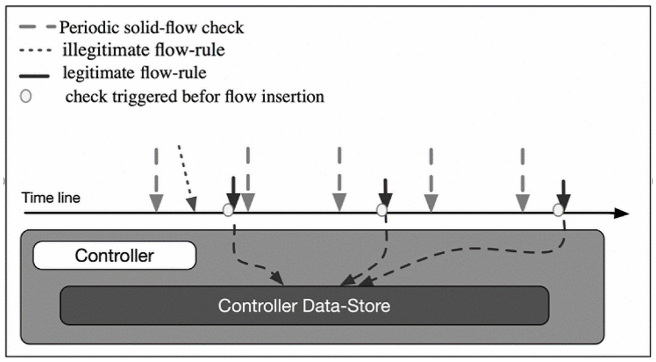

Fig. 8. Attack timeline

Each periodic check will pull a new fresh hash of the flow rules data-store, a hash calculation will be calculated for each flow rules and for the global content of the data store, then it

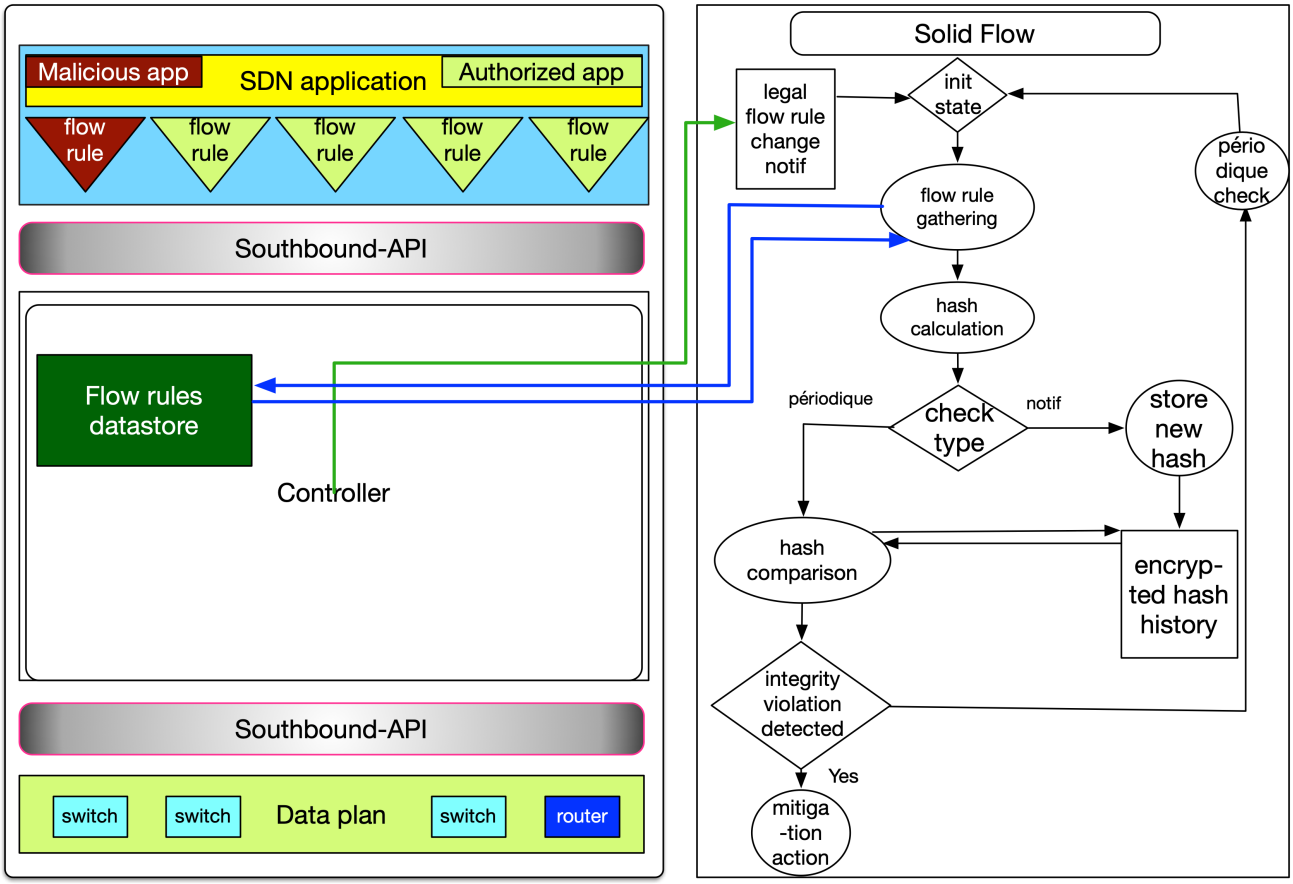

Fig. 9. Enhanced Solid-Flow architecture

will be compared with the previous hash's stored from earlier calculations.

When the controller triggers the Solid-flow mechanism by notification, assuming that this change is legal and made by one of the trusted methods using one of the solutions proposed earlier such as permOF [15], a new hash will be calculated before inserting the new flow rule in the config data-store. Afterward, solid-flow will confirm to the controller that no change has been made to the config data-store. Subsequently, the new flow rule could be inserted securely, and unique hashes will be calculated and encrypted for the next comparison.

Hash comparison will result either in normal config data-store state or in integrity violation. In the case of integrity violation, mitigation actions will be conducted depending on the check type 'periodic, reactive periodic or notification'.

The system will determine with high precision where the illegitimate change has been made

B. Solid-Flow design principals.

Classification hyperplane of training data may be divided by linear classification plane or not via mapping the training data vector to higher dimensional space with some function and transferring the problem to a linear classification problem in that space. After the mapping procedure, SVM finds out a linear separating hyperplane with the maximum margin in the space.

Our module will run during the controller initialization stage. After the initiation phase, the Solid-Flow mechanism will be prompted to run in three scenarios:

Scenario 1: The periodic hash gathering.

Event 1): Initializing runtime environment and checking controller state.
Event 2): Gathering policies information from the controller flow rules data-store.
Event 3): New hash calculation.
Event 4): Hash comparison (between the latest and previous hash).
Event 5): If integrity violation detected, take mitigation action.
Event 5): Periodic check timer initiation.

Scenario 2: Event initiated by the controller in case of new flow rule arrival.

Event 1): Notification form the controller about new flow inserted by a trusted entity and wait for the Sloid-flow confirmation.
Event 2): Gathering policies information from the controller flow rules data-store.
Event 3) New hash calculation.
Event 4): Hash comparison.
Event 5): If integrity violation detected, take mitigation action. Otherwise, permit to the controller to insert new flow-rule.
Event 4): Hash storing (new hash stored for feature comparison).

Scenario 3: Reactive periodic mode hash gathering.

Event 1): Initializing runtime environment and checking controller state.
Event 2): Gathering policies information from the controller flow rules data-store.
Event 3): New hash calculation for each rule and the global config data store.
Event 4): Hash comparison (between the latest and previous hash).
Event 5): If integrity violation detected, take mitigation action.
Event 5): Periodic check timer initiation using the proposed formulate.

The above scenarios are translated into algorithms. We propose 3 algorithms related to each previously described scenario to improve flow rules database integrity inside the SDN controller. All the parameters used in different algorithms are presented in Table III.

Table III
PARAMETERS OF ALGORITHMS

Parameters	Meaning
PT	Periodic Timer
DPT	Default Periodic Timer
RT	Reactive Timer
DHA	Data Store Hash
AHA	Actual Hash
CT(ER)	Number of wrong hash counter
CST	Controller State
CVF	Controller Verification
T_HRMAX	Maximum Threshold (time)
T_HRMIN	Minimum Threshold (time)

Among the parameters, PT, DPT, RT, represent the periodic timer, the default periodic timer, and the reactive timer respectively. DHA and AHA, describe the data store hash and the actual hash. CT(ER), CST and CVF represent the number of the wrong hash counter, the controller state, and the controller verification. T_HRMAX and T_HRMIN represent the maximum threshold and the minimum threshold, respectively. The value of some parameters is set according to the attack model or scenario setting.

For the first scenario, as shown in algorithm 1, we initiate the Periodic Timer PT to a Default Periodic Timer. The Number of wrong hash counter CT(ER) is in state 0. If the Controller State CST is initialized, then the value stored AHA is the same as Data Store Hash DHA. At the beginning of time period DPT, if the stored value Actual Hash AHA is different from that of the Data Store Hash DHA, the Number of the wrong hash counter will be 1, and Take mitigation action.

Algorithm 1: The periodic hash gathering

```
1: Let PT = DPT;
2: Let CT(ER) =0;
3: if CST is initiated then
4:          AHA = DHA;
5: End if
6: At the beginning of time period DPT :
7:          If AHA == DHA then
8:             AHA = DHA;
9:         Else  if AHA != DHA
10:             CT(ER) += 1;
11:         Take mitigation action
12:        End if
13: end of time period DPT restart over
```

For the second scenario, the event is initiated by the controller in case of a new flow rule arrival. In this case, the controller sends a verification about new flow inserted by a trusted entity and waits for the Sloid-flow confirmation. The newly calculated hash called in the algorithm. Actual Hash AHA is compared with the data stored hash DHA. If integrity violation detected, take mitigation action. Otherwise, permit the controller to insert a new flow-rule.

The new hash is stored for feature comparison.

Algorithm 2: Event initiated by the controller in case of new flow rule arrival

```
1: Let PT = DPT;
2: Let CT(ER) =0;
3:  Event:      event  ("controller  ask  for
verification")
4: Action:  (If AHA == DHA then
5:                  AHA = DHA;
6:             Else  if AHA != DHA
7:                  CT(ER) += 1;
8:            Take mitigation action
9;           End if)
10: if CST is initiated then
11:    AHA = DHA;
12: End if
13: At the beginning of time period DPT :
14: Wait-event();
15:       If AHA == DHA then
16:          AHA = DHA;
17:       Else  if AHA != DHA
18:             CT(ER) += 1;
19:          Take mitigation action
20:       End if
21. end of time period DPT restart over
```

For the last scenario, relative to reactive periodic mode hash gathering. We were initializing the runtime environment and checking the controller state. The New hash calculation for each rule and the global config data are calculated and compared (the latest and previous hash). If integrity violation detected, take mitigation action. and the periodic check timer initiation using the proposed formula:

DPT= $\boxed{(Thrmax - Thrmin)e^{-\lambda} + Thrmin}$

Algorithm 3: Reactive periodic mode hash gathering

```
1: Let PT = DPT;
2: Let CT(ER) =0;
3: Event:   event ("controller ask for
verification")
4: Action:  (If AHA== DHA then
5:          AHA=DHA;
6:             Else  if AHA!=DHA
7:           CT(ER)+= 1;
8:             Take mitigation action
9:           End if)
10: if CST is initiated then
11:            AHA=DHA
12: End if
13: At the beginning of time period Dpt :
14: Wait-event();
15:          If    AHA=DHA then
16:                 AHA=DHA;
```

```
17:          Else  if HA!=DHA
18:               CT(ER)  += 1;
19:                  λ =  CT(ER),
20:DPT (Thrmax − Thrmin)e^−λ + Thrmin ;
21: Take mitigation action
22:           End if
23: end of time period DPT restart over
```

V. Performances Analysis and Discussion

We have implemented the enhanced version of SOLID FLOW as a separate program that runs within the controller environment, using java, which is the programing language of all recent Open-Flow controllers.

The proposed solution was tested on a floodlight controller, which is a large-scale network simulator. SOLID-FLOW can be used with any other open-flow controller offering a rest API. The program communicates with the floodlight controller using secured calls to the REST-API based on trust-based TLS/SSL channel. In this section, we will present the use case environment variables and calculate different metrics (System reactivity, The CPU rate, response time to attacks).

The tested use case was conducted using a floodlight controller installed on Ubuntu Linux distribution. The SDN data layer was created using Mininet [29], which is a software emulator for prototyping an extensive network on a single machine.

The Implementation parameters are presented in Table IV.

TABLE IV
IMPLEMENTATION PARAMETERS

label	value
Emulator	Mininet
Number of nodes	3000
Number of OV switches	300
Number of flow rules	1000
Dynamic Periodic timer	Start from 2 seconds

A. System reactivity

In this test, we wanted to demonstrate the reactivity of the system in the reactive mode compared the normal running mode. The conducted test results in a sensitive system that detected the malicious flow rules depending on the checksum fault rate in each flow rules as they acquire and accordingly adjust the periodic timer to match the intensity of the attacks in the previous cycle.

The periodic timer is calculated following the next equation:

$$TP = (Thrmax - Thrmin)e^{-\lambda} + Thrmin$$

Where :

- TP: The periodic timer.
- Thrmin and Thrmax are the minima and maximum threshold of the time interval with which the Solid Flow performs a periodic check.
- λ: the number of errors in the previous cycle

We have set up a limitation to our mechanism to avoid exhaustion of the CPU usage in case of the significant flow of attacks, assuming the system administrator is present to take appropriate action in time, for that we will have two main scenarios:

1. In case of large number of attacks targeting the controller, the periodic verification time will drop dramatically according to the previously cited equation. The performance of the system may deteriorate if this time becomes very low resulting in large load on the CPU hence a high CPU consumption. The Thrmin will allow setting a minimum threshold not to exceed by our system making it more stable.
2. The case where the system knows a much-reduced number of attacks or outright no attacks, Thrmax makes it possible to set a maximum threshold of the verification time, thus guaranteeing a better rate TPR (True Positive Rate). Indeed, with a very long time of verification, the system may not be aware of the attacks that would occur between the verification sequences as mentioned before in Fig. 9.

Fig. 10. demonstrates the reactivity of the system to the number of attacks. As we can observe the more attacks acquire the shorter time become and vice-versa. This reactiveness allow our module to be more agile in response to the amount of acquiring attacks.

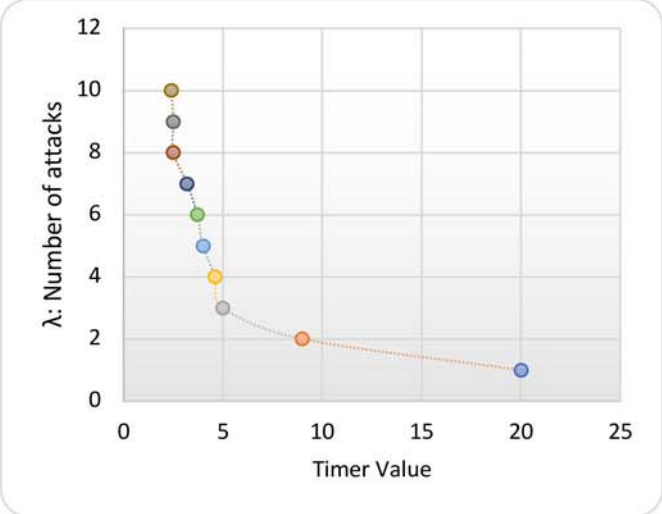

Fig. 10. The system reactivity under attacks

B. The CPU rate used at the SDN controller

Running the Solid-Flow program in parallels with SDN controller does not require any high-consumption of CPU while running it in reactive mode enables us to save 40% of CPU usage. The CPU reduction is due the low rate of check in case of low rate of attacks. Using our module in both mode (within the controller environment and as Res-Api module) rendered almost the same results.

The number of flow rules remain significant to our tests and increase the CPU rate as the sloid flow need to deal with large amount of data.

In Fig. 11, we note that the CPU rate used at the SDN controller level changes linearly with the increase in the number of nodes in the SDN network and remains remarkably low. It should also be noted that improving the time to check data integrity in reactive mode significantly enhances the CPU rate at the controller level, about 40%, allowing better response to potential attacks.

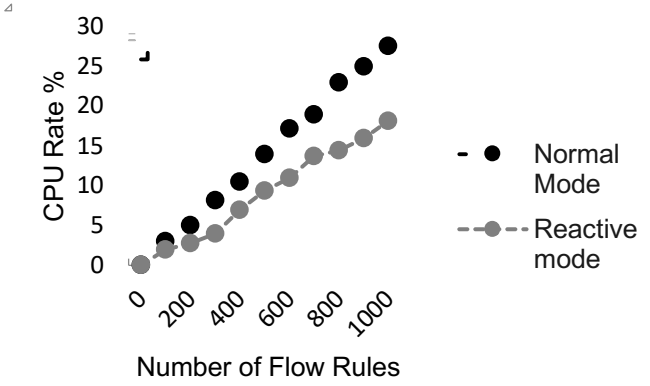

Fig. 11. CPU usage

C. *SDN Controller response time to attacks*

By minimizing requests for data checks using the reactive mode, the SDN controller does not need to be entirely dedicated to processing and responding to periodic checks, which will naturally improve the time of its response to the actual attacks. Fig. 12. clearly shows that the SOLID-FLOW reactive mode effectively makes it possible to have a minimization of the order of 30% of the response time compared to the normal mode.

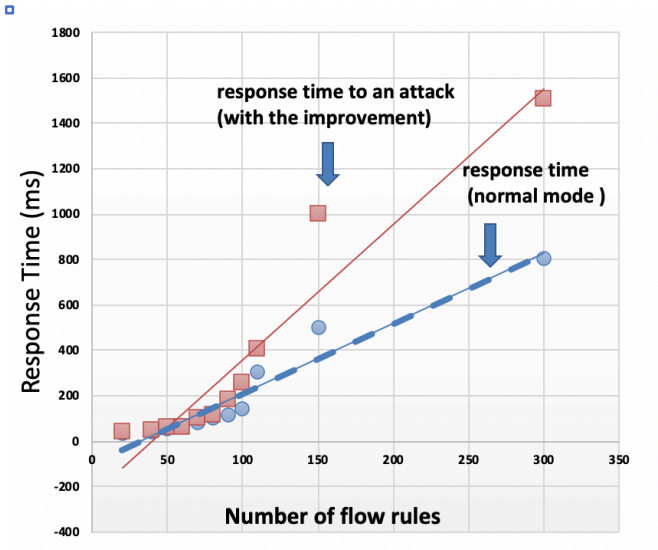

Fig. 12. SDN controller response time to attacks

VI. CONCLUSION AND FUTURE WORK

Currently, SDN technologies attract a lot of interest in the networking industry. However, their deployment exposes the network to unknown risks and still poorly documented.

The SDN networks differ from conventional paradigms, which introduce new security challenges alongside essential practice (although not explicit in standards) such as the use of a dedicated management network, the establishment of an authentication solution, and integrity as TLS, or equipment redundancy in charge of routing. In this paper, we gave state of the art on the various security threats targeting the control layer and challenges in SDN, and the multiple countermeasures proposed. We also introduced a Solid-Flow module for ensuring the integrity of the data-store.

We have implemented the Solid-Flow module and tested its behavior regarding deferent aspects; the test result was encouraging, as shown in the previous section. In our next work, we envisage investigating the mitigation action section resulting in a complete trusted config data-store in SDN controllers and correct mitigation behavior.

REFERENCES

[1] Nick McKeown, Tom Anderson, Hari Balakrishnan, Scott Shenker and Jonathan Turner, "OpenFlow: Enabling Innovation in Campus Networks", ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69-74, April 2008.

[2] Youssef Qasmaoui and Abdelkrim Haqiq, "Solid-flow: A flow rules security mechanism for SDN," in the Proceedings of the 3rd International Conference of Cloud Computing Technologies and Applications, CloudTech 2017, Rabat, Morocco, 24-26 Oct. 2017 (Published in IEEE Xplore: DOI: 10.1109/CloudTech.2017.8284734, 08 February 2018).

[3] Yawar Abbas Bangash, Qamar ud Din Abid, Alshreef Abed Ali A and Yahya E. A. Al-Salhi, "Security Issues and Challenges in Wireless Sensor Networks: A Survey", IAENG International Journal of Computer Science, vol. 44, no.2, pp135-149, 2017.

[4] Z. Yan, P. Zhang and A. V. Vasilakos, "A security and trust framework for virtualized networks and software-defined networking", Secur. Commun. Networks, vol. 9, no. 16, pp. 3059–3069, 2016.

[5] A. Mateen, Q. Zhu, S. Afsar and S. A. Sahil, "Effect of Encryption Delay on TCP and UDP Transport Layer Protocols in Software Defined Networks (SDN)", in the Proceedings of the International MultiConference of Engineers and Computer Scientists, Hong kong, Hong kong, 13-15 March, 2019.

[6] Dongkyun Kim, Yong-Hwan Kim, Chanjin Park and Kyu-Il Kim, "KREONET-S: Software-Defined Wide Area Network Design and Deployment on KREONET", IAENG International Journal of Computer Science, vol. 45, no.1, pp. 27-33, 2018.

[7] M. C. Dacier, H. König, R. Cwalinski, F. Kargl and S. Dietrich, "Security Challenges and Opportunities of Software-Defined Networking", IEEE Secur. Priv., vol. 15, no. 2, pp. 96–100, 2017.

[8] Yunhe Cui, Lianshan Yan, Saifei Li, Huanlai Xing, Weike Pan, J. Zhu and Xiao-Yang Zheng, "SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks", Journal of Network and Computer Applications, vol. 68, pp. 65-79, June 2016.

[9] C. Cremers, M. Horvat, S. Scott and T. V. D. Merwe, "Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication," in the Proceedings of the 37th IEEE Symposium on Security and Privacy (SP), pp. 470–485, Fairmont, San Jose, California, USA, May 23-25, 2016.

[10] C. Lorenz , D. Hock , J. Scherer, R. Durner, W. Kellerer , S. Gebert , N. Gray ,T. Zinner and P. Tran-Gia, "An SDN/NFV-Enabled Enterprise Network Architecture Offering Fine-Grained Security Policy Enforcement," IEEE Commun. Mag., vol. 55, no. 3, pp. 217–223, 2017.

[11] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story and David Walker, "Frenetic: A Network Programming Language", ACM SIGPLAN Notices, vol. 46, no. 9, September 2011.

[12] A. Voellmy, H. Kim" and N. Feamster, Procera: A language for high-level reactive network control", in the Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks (HotSDN'12), 10.1145/2342441.2342451. 2012.

[13] Christopher Monsanto, Nate Foster, Rob Harrison, and David Walker, "A Compiler and Run-time System for Network Programming Languages", in the Proceedings of the 39th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages: POPL'12, Philadelphia, PA, USA, January 25–27, 2012.

[14] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, "Towards a secure controller platform for openflow applications", in the Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, New York, USA, pp. 171-172 , 2013.

[15] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Model checking invariant security properties in OpenFlow", in the Proceedings of the 2013 IEEE International Conference on Communications (ICC), Budapest, Hungary, pp. 1974-1979, 2013.

[16] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson and G. Gu, "A security enforcement kernel for OpenFlow networks", in the Proceedings of the first workshop on Hot topics in Software Defined

Networks (HotSDN'12), pp. 121–126, Helsinki, Finland, August 13, 2012.
[17] M. Wang, J. Liu, J. Chen, X. Liu and J. Mao, "PERM-GUARD: Authenticating the Validity of Flow Rules in Software Defined Networking", in the Proceedings of the 2nd IEEE Int. Conf. Cyber Secur. Cloud Comput. CSCloud 2015 - IEEE Int. Symp. Smart Cloud, IEEE SSC New York, USA, no. 37, pp. 127–132, 2016.
[18] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martin Casado, Nick McKeown and Scott Shenker, "NOX: Towards an operating system for networks", ACM SIGCOMM Computer Communication Review", vol. 38, no. 3, pp. 105-110, July 2008.
[19] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, and V. Vercellone, "Scalability of ONOS reactive forwarding applications in ISP networks", Comput. Commun., vol. 102, pp. 130–138, 2017.
[20] A. Tootoonchian and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow", in the Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, Berkeley (INM/WREN'10), CA, USA , April 2010.
[21] J. Shin, T. Kim, B. Lee and S. Yang, "IRIS-HiSA: Highly Scalable and Available Carrier-Grade SDN Controller Cluster", Journal of Mobile Networks and Applications, vol. 22, pp. 894–905, 2017.
[22] I. Ahmad, S. Namal, M. Ylianttila and A. Gurtov, "Security in Software Defined Networks: A Survey", IEEE Commun. Surv. Tutorials, vol. 17, no. 4, pp. 2317–2346, 2015.
[23] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow", in Ithe Proceedings of the EEE Local Computer Network Conference, Denver, Colorado , USA, pp. 408–415, 2010.
[24] L. Dridi and M. F. Zhani, "SDN-Guard: DoS Attacks Mitigation in SDN Networks," in the Proceedings of the 5th IEEE International Conference on Cloud Networking (Cloudnet), Pisa, Italy, pp. 212–217, 2016.
[25] M. V. O. De Assis, A. H. Hamamoto, T. Abrão, and M. L. Proença, "A Game Theoretical Based System Using Holt-Winters and Genetic Algorithm With Fuzzy Logic for DoS/DDoS Mitigation on SDN Networks", IEEE Access, vol. 5, pp. 9485–9496, 2017.
[26] Martin Fong, Phillip Porras, Keith Skinner and Vinod Yegneswaran, "Securing the Software-Defined Network Control Layer", in the Proceedings of the 2015 Network and Distributed System Security Symposium (NDSS), San Diego, California, February 8-11, 2015.
[27] S. Gueron, "Speeding Up SHA-1, SHA-256 and SHA-512 on the 2nd Generation Intel® CoreTM Processors," in the Proceedings of the Ninth International Conference on Information Technology - New Generations, Las Vegas, USA, pp. 824–826, 2012.
[28] De Oliveira, R. L. S., Schweitzer, C. M., Shinoda, A. A. and Prete, L. R, "Using mininet for emulation and prototyping software-defined networks", in the Proceedings of the IEEE Colombian Conference on Communications and Computing (COLCOM), Bogota, Colombia, pp. 1-6, 2014.

[1] **Youssef Qasmaoui** received his Master degree in Networks and IT Security and his Bachelor degree in Networks and IT Systems, respectively in 2012 and 2010 from the Faculty of Sciences and Techniques (FST), Settat – Morocco. He also received a second Master degree in Information System Engineering at the University of Western Brittany at Brest – France. He is also doing his Ph.D. thesis at the FST, Settat - Morocco. His research interests include Software Defined Networks, Virtual Laboratory and Networks Security.

[2] **Prof. Abdelkrim HAQIQ** has a High Study Degree (Diplôme des Etudes Supérieures de troisième cycle) and a PhD (Doctorat d'Etat), both in the field of modeling and performance evaluation of computer communication networks, from Mohammed V University, Faculty of Sciences, Rabat, Morocco. Since September 1995 he has been working as a Professor at the department of Applied Mathematics and Computer at the Faculty of Sciences and Techniques, Settat, Morocco. He is the Director of Computer, Networks, Mobility and Modeling laboratory: IR2M. He is an IEEE senior member and an IEEE Communications Society member. He is also a member of Machine Intelligence Research Labs (MIR Labs), Washington, USA. He was a co-director of a NATO Multi-Year project entitled "Cyber Security Analysis and Assurance using Cloud-Based Security Measurement system", having the code: SPS-984425. Prof. Abdelkrim HAQIQ's interests lie in the areas of modeling and performance evaluation of communication networks, mobile communications networks, cloud computing and security, emergent technologies, Markov chains and queueing theory, Markov decision processes theory, and game theory. He is the author and co-author of more than 170 papers (international journals and conferences/workshops). He supervised 15 PhD thesis and co-supervised 3 PhD thesis. Actually, he is supervising and co-supervising other PhD thesis. He is an associate editor of the International Journal of Computer International Systems and Industrial Management Applications (IJCISM), an editorial board member of the International Journal of Intelligent Engineering Informatics (IJIEI) and of the International Journal of Blockchains and Cryptocurrencies (IJBC), an international advisory board member of the International Journal of Smart Security Technologies (IJSST) and of the International Journal of Applied Research on Smart Surveillance Technologies and Society (IJARSSTS). He is also an editorial review board of the International Journal of Fog Computing (IJFC) and of the International Journal of Digital Crime and Forensics (IJDCF). Prof. Abdelkrim HAQIQ was a chair and a technical program committee chair/member of many international conferences and scientific events. He was also a Guest Editor and Co-Editor of special issues of some journals, books and international conference proceedings. From January 1999 to December 1999 he had a Post-Doctoral Research appointment at the department of Systems and Computers Engineering at Carleton University in Canada. He also has held visiting positions at the High National School of Telecommunications of Paris, the Universities of Dijon, Versailles St-Quentin-en-Yvelines and LAAS CNRS, Toulouse in France, the University of Ottawa in Canada, the FUCAM in Belgium, the National Engineering School of Sfax, Tunisia, the University of Naples Federico II, Italy and the University of Algarve, Portugal.