OpenFlowSIA: An optimized protection scheme for software-defined networks from flooding attacks

Conference Paper · July 2016

DOI: 10.1109/CCE.2016.7562606

OpenFlowSIA: An Optimized Protection Scheme for Software-Defined Networks from Flooding Attacks

Trung V. Phan ‡, Truong Van Toan †, Dang Van Tuyen ‡, Truong Thu Huong ‡,*, Nguyen Huu Thanh ‡
† Hanoi Open University, Vietnam
‡ Hanoi University of Science and Technology, Vietnam
Email: *20102384@set.hust.edu.vn*, *huong.truongthu@hust.edu.vn*

***Abstract*—In this paper, we propose an optimized protection mechanism (OpenFlowSIA) for Software-Defined Networks from flooding attacks (Distributed Denial-of-Service) based on Support Vector Machine and our proposed algorithm called the Idle−timeout Adjustment (IA). Our methodology not only utilizes SVM advantages in classification such as high accuracy and little processing time, but also applies effectively the IA algorithm and coherent policies to protect network from resource exhaustion caused by flooding attacks, particularly for the SDN controller and OpenFlow switches. Through comprehensive experiments, the OpenFlowSIA scheme illustrates that it can be an innovative solution to secure and save the network resources under flooding attacks in the Software-Defined Networks.**

***Index Terms*—Distributed Denial-of-Service, Support Vector Machine, Software-Defined Networks.**

I. Introduction

Software-Defined Networks (SDN) [1] has rapidly burst as a new promising network framework in the future. By decoupling the control plane from the data plane, the SDN brings about numerous advantages in network management and control in comparison with conventional networks. As an essential element of the SDN, OpenFlow protocol [2] is responsible for communication between a SDN controller (the control plane) and OpenFlow switches (the data plane). This protocol enables the SDN controller to approach OpenFlow switches for applying rules and collect network information. Hence, the SDN architecture is believed as the future internet technology because of its benefits and advantages.

However, the SDN model still causes the vulnerability to flooding attacks (Distributed Denial-of-Service). The original idea of flooding attacks is sending a massive network traffic generated from botnets to a victim and this kind of network attack is one of the most difficult challenges which all network systems have to meet in the field of network security nowadays [3], [4]. With regard to the SDN architecture, when the network system is under flooding attacks, both the SDN controller and OpenFlow switches have to suffer from resource exhaustion because of dramatically increases in number of flows and transferring packets created by botnets. Accordingly, the SDN controller and OpenFlow switches may stop working, this leads to whole network system is broken down.

A wide range of methods [5] have been proposed to diminish flooding attack effects in the conventional networks. Meanwhile, in the SDN environment, because the SDN controller is a network brain; thus, several approaches based on machine learning or neural network have been recommended to handle flooding attacks. Kokila RT *et al*. [6] applied the usage of Support Vector Machine (SVM) to DDoS detection and analysis in SDN with a higher accuracy in classification than others. Braga *et al*. [7] created a DDoS detection scheme using Self-Organizing Map (SOM) with 4 and 6 tuples of attributes. Trung *et al*. [8] also introduced the usage of Fuzzy Interference System into DDoS attack prevention in the SDN. In addition, AVANT-GUARD [9] is provided to migrate the bottleneck problem from saturation attacks. FLOOD-GUARD [10] provides two modules: proactive flow rule analyzer and packet migration to preserve network policy enforcement and protect the controller from being overloaded. In summary, all previous works have the same purpose of dealing with flooding attacks. In this work, we propose an optimized protection scheme that handle not only flooding attacks but also resource exhaustion of SDN components.

The rest of the paper is arranged as follows. Section II describes background knowledge. Next, The proposed OpenFlowSIA Scheme will be presented in Section III. Afterwards, Section IV provides a full implementation in SDN environment of the OpenFlowSIA. Performance Evaluation will be then given in Section V. Finally, conclusion and future work are mentioned in Section VI.

II. BackGround

This section first depicts an overview of Software-Defined Networks (SDN) and OpenFlow Protocol. Next, the detail of Support Vector Machine algorithm and a general summary of flooding attacks (DDoS) in SDN are mentioned.

A. SDN and OpenFlow Protocol

Software-Defined Networks (SDN) has been believed as an innovative network model recently with the separation of the control plane from the data plane as in Fig. 1. As a brain of network, the SDN controller (the control plane) controls and monitors whole network activities through communication with OpenFlow switches (the data plane) and network applications such as routing, forwarding services, topology management and etc. The communication between two planes is defined by OpenFlow protocol [2], in which a secure connection (SSL) is established for authentication and message

* Corresponding author

978-1-5090-1801-7/16/$31.00 ©2016 IEEE

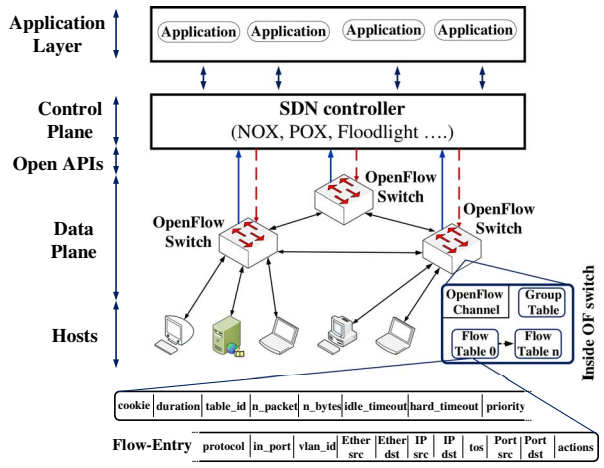

Fig. 1. The SDN Architecture and OpenFlow Protocol

exchange. This protocol allows the SDN controller access to flow-tables in the OpenFlow switches with the purpose of flow configuration and management. Therefore, the SDN architecture is considered as an new network framework which brings about numerous advantages in network control and management.

As seen in Fig. 1, the flow-entry maintained in an Openflow switch includes several information such as: duration, table-id, packet number, byte number, protocol, source and destination IP address, service port and etc. By using statistical messages, the SDN controller can collect the flow information from a specific flow-table; hence, statistics and network management in SDN are easier if compared to conventional networks.

B. Support Vector Machine

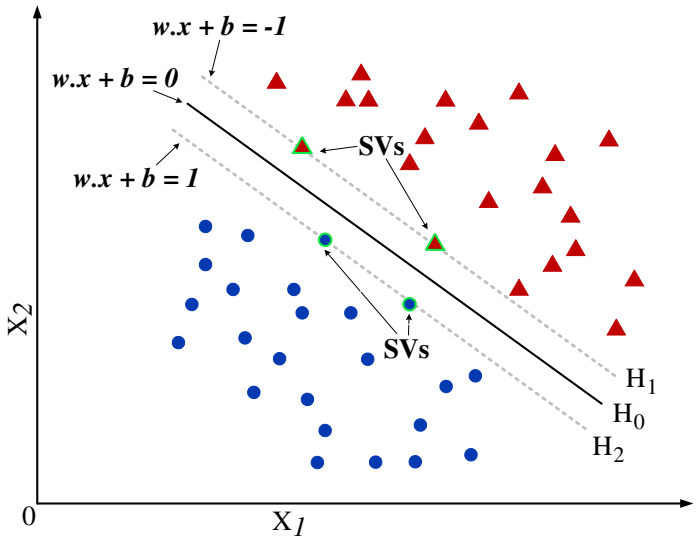

Fig. 2. The Support Vector Machine representation

Given a data set for training $S = \left\{\left(\vec{x}_1, y_1\right), \left(\vec{x}_2, y_2\right), \ldots, \left(\vec{x}_N, y_N\right)\right\}$, where $\vec{x}_i \in \mathbb{R}^n$ and $y \in \{+1, -1\}$. As Fig. 2 shows that the SVM learning algorithm tries to find an optimal hyperplane ($\mathscr{H}_0$) $\left\{\vec{x} \in \mathbb{R}^n : \left\langle \vec{w}, \vec{x} \right\rangle + b = 0,\ \vec{w} \in \mathbb{R}^n,\ b \in \mathbb{R}\right\}$ which separates maximally two data groups using below function:

$$f\left(\vec{x}\right) = \text{sign}\left(\left\langle \vec{w}, \vec{x} \right\rangle + b\right). \tag{1}$$

In this work, two linearly separable data is considered; thus, the optimal hyperplane is performed such that

$$y_i\left(\left\langle \vec{w}, \vec{x} \right\rangle + b\right) \geq 1, \qquad i = 1, \ldots, N. \tag{2}$$

Accordingly, the optimal problem transfers to minimize $\|w\|$ while still meets the condition in (2). To perform this optimization, the Primal problem [11] must be solved. It is true that if $\|w\|$ is altered by $\left(\frac{1}{2}\right)\|w\|^2$, they are the same thing; hence, it now turns to minimizing $\left(\frac{1}{2}\right)\|w\|^2$. Fortunately, this substitution is not different from the original problem and it becomes a quadratic programming optimization problem [11] illustrated as

$$\begin{aligned} &\arg\min_{w,b} \quad \frac{1}{2}\|w\|^2 \\ &\text{s.t.} \quad y_i\left(\left\langle \vec{w}, \vec{x} \right\rangle + b\right) \geq 1, \qquad i = 1, \ldots, N \end{aligned} \tag{3}$$

By applying the Lagrange Multiplier α_i [11], the above constrained problem is expressed as

$$\arg\min_{w,b}\max_{\alpha \geq 0} \quad \left\{\frac{1}{2}\|w\|^2 - \sum_{i=1}^{N}\alpha_i\left[y_i\left(\left\langle \vec{w}, \vec{x} \right\rangle + b\right) - 1\right]\right\}. \tag{4}$$

To set up the optimal hyperplane, a saddle point must be found out. To do this, points that satisfy $y_i\left(\left\langle \vec{w}, \vec{x} \right\rangle + b\right) - 1 > 0$, are not numbered because the corresponding α_i is set to zero. Consequently, a standard quadratic programming technique is an effective solution to solve this problem. The solution can be conducted by referring to the stationary Karush-Kuhn-Tucker condition [11] and the saddle point is finally expressed as a linear combination of training vectors

$$\vec{w} = \sum_{i=1}^{N}\alpha_i y_i \vec{x}_i\,. \tag{5}$$

There are only some points having α_i greater than zero, these points lie on two margin lines ($\mathscr{H}_1$, $\mathscr{H}_2$) and satisfy $y_i\left(\left\langle \vec{w}, \vec{x} \right\rangle + b\right) = 1$. From this condition, the value of offset b can be define based on the average of all support vectors, $\mathscr{N}_{sv}$. The b offset can be calculated as

$$b = \frac{1}{\mathscr{N}_{sv}}\sum_{i=1}^{\mathscr{N}_{sv}}\left(y_i - \left\langle \vec{w}, \vec{x} \right\rangle\right). \tag{6}$$

The Linear Support Vector Machine is judged as a classifier possessing the highest accuracy in the filed of machine learning. Its application are common in a wide range of researches, especially network security. In this paper, this is the reason why we choose the Support Vector Machine to be our classifiers in detecting abnormal flows. For more information, interested readers can refer to [12].

TABLE I
COMMON FLOODING ATTACKS IN SOFTWARE-DEFINED NETWORKS

Type I	Type II
ICMP flooding attack	TCP SYN attack
IGMP flooding attack	UDP flooding attack
Smurf attacks	PUSH + ACK attack
Fraggle attacks	Malformed Packet attack

C. Flooding attacks in Software-Defined Networks

Before considering flooding attacks in SDN we first pay attention to conventional networks. In these architectures, there are a wide range of flooding attack methods and we can summarize two major of flooding attacks : bandwidth depletion and resource depletion attacks. With regard to the bandwidth depletion attacks, proposed methods aim to flood a victim with unwanted traffic which prevents legal traffic from accessing the victim network. For example, UDP, ICMP flooding attacks or amplification attacks including Smurf and Fraggle attacks. In terms of the resource depletion attacks, attackers send IP packets, which are malformed or misuse network protocol communications. It leads to the network resource exhaustion with which legitimate users are not able to access to the network. TCP SYN attack can be a good example which exploits the three-handshake process between sender and receiver before data packets are transmitted. Similarly, Malformed Packet attack tends to send wrong formed IP packets to the victim with the purpose of crashing the victim network system.

From the perspective of a flow-based network architecture such as SDN, we also can summarize and classify flooding attacks into two main types. Type I is defined as follow: a source IP address generates only one or two flows to the victim but the volume of packets or data in each flow is always at high level such as ICMP flooding attack, Smurf and Fraggle attacks. Type II bases on the increase of flow number to wreck the victim network system. The basic idea is that a source IP address can open a huge number of connections to the victim system. A good illustration is TCP SYN attack, a spoofed IP address sends out hundreds of requests to a target Web server; thus, this attack requires OpenFlow switches to add tones of flows in their flow-tables when the system is under attack. This not only makes the victim, but also network devices such as OpenFlow switches or the SDN controller to be crashed. In conclusion, we can look at Table I which depicts our survey of common flooding attacks in Software-Defined Network environment.

III. THE PROPOSED OPENFLOWSIA SCHEME FOR FLOODING ATTACKS IN SOFTWARE-DEFINED NETWORKS

This section gives readers a detail of the proposed OpenFlowSIA scheme and the IA algorithm. The workflow and the internal modules will be provided hereinafter.

As depicted in Figure 3, the OpenFlowSIA consists of five major modules: Flow Collector, Feature Extractor, SVM-*i*, Policy Enforcement and IA Algorithm. The workflow of OpenFlowSIA mechanism includes continuous processes. The Flow Collector first collects flow information from OpenFlow

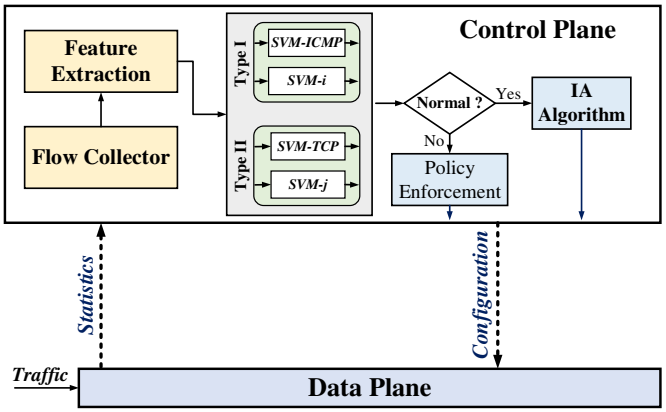

Fig. 3. The proposed OpenFlowSIA scheme

switches, next these information will be extracted by the Feature Extractor. Afterward, features of flow are processed by the corresponding SVM-*i* and the next step bases on the SVM-*i* output. If the output is normal, the flow information will be forwarded to the IA algorithm module to process. Otherwise, the flow is sent to the Policy Enforcement module. The above procedures describe an overview of the OpenFlowSIA. We go deeply into modules hereafter.

A. Flow Collector

The Flow Collector module runs in the SDN controller and in which it sends and receives statistical messages from OpenFlow switches in a predetermined period of time (*StatsTime*).

B. Feature Extractor

The Feature Extractor module is responsible for feature extraction of flow from the former module outputs. Regarding to the features, we extract two features from the flow: *packet number* and *duration*. The first feature indicates the number of packets which is transmitted via the flow; meanwhile, the second shows how long the flow has existed in a flow-table.

C. SVM-i Module

In our proposed scheme, the SVM-*i* is a classifier for each type of network protocol such as TCP, UDP, ICMP or ARP. For example, if *i* is *TCP*, TCP flows will be forwarded to the SVM-*TCP*. As mentioned in the Section II, the SVM-*i* used a dataset for training procedure, then it produces a data distribution and draws the hyperplane, support vectors as shown in Fig. 2. In OpenFlowSIA approach, we set as follow: X_1 axis is assigned to the duration and the packet number belongs to X_2 axis.

D. Policy Enforcement Module

As discussed in the Section II, there are two common types of attacks nowadays. Firstly, attackers tend to send a large number of flows and each flow consists of a absolutely high packet number. To tackle this attack type, we apply a policy with a drop action for the attack flow; thus, the victim or inside

networks are not affected by these abnormal flows. Secondly, flooding attackers generate a vast number of flows using fake source IP addresses and each flow transmits only from 1 to 3 packets. Our solution for this attack is sending messages to OpenFlow switches to delete these flows from flow-tables. Drop or delete actions aim to ensure the robustness of victims, OpenFlow switches and the SDN controller.

Algorithm 1 The Idle−timeout Measurement Algorithm

$N \leftarrow$ Number of flow entries in each Loop;
$Flag \leftarrow$ The SVM-i output of the considering flow;
$Type \leftarrow$ Traffic type: 1, 2;
$StatsTime \leftarrow$ The interval time of statistics;
$IniVal \leftarrow$ The initial Idle-timeout value;
$CurVal \leftarrow$ The current Idle-timeout value;
$SetVal \leftarrow$ The set Idle-timeout value;
$MaxVal \leftarrow$ The maximum Idle-timeout value;
loop
  for $i = 1$ to N **do**
    if $(Flag == Attack)$ **then**
      if $(Type == 1)$ **then**
        Call $SetToZezo$; {The function sets the flow Idle-timeout to 0}
      end if
      if $(Type == 2)$ **then**
        Call $DeleteFlowEntry$; {The function deletes the flow from Flow Table}
      end if
    else
      do nothing
    end if
    if $(Flag == Normal)$ **then**
      if $(CurVal < MaxVal)$ **then**
        if $(CurVal == IniVal)$ **then**
          $SetVal \leftarrow (CurVal + StatsTime)$;
          Call $SetIdletimeout$; {The function sets the flow Idle-timeout by $SetVal$}
        else
          $SetVal \leftarrow MaxVal$; {The flow is classified as a normal second time}
          Call $SetIdletimeout$;
        end if
      else
        do nothing
      end if
    else
      do nothing
    end if
  end for
end loop

E. The Idle-timeout Adjustment Algorithm

In case incoming traffic is detected legitimately, Idle-timeout Adjustment (IA) Algorithm will be applied to enhance the robustness of main network components in Software-Defined Networks. Our main idea in designing the IA algorithm is adjusting a flow idle-timeout based on the SVM-i outputs. Whenever a new flow-entry is installed in a flow-table, we set the idle-timeout value of the flow by an initial value (*InVal*) which is always less than a maximum value (*MaxVal*). The *MaxVal* is normally the default value of the SDN controller, for example, POX controller is 10 seconds. At the next stage, when the SDN controller gathers the flow information and processes the corresponding SVM-*i* of the flow. At the SVM-*i*, if the flow is classified as an attack flow (*Flag* is Attack), the flow will be transferred to the Policy Enforcement module. At the Policy Enforcement module, we check the type of attack (see II-C), if the attack belongs to type I (*Type* is 1), a *SetToZero* function will be called to set the idle-timeout value of the flow to 0, this policy drops any packets passing through the flow. If the attack is type II, the attack flow becomes an input of a *DeleteFlowEntry* function, this function deletes the flow from flow-tables. In the case, the flow is believed as a normal flow, then the flow idle-timeout value will be defined by a *SetVal*. We decide the *SetVal* by comparing the current idle-timeout of the flow (*CurVal*) with the *IniVal*. If the *CurVal* is equal to the *Inival* which indicates that the flow has been just added in flow-tables and this is the first time of consideration, the *SetVal* is set by the summary of the *Curval* and the *StatsTime*, then a *SetIdletimeout* function is called to adjust the flow idle-timeout value. Otherwise, the *SetVal* will be the *MaxVal* because the flow was classified as an normal flow once; hence, we increase the flow idle-timeout to the default value. This increase of the idle-timeout value not only allows normal flows exist in flow-table in a longer time than anomaly flows, but also ensures abnormal flows will be applied policies as soon as detected. The process is repeated as a loop for each time when the SDN controller receives flow statistics. The detail of the IA algorithm is explained in Algorithm 1 by Pseudocode.

IV. Experiments

This section provides readers with a detailed OpenFlowSIA implementation in Software-Defined Networks to deal with flooding attacks. In addition, we introduce to readers a network traffic dataset - CAIDA [13], [14] and a flooding attack tool - BoNesi [15] which are used for SVM training and generating real traffic, respectively. Our implementation topology showed in Fig. 4 includes a POX controller, an OpenvSwitch, a Server, legitimate users, botnets and the Internet connection.

A. CAIDA Datasets

SVM training samples are extracted from CAIDA Datasets [13], [14] (Normal traffic - on 21*st* May, 2015 and DDoS attack traffic - on 04*th* August, 2007). These datasets are collected from diverse ocations worldwide with real network traffic (Web, FTP, Ping and etc). Table II describes the proportion of different protocols of two CAIDA datasets. We can see ICMP and TCP protocols make up the highest proportion in attack and normal datasets, respectively; therefore, in our experimentation we pay attention to two kinds of attacks: ICMP flooding (Type I) and TCP SYN flooding (Type II). After the extraction sample process, we train for SVM-*ICMP*

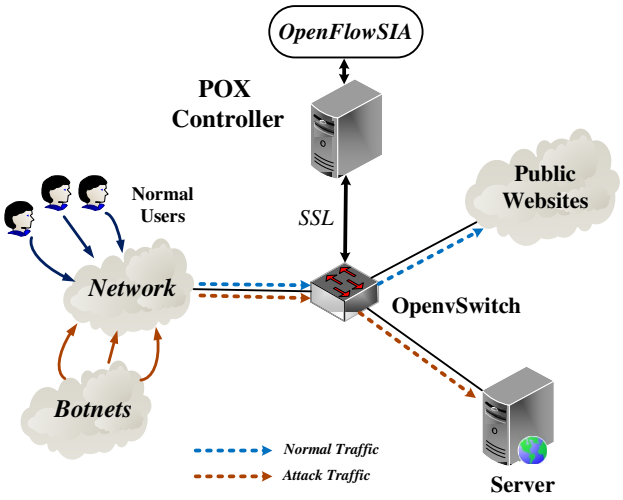

Fig. 4. The OpenFlowSIA implementation in SDN

(Type I) and SVM-*TCP* with the quantity of samples as illustrated in Table III.

TABLE II
CAIDA DATASETS

Traffic State	ICMP (%)	TCP (%)	Others (%)
Normal	6.0	88.45	5.55
Attack	91.25	7.58	1.17

TABLE III
SVM TRAINING SAMPLES

Traffic Type	SVM-*ICMP*	SVM-*TCP*
Normal	5000	5000
Attack	5000	5000

B. Flooding Attack Tool

We launch a flooding attack tool named BoNeSi [15]. This tool simulates many types of flooding attacks using preset botnets to attack a victim or a target network. With regard to our proposed mechanism, the BoNeSi tool is considered as a powerful tool to generate real flooding attacks.

C. Testing Process

In the testing produce, we experiment on five separate test cases under both normal traffic and a combination of ICMP and TCP SYN flooding attacks in 5 minutes. In the first case, we only use SVM-*i* modules to detect attack flows. Meanwhile, the second case is carried out with SVM-*i* modules and a defense module (*PolicyEnforcement*) to diminish attack effects. Three remaining cases, we all apply the proposed OpenFlowSIA scheme and each test uses a different initial Idle-timeout values (*IniVal*) 7, 5 and 3 seconds, respectively. Besides, we set the interval time of statistics (*StatsTime*) to 3 seconds and generate a same number of flow for all cases.

V. PERFORMANCE EVALUATION

In this section, we provide readers with in-depth analyses of the proposed OpenFlowSIA and other test-cases in number of flows, CPU usage of the SDN controller and the OpenvSwitch.

A. Flow Number Comparison

Figure 5 shows the quantity of flow existing in flow-tables of the OpenvSwitch under flooding attacks for mentioned five cases. There is a considerable difference between the SVM case having non-defense mechanism and others. We can see that our proposed schemes provide a greater performance if compared to SVM and SVM+Defense cases, in which the number of flow is diminished significantly, while the operation of legal flows is still guaranteed. To be more detailed, the SVM test posses a range from around 4000 to just under 6000 flows and a better case is the SVM+Defense with approximately 1700 flows in the switch's flow-tables. Meanwhile, the OpenFlowSIA cases always contain a lower flow number on average, for example, the OpenFlowSIA with the smallest *IniVal* value (3 seconds) has only around 1000 flows. The decrease is in direct proportion to the value of *IniVal*, this is because the shorter Idle-timeout makes attack flow's life to be shorter; therefore, there are less flow existing in flow-table at the same time. Besides, normal flows are passed through the IA module to make their longer maintenance.

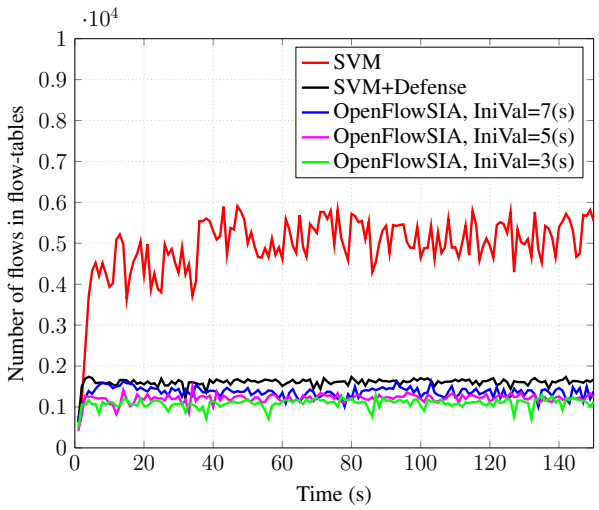

Fig. 5. The number of flows in flow-tables of the OpenvSwitch

B. CPU Usage Comparison

1) SDN Controller's CPU Usage Comparison: The controller's CPU usage is considered as a main criterion to assess the OpenFlowSIA performance in comparison with other schemes.

Figure 6 indicates the usage of CPU for five scenarios. Overall, it is easily to realize a better performance of our proposed method if compared to the SVM and SVM+Defense cases. The level of the controller's CPU expenditure in the SVM case is always the highest value, around 52.0% on average, which is followed by the SVM+Defense scenario just above 42.0% on average when the system is under flooding

attacks. The use of CPU in the OpenFlowSIA cases are minimizing remarkably, with the *IniVal* value are 7, 5 and 3 seconds, the average CPU consumption are 37.0%, 31.5% and 28.5%, respectively. Hence, the OpenFlowSIA saves 25%-33.5%, 5%-13% CPU resource in comparision with the SVM and SVM+Defense cases, respectively. This resource saving can be explained based on the diminution of the flow number in the OpenvSwitch. Thus, both the Flow Collector and Feature Extraction modules have to deal with less messages sent from the OpenvSwitch to the POX controller. It leads to the reduction of CPU consumption of the controller.

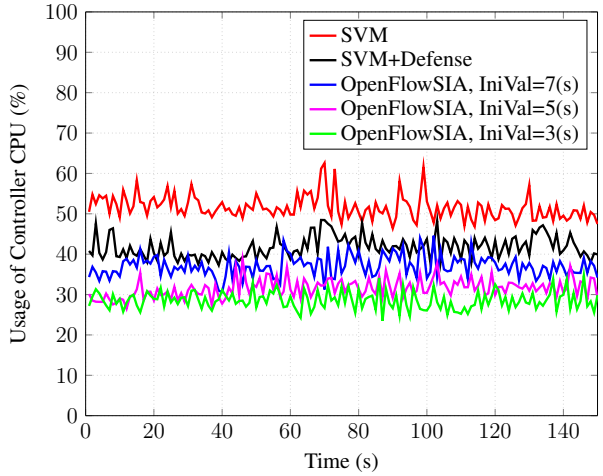

Fig. 6. The CPU usage of the SDN controller

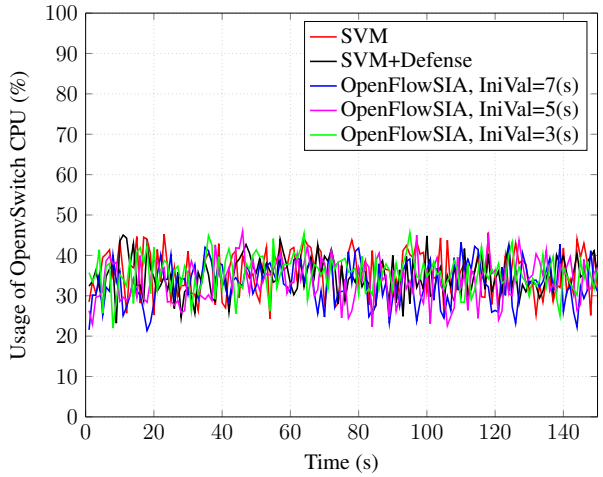

Fig. 7. The CPU usage of the OpenvSwitch

2) OpenvSwitch's CPU Usage Comparison: The next evaluation criterion is the usage of the OpenvSwitch's CPU. From Fig 7 illustration, we can see that there is no big differences between five cases, all lines are around 33.0% on average. This similarity can be clarified by following explanations. The OpenvSwitch in the SVM case having the highest flow number only has to spend its CPU resources on gathering flow information for statistic and sending to the Controller. Because of the high number of flow; thence, the CPU consumption is also considerable. The SVM+Defense has a Policy Forcement module on the controller; hence, the OpenvSwitch has to work following the controllers instruction to minimize attack effects. However, in this case, the flow number reduces dramatically in comparison with the previous test case; therefore, the CPU expenditure is approximate with the first scenario. With regard to the OpenFlowSIA tests, it is also because of a great reduction of the flow number and it is always lower than the SVM and the SVM+Defense schemes but the IA module is applied; hence, the OpenvSwitch has to process the IA rules from the controller. The balance between the flow number and the rule enforcement leads to the similarity of the CPU usage of the OpenvSwitch for all five cases.

VI. Conclusion and Future Work

In this paper, we propose an optimized protection scheme for Software-Defined Networks from flooding attacks based on the Support Vector Machine classifier and our proposed Idile-timeout Adjustment algorithm (IA). We demonstrated that our new mechanism totally differs from previous works and it outperforms original methods to save the SDN resources. As for future work, we plan to extend the OpenFlowSIA to support more types of attacks.

References

[1] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Commun. Surv. Tutor.*, vol. 16, no. 3, pp. 1617–1634, Aug 2014.

[2] The Open Networking Foundation, "OpenFlow Switch Specification Version 1.4.0," Accessed 2015 [Online]. Available: https://www.opennetworking.org

[3] Kaspersky Lab, "Statistics on botnet-assisted DDoS attacks report," Tech. Rep., 2015.

[4] John Pescatore, "DDoS Attacks Advancing and Enduring: A SANS Survey," Tech. Rep., 2014.

[5] S. Zargar, J. Joshi and D. Tipper, "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks," *IEEE Commun. Surv. Tutor.*, vol. 15, no. 4, pp. 2046–2069, Nov 2013.

[6] Kokila RT, s. Thamarai Selvi and Kannan Govindarajan, " DDoS Detection and Analysis in SDN-based Environment Using Support Vector Machine Classifier ," in *6th Inter. Conf. on Adv. Comput. (ICoAC)*, pp.205–210, Chennai, 2014.

[7] R. Braga, E. Mota and A. Passito, "Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow," in *35th IEEE Conf. Lo. Com. Net.*, pp.408–415, Denver, Colorado, 2010.

[8] P.Van Trung *et al*., "A Multi-criteria-based DDoS attack prevention solution using Software Defined Networking ," in *IEEE Inter. Conf. on Advan. Tech. for Commun.*, pp.308–313, 2015.

[9] S. Shin, V. Yegneswaran, P. Porras and G. Gu, "AVANT-GUARD: scalable and vigilant switch flow managament in software-defined networks," in *Proc. ACM SIGSAC Conf. Com. Commun. Sec.*, pp.413–424, 2013.

[10] Haopei Wang, Lei Xu, Guofei Gu, "FloodGuard: A DoS Attack Prevention Extension in Software-Defined Networks," in *In Proc. of the 45th Annual IEEE/IFIP Inter. Conf. on Dep. Sys. and Net.* Brazil, 2015.

[11] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge Univ. Press, UK, 2007.

[12] J. Shawe-Taylor and N. Cristianini, *Support Vector Machines and other kernel-based learning methods*. Cambridge Univer. Press, UK, 2000.

[13] CAIDA Datasets, "Anonymized Internet Traces 2015," Accessed 2015 [Online]. Available: https://data.caida.org/datasets/passive-2015/.

[14] CAIDA Datasets, "DDoS Attack 2007," Accessed 2015 [Online]. Available: https://data.caida.org/datasets/security/ddos-20070804/.

[15] BoNeSi, "The DDoS Botnet Simulator," Accessed 2015 [Online]. Available: https://github.com/markus-go/bonesi.