Journal of Network and Computer Applications 91 (2017) 14–25

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Distributed-SOM: A novel performance bottleneck handler for large-sized software-defined networks under flooding attacks

Trung V. Phan, Nguyen Khac Bao, Minho Park*

Department of ICMC Convergence Technology, Soongsil University, Seoul 156-743, Republic of Korea

ARTICLE INFO

Keywords:
Flooding attacks
Distributed denial-of-service
Self-organizing map
Software-defined networks

ABSTRACT

Software-Defined Networking (SDN) is a new programmable networking model that features the detachment of control and data planes. In this network, the network brain is an SDN controller that is used to centrally monitor and control the data plane based on the OpenFlow protocol and applications located in the application layer. In recent years, a vast number of issues relating to security have been seriously debated for this networking paradigm, especially the large-scale model. In particular, flooding attacks have been on the rise, providing great challenges for the SDN architecture to cope with. In this paper, we present a novel mechanism using the Self-Organizing Map (SOM) application to solve the performance bottleneck and overload problems for the upper layers in a large-sized SDN in case of flooding attacks. Our proposed approach integrates a Distributed Self-Organizing Map (DSOM) system to OpenFlow Switches instead of using a standalone SOM. By exploiting SDN advantages, such as flexibility and overhead reduction, we implement and test both a DSOM system and a single SOM system on multi-criteria to compare the performance of our introduced system. Our experimental results show that the DSOM solution can effectively detect abnormal traffic, solve bottleneck problems and increase the system reaction speed to attack traffic, while presenting a smaller overhead to the network system.

1. Introduction

Software-Defined Networking (SDN) (Masoudi and Ghaffari, 2016) has recently been recognized as providing an outstanding network architecture. In this network model, the separation of the control and data planes brings about enormous benefits for network operators in traffic management. OpenFlow protocol (OpenFlow, 2013) is a major component in the SDN paradigm, and it operates as a communicator between the control and data planes (an SDN controller and OpenFlow switches). The SDN controller uses the OpenFlow protocol to configure and update flow-tables inside OpenFlow switches according to network service instructions such as security and routing policies.

1.1. Problem statement

SDN promises a foreseeable solution for a wide range of services and applications in network configuration and management. The SDN technology has been deployed in large-sized networks (Jain et al., 2013) to manage internal traffic by running network services and applications. However, these networks are facing with following common problems.

In a large-sized network where there are a higher number of OpenFlow switches, the SDN controller naturally becomes a performance bottleneck because of the resource allocation for security services and applications. In particular, security applications always require advanced processing and analyses to operate their functionalities in the network. For example, Fig. 1 shows that two upper planes usually consist of three modules of *StatsCollector*, *SecurityApplication* and *PolicyEnforcement* to launch just a security service, and there is also no significant difference in case of using Middlebox servers for security purposes.

In addition, the communication channel between the data plane and control plane may become a bottleneck that increases latency and hinders the amount of traffic to the application plane (Wang et al., 2015) in case of the high traffic volume. Hence, many researchers have recently introduced multiple controllers for large-sized Software Defined Networks recently. A good illustration is Google's B4 network (Jain et al., 2013) which is a global deployment of the B4 SDN project. Similarly, Hassas Yeganeh and Ganjali (2012), Koponen et al. (2010) propose the multiple controller approach to diminish disadvantages of the single controller model. These works introduce either a two-level hierarchical controller or a distributed controller system. On the other hand, in a two-level hierarchy the upper controller is also a network bottleneck, and in a distributed controller scheme, while running many

* Corresponding author.
E-mail addresses: trungpv@ssu.ac.kr (T.V. Phan), khacbao@ssu.ac.kr (N.K. Bao), mhp@ssu.ac.kr (M. Park).

http://dx.doi.org/10.1016/j.jnca.2017.04.016
Received 2 May 2016; Received in revised form 22 March 2017; Accepted 27 April 2017
Available online 29 April 2017

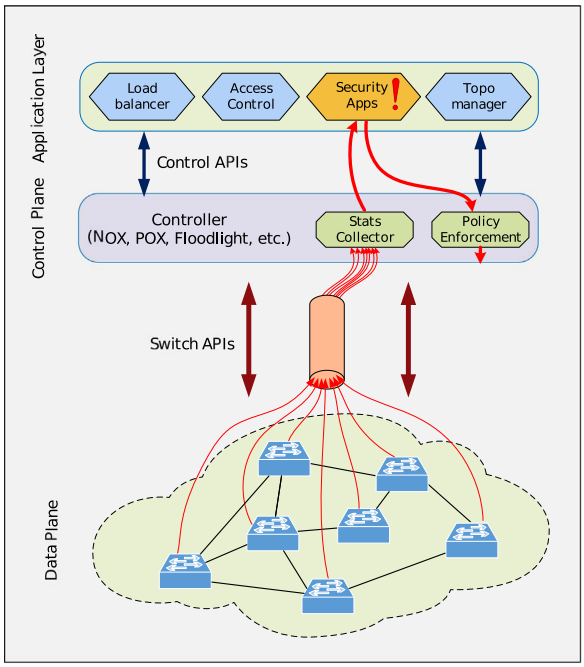

Fig. 1. The large-sized software-defined networks.

network services and applications at the same time, the overhead of synchronization is of growing concern.

These above problems are believed to be serious defects of large-scale SDN architecture in case of flooding attacks or Distributed Denial-of-Service (DDoS) attacks (Gu and Liu, 2012; Kaspersky, 2015; John Pescatore, 2014), which send an extreme high volume of network traffic to the victim system in order to exhaust resources and degrade quality of public services. Consequently, security applications located in upper layers of the SDN architecture have to process a great deal of traffic information, and the system may collapse because of resource exhaustion and a resultant performance bottleneck.

Accordingly, in order to reduce the tremendous performance pressure for running security applications on the upper planes in case of the high traffic volume in the large-sized SDN model, the data plane needs to support more advanced functionality.

1.2. Our proposal: distributed-SOM

In order to resolve the above serious issues and enhance the robustness of the large-scale SDN model, we propose a Distributed-Self Organizing Map (Distributed-SOM or DSOM) system as a new mechanism for tackling the performance bottleneck problem under flooding attacks in the large-sized Software Defined Networks. The introduced method uses multiple SOMs integrated into OpenFlow switches instead of using a single SOM in upper planes as a security service. Each DSOM operates as a single SOM, but it independently processes incoming traffic from outside networks to the switch it resides in. The DSOM mechanism now solves aggregation concerns, while still ensuring the performance of detection abnormal traffic; as a result, upper planes are able to avoid the resource exhaustion because of the performance bottleneck in the case of flooding attacks.

1.3. Proposal feasibility

We introduce the DSOM mechanism as a new security solution to address the problems discussed above, and the proposed system requires some extensions to the OpenFlow data plane. Our proposal is believed as a practical and feasible solution for implementation because of following reasons. *Firstly*, the OpenFlow standard has many data plane limitations discussed in Sonchack et al. (2016), Shin et al. (2013),Park et al. (2016), which motivate these work to propose extensions to handle their facing problems. With their introduced extensions, the data plane can do more advanced functionality in comparison with original OpenFlow switches. For example, the OpenFlow extension framework is proposed in (Sonchack et al., 2016) to enable security applications to efficiently run on switches (the author tested by using a Pica8 3290 switch with a Broadcom Firebolt-3 forwarding engine), while AVANT-GUARD (Shin et al., 2013) also introduces an extension called *connectionmigration* to reduce the amount of data-to-control-plane interactions. And, UNISAFE (Park et al., 2016), which is a new software switch architecture, is introduced to enrich security functions/features in software-defined environments. *Secondly*, vendors have added software-based extensions to the data plane, which can be referred as slow path operations in Medhi (2007), with purpose of providing additional functionalities along with the hardware-based forwarding path. For instance, BigSwitch Network moved the ARP handling operation from its FloodLight Controller (Floodlight is an Open SDN Controller, 2017) to its Switch Light that runs on a whitebox SDN switch (FERRO, 2014). Meanwhile, the embedded event manager (EEM) (Cisco,), which can trigger minor reconfigurations based on a predetermined selection events, is integrated into Cisco switches. *Moreover*, most modern switches support a small-scale operating system (Open network linux, 2017; Pica os white box switch os, 2017) (typically Linux-based) with computational capabilities; hence, this allows developers easily to add a functionality to the switch to achieve a given task. From mentioned reasons, we totally believed that our proposal is a practical and feasible solution for reducing security vulnerabilities in Software-Defined Networking.

1.4. Contributions

In summary, our contributions are as follows:

- We propose a new SDN security mechanism that resolves the performance bottleneck problem and resource exhaustion at upper layers in large-sized SDN architecture. In this proposed scheme, security modules are distributed to OpenFlow switches instead of locating at the control plane or the application layer, and these modules are controlled by a distributed system controller which runs as a security application in the application layer in the SDN architecture. Besides, we implement extension modules in OpenvSwitches, which allows us to connect to default agents of an OpenvSwitch with the purpose of statistics, rules modification and communication with the distributed system controller.
- We apply the Self-Organizing Map applications to the distributed system as a main classifier at each OpenvSwitch. We show that our presented mechanism can effectively detect abnormal traffic and solve bottleneck problems, while presenting a smaller overhead to the network system.

In this article, Section 2 gives readers an overview of the SDN architecture and its benefits and explains why we chose the Self Organizing Map by investigating its applications in network traffic classification. In Section 3, we describe the proposed DSOM system and its components in great detail. Section 4 details the experiments. Section 5 presents the results and performance evaluation of the DSOM system. Section 6 discusses previous researches related to our work. Finally, Section 7 concludes the paper.

2. Background

2.1. Software-defined networking and openflow protocol overview

Software-Defined Networking (SDN) offers a promising network framework for the future. Fig. 2 demonstrates that in this network, the

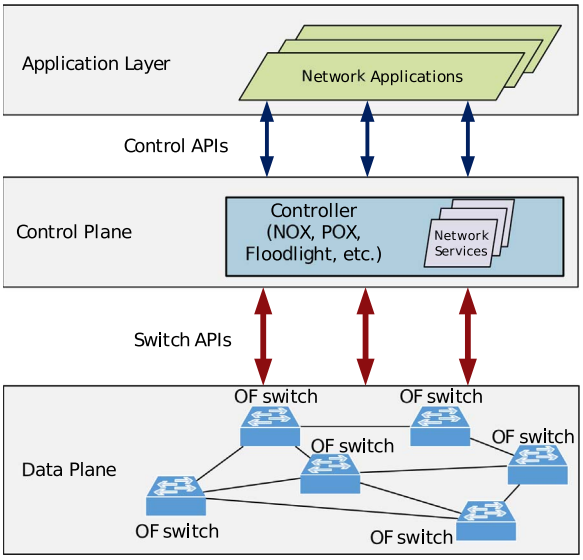

Fig. 2. The SDN architecture.

control plane, which is located in an SDN controller, is uncoupled from the data plane. Applications running on the application layer can provide complex network services and functions. For instance, a routing module is accountable for computing paths between networks or a dynamic host configuration protocol provides hosts with IP and DNS information. We are also able to create new network services or functions by writing an application in Python, Java, etc. The data plane deals with the hardware level and focuses on packet processing based on configurations from the controller. Thus, this separation brings numerous advantages to network managements, especially security matters.

The communication between the control plane (the SDN controller) and the data plane (OpenFlow switches) is defined by OpenFlow protocol (OpenFlow, 2013). Before exchanging messages, an OpenFlow switch has to establish a secure connection to the SDN controller for authentication. This protocol gives the controller access to flow-tables in the OpenFlow switch for control, configuration and statistics using secure connections.

Fig. 3 depicts major components of an OpenFlow switch including Flow-Tables and OpenFlow Channel, which are explained in depth in OpenFlow (2013). Fig. 4 shows a flow-entry structure in a Flow-Table that consists of numerous items of statistical information such as: duration, number of packets, number of bytes, protocol, IP address, service port and etc. To collect this information for all flow-entries, the controller just sends a request message to an OpenFlow switch and receives response messages. Hence, working on network management becomes easier in comparison with conventional networks.

SDN has attracted much attention from both industry and research in recent years because of the innovations and benefits the it offers. We can summarize some major benefits of SDN as follows:

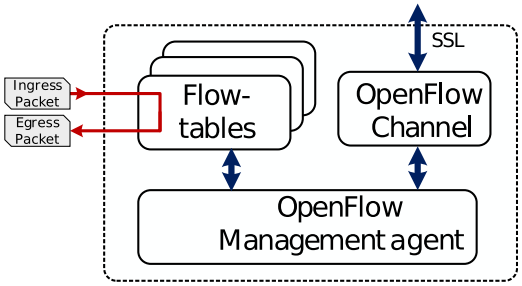

Fig. 3. The main components of an OpenFlow switch.

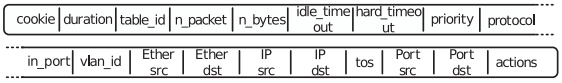

Fig. 4. A flow-entry structure.

- SDN switches are becoming simpler than traditional ones because they are implemented with fewer protocols.
- SDN architecture allows changes of data or control plane components as long as they support a standard interface (e.g OpenFlow protocol).
- Network services and applications are more flexible to deploy or implement on the upper layers, when compared with conventional methods.
- SDN can adapt to different vendors developing their own extensions and changes, which makes SDN more attractive to developers and researchers.

2.2. Self organizing map algorithm

The Self Organizing Map (SOM) (Kohonen, 2001) is one of the unsupervised learning solutions in Artificial Neural Networks. This algorithm transmutes a higher-dimensional input space into a lower-dimensional representation called a SOM map. SOM classifies or detects a new input vector based on two major modes: training and mapping. The training process builds, reorganizes the map using input samples, while the mapping process automatically classifies new input vectors by finding its winning neuron or node in the map.

The detail of the algorithm is expounded in Kohonen (1990). Hence, we briefly describe the training process of the SOM algorithm, and Fig. 5 depicts the SOM map. Before going into the specific steps, we attend to some predetermined parameters:

- n is number of iterations or number of training samples $[\vec{x_1}, \ldots, \vec{x_n}]$.
- The map has S neurons and each neuron is represented by a vector, W, which consists of m weight values.
- R is radius of the rectangular map which is defined as

$$R = \frac{\max(MapWidth,\ MapHeight)}{2}. \tag{1}$$

- λ is a time constant and calculated as

$$\lambda = \frac{n}{\lg(R)}. \tag{2}$$

- $\sigma(t)$ is the neighborhood radius of a winning neuron (the Best Matching Unit) and diminishes over time as shown in Fig. 6. Assuming that the neighborhood remains centered on the same neuron, in practice the Best Matching Unit will move around based on the computation in Step 2 below. At the iteration tth of the

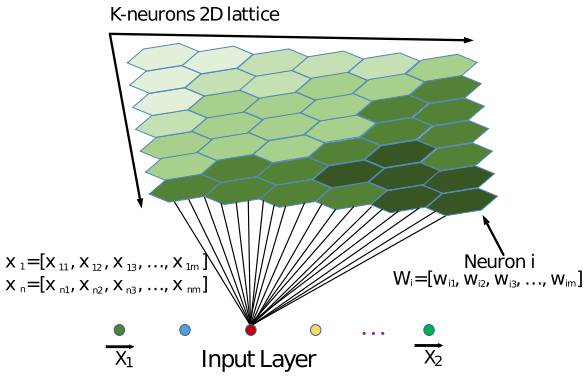

Fig. 5. The self organizing map.

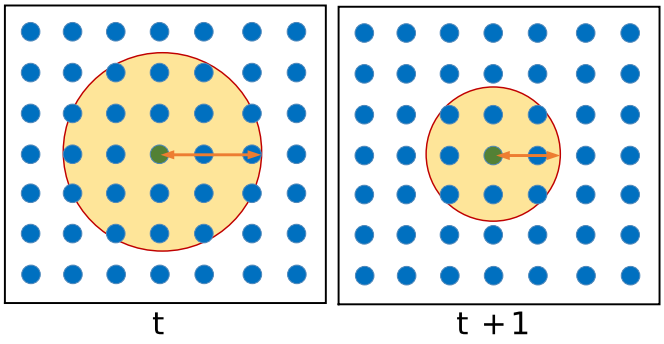

Fig. 6. The radius degradation in the map.

algorithm, $\sigma(t)$ is calculated as

$$\sigma(t) = R \times \exp\left(-\frac{t^2}{\lambda}\right) \quad t = 1, \ldots, n. \tag{3}$$

The four main steps of the algorithm are explained as follows:

1. *Initialization*: Neuron's m-dimensional weights, $W_i = [W_{i1}, W_{i2}, W_{i3}, \ldots, W_{im}]$, where $1 \le i \le S$, are initialized with random or fixed values.
2. *BestMatchingUnit (BMU) Selection*: An input vector, $x_k = [x_{k1}, x_{k2}, x_{k3}, \ldots, x_{km}]$, is fed to the map by computing its Euclidean distance to all neurons i, written as

$$D_{ist} = \sqrt{\sum_{j=0}^{m} (x_{kj} - W_{ij})^2}. \tag{4}$$

The neuron having the smallest D_{ist} is selected to be the Best Matching Unit for the input vector, x_k.
3. *UpdateBMU'sNeighborWeight*: The BMU's neighbors are calculated as in Eq. (3). Afterwards, these neighbor's weight for the next iteration are adjusted to make them closer to the input vector according to the following equation:

$$W(t+1) = W(t) + L(t)*\Theta(t)*(x_k(t) - W(t)) \tag{5}$$

where $L(t)$ is the learning rate, which also shrinks over time and computed as

$$L(t) = L_0 * \exp\left(-\frac{t}{\lambda}\right), \tag{6}$$

and where $\Theta(t)$ is the amount of influence that a neighbor neuron's distance from the BMU has on its learning and defined as

$$\Theta(t) = \exp\left(-\frac{D_{ist}^2}{2\sigma^2(t)}\right). \tag{7}$$

4. *Loop*: Repeat steps 2 and 3 until there is no more input vector fed to the map.

2.3. Self organizing map applications

As shown in 2.2, the SOM algorithm uses a ready-made input set for training and mapping procedures, and for deciding on a new input bases on trained neurons in the SOM map. Because of its classification capability, SOM has been applied to a wide range of research fields for pattern recognition. In particular, many previous works utilized the usage of SOM in network traffic classification and abnormal traffic detection. To illustrate, Jankowski and Amanowicz (2015) and Braga et al. (2010) take the SOM as a brain to detect unauthorized activities and DDoS detection in Software-Defined Networks. Meanwhile, Langin et al. (2009) uses the SOM to classify botnet traffic and other malignant network activities. Mitrokotsa and Douligeris (2005) proposes a modified version of the SOM called Emergent SOM to detect DoS and a method using SOM is presented by Zanero (2005) to analyze TCP traffic patterns. Similarly, researches in Ramadas et al. (2003), Gunes Kayacik and Zincir-Heywood (2007), Jiang et al. (2009), Alsulaiman et al. (2009), Wang et al. (2009), Kiziloren and Germen (2007), Ting et al. (2010), Nascimento et al. (2013) also introduce different approaches to taking advantage of SOM in network traffic classification with the purpose of securing network systems. All the mentioned works prove that the SOM is one of the most effective classifiers, which is always applied to every Intrusion Detection System. Consequently, in this research, we decided to choose the Self-Organizing Map algorithm as the main classifier to use for classification processes.

2.4. Flooding attacks under the SDN perspective

In conventional network frameworks, flooding attacks are generally divided into two main types: bandwidth depletion attacks and resource depletion attacks. In the former type, attackers tend to flood a victim with unwelcome traffic with the purpose of exhausting victim network's bandwidth. This leads to legal traffic not being able to access the victim system as a normal case. ICMP flooding, UDP flooding, or Smurf and Fraggle attacks are good examples to illustrate this attack type. Turning to the resource depletion attacks, attackers want to flood malformed IP packets or misused network protocol packets to the victim. Accordingly, whenever the volume of opening connection reaches the system threshold, the victim suffers from resource exhaustion, and may stop working. For example, TCP SYN flooding exploits the three-handshake protocol between sender and receiver before a TCP connection is established.

From the flow-based SDN perspective, we can also divide flooding attacks into two main types as follows:

2.4.1. Type I

The first type is based on the volume of packets transmitted in a flow. A distinctive characteristic of this attack type is that a client connects to the victim system by producing a few flows with an extremely high packet number in each flow. Examples are ICMP flooding attacks, and Smurf and Fraggle attacks.

2.4.2. Type II

The second type relies on the number of flow coming to the victim network system. A client produces a huge number of flows to the victim network in a short period of time. A good illustration is TCP SYN flooding attack, in which a client spoofing its IP address generates hundreds of requests to a victim server. In this type of flooding attack, a vast number of clients send their requests at the same time, which causes not only the victim server, but also SDN components such as the SDN controller and OpenFlow switches to break down.

In summary, common flooding attacks happening in the Software-Defined Networking environment can be classified as in Table 1.

Table 1
Flooding attacks under the SDN perspective.

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

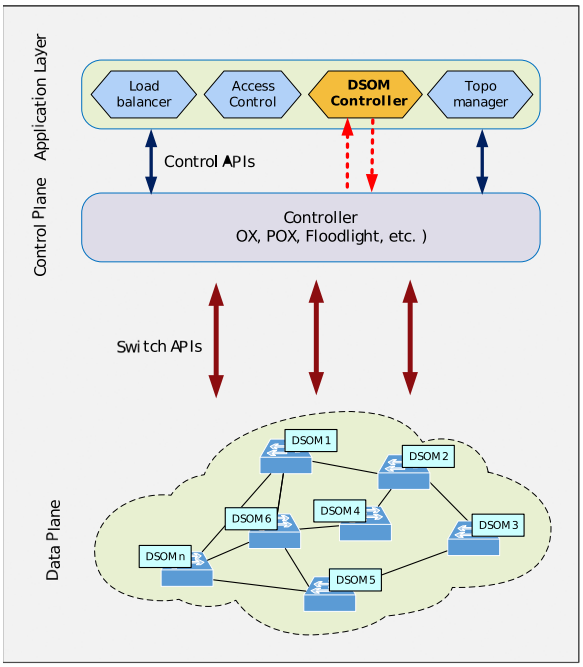

Fig. 7. The proposed Distributed-SOM system.

3. The proposed distributed-SOM system

3.1. The DSOM system overview

We introduce a Distributed Self Organizing Map system in large-scale Software-Defined Networks to handle the performance bottleneck issue in the case that the network system is under flooding attacks. Fig. 7 shows the proposed system overview, which consists of modules located in the control plane and application layer, and DSOM⊕OpenFlow switches. All switches are under the control of a SDN controller via a secure connection. In this work, a DSOM module is integrated into an OpenFlow switch by adding extension modules, and we use the OpenvSwitch (OpenvSwitch, 2014) for customization. In the application layer, we place an application named *DSOMController* that controls the DSOM operation. We describe the operation of the DSOM system and DSOM extension modules in greater detail in the following sections.

3.2. The DSOM system workflow

In summary, the introduced DSOM system is designed with four main processes: *Initialization*, *Merging*, *Updating* and *Classifying*. We first look at the workflow of the DSOM system containing these procedures as follows:

1. *Initialization*: The *DSOM Controller* located in the application layer sends ready-made datasets to switches that integrate with DSOM extension packages. At each switch, DSOM Map is trained by using a dataset that it receives from the Controller.
2. *Merging DSOM*: This stage is responsible for consolidating DSOMs in a preset period of time. The *DSOM Controller* collects DSOM maps from OpenFlow switches to make a Merged SOM map, expressed as

$$MergedMap = \sum_{j=1}^{j=n} \frac{\chi_j}{\sum_{i=1}^{i=n} \chi_i} \times MapDSOM_j. \tag{8}$$

where χ_j is total number of trained input samples of DSOM j_{th} and $MapDSOM_j$ is the map of DSOM j_{th}.

For clarity, let us explain the merging procedure in greater detail by assuming that:

$$C_j = \sum_{j=1}^{j=n} \frac{\chi_j}{\sum_{i=1}^{i=n} \chi_i} \quad j = 1,\ldots,n, \tag{9}$$

In order to make a single map from n DSOM maps, neuron weight components in the Merged SOM map are calculated as

$$MergedW_{ik} = \sum_{j=1}^{j=n} C_j \times W_{ik} \quad k = 1,\ldots,m. \tag{10}$$

where W_{ik} is weight component value k_{th} of neuron i_{th} in DSOM map j_{th} and m is the weight dimension.

3. *Updating*: The *DSOMController* sends the Merged SOM map to all OpenFlow switches. The $DSOM_j$ map is replaced by the Merged SOM map to continue the classification process at the switch.
4. *Classifying*: At this stage, the $DSOM_j$ conducts computations for inputs based on their neurons and gives out results. These results are then sent to the *DSOMController* for further decisions.

Note that the *DSOMController* calls and carries out the *MergingDSOM* and *Updating* processes as a loop in a period of time determined by an administration. We investigate DSOM modules in depth at both control and switch levels in Sections 3.3 and 3.4.

3.3. The control DSOM level

At the control level, this model is a combination of the control plane and the application layer. Fig. 8 shows that the *DSOMController* runs as an application with the help of a module named OpenFlow Controller placed in the control plane.

The *DSOMController* application includes five major modules: DSOM Controller Agent, Training Database, DSOM Interface, MergingSOM, and Policy Checking. The DSOM Interface is a module that opens a channel between DSOM modules and the OpenFlow Controller. Note that each DSOM module has its own channel communication; in this work, we use different socket connections for different DSOM functions. The DSOM Controller Agent controls the operation of the Training Database and the MergingSOM to launch the above three processes (*Initialization*, *MergingSOM* and *Updating*). Moreover, it also manage the Policy Checking for verifying rules.

With respect to the *Initialization* procedure, the DSOM Controller Agent exchanges messages with the OpenFlow Controller to get information of switches that are registered to use the *DSOMController* application, and this procedure is declared by an administration in the

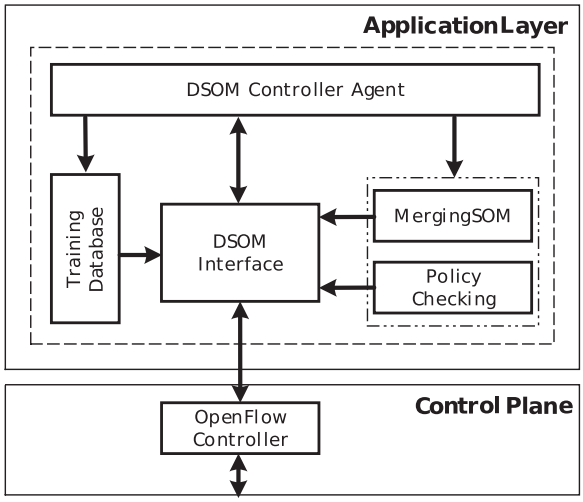

Fig. 8. The control DSOM level.

OpenFlow Controller. The DSOM Controller Agent maintains a DSOM-table that stores switch information and the corresponding DSOM map data. After getting switch information, the DSOM Controller Agent sends commands to the Training Database to distribute initial training datasets to registered switches via the DSOM Interface and the OpenFlow Controller. DSOM Maps located in switches are then trained with the received datasets, and the *Initialization* process is completed. In case a new switch is registered to join the DSOM system, the DSOM Controller Agent will make a separate initialization process for the new switch.

In the MergingSOM process, the DSOM Controller Agent sends messages to switches to gather DSOM maps. Whenever the DSOM map information arrives at the DSOM Interface, it is forwarded to the DSOM Controller Agent. Relying of the DSOM-table, the DSOM Controller Agent checks connected switches using the switch *dpid* number and the negotiated OpenFlow *version*, and puts this information into a stack for merging preparation. After receiving all DSOM maps of switches, the DSOM Controller Agent sends the stack containing DSOM map data to the MergingSOM module. At this module, the merging process runs as illustrated in Section 3.2. Then, the MergingSOM module sends a merged SOM map back to all switches via the DSOM Interface and the OpenFlow Controller.

Turning to the Policy Checking function, this module is accountable for verifying rules installed by the Fast Policy Enforcement at switches, and this module can overwrite installed rules if there is something that is needed to change such as *ruleaction* or *ruledestination*. Whenever the DSOM Interface receives any messages related to rule application, they are transferred to the DSOM Controller Agent, and the last destination is the Policy Checking module. We investigate how this module works in the next sections.

3.4. The switch DSOM level

At the switch level, we add some additional modules called DSOM extension modules to the OpenFlow switch: Flow Collector, Feature Extractor, Training Database, DSOM Map, Fast Policy Enforcement and DSOM Switch Agent. Fig. 9 shows how these modules are connected, and they work with default modules of the OpenFlow switch. The DSOM Switch Agent acts as a brain of the DSOM system at a local switch level, and controls the training and updating DSOM map procedures, sending classification results to both the Fast Policy Enforcement and the *DSOMController*. In this research, we developed these extension modules in the environment of open source OpenvSwitches (OpenvSwitch, 2014).

At the *Initialization* step, a training dataset sent from the *DSOMController* arrives at the DSOM Switch Agent via the OpenFlow Channel of the OpenFlow switch. It is then forwarded to the Training Database, and becomes training input for the DSOM map. When the training process is completed, the DSOM Switch Agent sends start commands to the DSOM Map, the Feature Extractor and the Flow Collector to launch these modules. As the scheme works, the Flow Collector periodically sends statistical messages to the switch data plane to get individual flow information during the Feature Extractor phase, which focuses on feature extraction for every client passing through the switch. Besides, the Training Database is updated by outputs of the Feature Extractor, and the training DSOM map procedure is called again in a time period set by the DSOM Switch Agent. This training keeps the DSOM map adapting to the coming traffic, and it improves the classification performance at each local DSOM switch.

Let us look in greater detail at the Feature Extractor to understand how many features are used in an input of the DSOM map. The features can be diverse for each type of network traffic; however, as described in Section 2.4 we consider common flooding attacks in the SDN perspective, and classify them into two main types (Type I and Type II). Therefore, this work focuses on two types of flooding attacks, and our

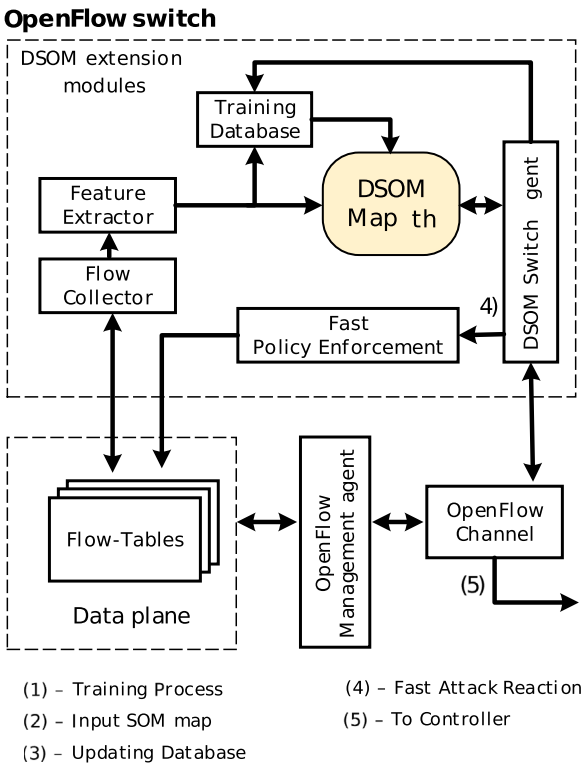

Fig. 9. The switch DSOM level.

approach is observing every client in a predetermined period. Client information then is fed to the DSOM map as a tuple of the six following features (6-tuples) to make a decision:

a) Number of flows: This is a fundamental attribute for flooding attacks. In particular, the Type I flooding attack has only one flow, while the Type II generates a large number of flows for a client.
b) Number of packets per flow: The second feature that can be chosen is the amount of packets for one flow in the observation time. The Type I case carries a vast number of packets per flow, but in Type II, this feature becomes the smallest with a few packets.
c) Number of bytes per flow: The appearance of this factor looks similar to the previous one, due to the fact that a greater number of incoming packets implies a larger number of bytes per flow for Type I, while in contrast the Type II attack carries fairly small bytes.
d) Duration: This presents how long a client connects to a server in the destination network. For different type of attacks, the duration of a client is also dissimilar. For example, in Type I attacks, attackers tend to send packets to a server for a long time. However, in the Type II case, attackers spoof their source IP addresses and service ports; therefore, the connection period is not usually long.
e) Growth of client ports: Under flooding attacks, for the Type I attack, there is no change in the number of client ports during the observation time. Whereas in the case of the Type II attack, a client produces numerous service ports.
f) Protocol: This attribute plays a key role in classifying the type of flooding attacks. Diverse protocols can be expressed as different integers, such as ICMP 0 and TCP 1.

The DSOM map performs the same as a single SOM. It determines which clients are normal or anomalous users. In order to make a final decision, we introduce a $K-means$ clustering algorithm (Tapas Kanungo et al., 2002) that partitions n patterns into K clusters. We use K=20 clusters to carry out our proposed approach experiments in Section 4. After attacking clients are clearly defined, the DSOM map sends client information to the DSOM Switch Agent, then this agent

transfers these data to both the Fast Policy Enforcement and the *DSOMcontroller* via OpenFlow Channel. The purpose of placing the Fast Policy Enforcement module at the switch is for quicker reaction to attack traffic by installing direct rules in flow-tables, while the Policy Checking module located in the *DSOMcontroller* is accountable for verifying rules and it can overwrite installed rules if needed.

The Fast Policy Enforcement module is programmed to process attacking client's information and directly insert rules into flow-tables. We have modified source code in path *ovs/lib/ofp − actions*, and that modification allows the Fast Policy Enforcement to install flow rules into OVS flow-tables through *ovs − ofctl* or *ovs − dpctl*. From our above discussion in Section 2.4, these rules can be categorized as follows:

- With regard to clients belonging to the Type I attack, the Fast Policy Enforcement generates rules with a *drop* action and insert to flow-tables in order to discard all packets coming to the switch. Therefore, there are no malicious packets passing through the switch, and the inside network is protected from attack packets.
- To mitigate clients of the Type II attack, the Fast Policy Enforcement first sends appropriate *delete* commands to flow-tables to remove all flows of attacking clients and then installs a new rule with a *drop* action with the purpose of discarding any requests from abnormal users. These actions protect not only the inside network from attacking requests, but also SDN devices from resource exhaustion, due to the large number of packets sent to the SDN controller to install new flow-entries in the switch.

4. Experiments

In this section, we first take a glance at datasets (CAIDA Dataset, 2015, 2007; NSL-KDD Data Set for Network-based Intrusion Detection Systems, 2009; LANDER:DARPA, 2009) and a flooding attack simulator (BoNeSi,). Next, we explain how the DSOM system is implemented in Software-Defined Networks. Finally, we perform several tests to evaluate our proposed system in comparison with traditional way, and we set the observation time as *one* minute before doing our experiments.

4.1. Datasets and attack simulator

4.1.1. Datasets for training and testing

We analyze and build the set of SOM training and test samples from three datasets: CAIDA (CAIDA Dataset, 2015, 2007), NSL-KDD (NSL-KDD Data Set for Network-based Intrusion Detection Systems, 2009) and DARPA 2009 Intrusion Detection (LANDER:DARPA, 2009) datasets. The short description of these datasets is summarized as follows.

- The CAIDA Dataset (Normal traffic on 21st May 2015 and DDoS attack traffic on 04th August 2007) collects various real network traffic types at geographically and topologically different locations. It has become the most credible network dataset and is available to research society with a mixture of several different types of traffic, such as Web, FTP, and Ping. From the description of Table 2, both TCP and ICMP protocols usually constitute the highest proportion of network traffic, respectively.
- As a new dataset, which is improved from the complete KDD dataset (Cup, 1999), the NSL-KDD has several advantages that outweighs the previous version. Firstly, no redundant records in the training set; hence, the classifier will not produce any biased results. In addition, there are no duplicate record in the test set, which produce better reduction rates. Moreover, the number of selected records from each difficult level group is inversely proportional to the percentage of records in the original KDD dataset. The training dataset includes 21 different attacks, while there are 37 attacks type in the test dataset. The attack types are categorized into four major groups (DoS, Probe, U2R and R2L). In this work, we are investigating the DoS attacks; therefore, we take only DoS samples into account for training and testing our proposed system as shown in Table 3.
- The 2009 DARPA Intrusion Detection dataset is created to aid in the evaluation of networks intrusion detectors. The dataset was made from between the 3rd and the 12th of November of year 2009 between a/16 local subnet and the Internet traffic, and it consists of synthetic HTTP, SMTP, and DNS background traffic. The major traffic contains background traffic and a SYN flood DDoS attack on one target from about 100 different IPs in around 6 min. Interested readers can refer to LANDER:DARPA (2009) for more information.

Table 2
CAIDA datasets in bytes.

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

Table 3
DoS attacks in NSL-KDD dataset.

DoS Attack Type	Training Patterns	Testing Patterns	Pattern Features
back land neptune pod smurf teardrop	45927	7458	41

We use a software named Omnipeek (Omnipeek Network Analysis, 2017) to extract and randomly choose samples (including normal and attack patterns) from pcap files in three datasets, and each sample includes six features as discussed in Section 3.4. The ready-made datasets will be used for the system testing in Section 4.3.

4.1.2. Attack simulator

We execute our test by running a flooding tool named BoNeSi (). This flooding attack tool not only simulates Botnet traffic, but also generates TCP, ICMP or UDP flooding attacks to a target network or a specified IP address from defined botnets. With regard to the implementation of our proposed approach, the BoNeSi tool provides us with attack traffic to evaluate the performance in the SDN environment.

4.2. System setup in SDN

The implemented topology is connected as in Fig. 10, in which we use a Controller, four OpenvSwitches integrated with DSOM modules, one Web server and three hosts as traffic generators. Three edge OpenvSwitches connect to a hardware Openflow-enabled switch (HP E3800 (HP,)) to gain access to the Web server. All these switches are controlled by the Controller via secure connections. Tables 4, 5 enumerate detailed parameters of the DSOM maps and devices.

4.3. System testing

4.3.1. Performance bottleneck test

In order to evaluate our proposed system in terms of performance bottleneck solving in comparison with the single SOM. We set up our system as illustrated in Fig. 10 and conducted two test cases: a single SOM and the DSOM system. In the first test scenario, we implement a SOM module in the controller containing four functions: Flow

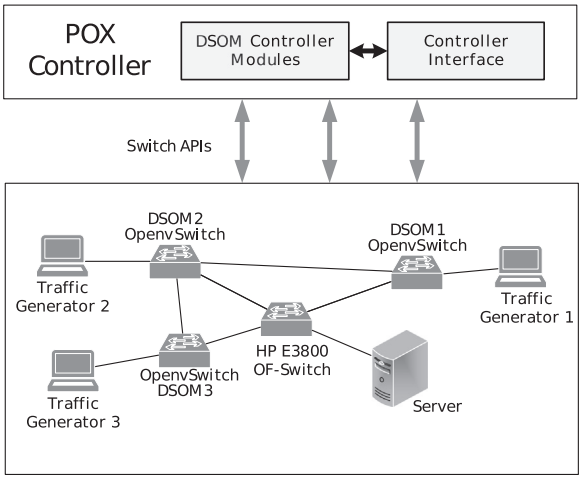

Fig. 10. The DSOM implementation topology.

Table 4
SOM map parameters.

Parameter	Value
Radius	(Width+Height)/2
Learning Rate	0.1
Number of Neurons	400, 900, 1600
Input Dimension	6
Output Dimension	2

Table 5
Device information.

Name	Type or Version
SDN Controller	POX
OpenFlow switch	OpenvSwitch 2.3.0 HP E3800
OpenFlow	1.0
Traffic Generator	Unbuntu 12.04

Collector, Feature Extractor, SOM map, Policy Enforcement. Meanwhile, in the proposed system, we only run the DSOM Controller modules as an application in the SDN controller and other modules are implemented in switches. In two cases, we used a same dataset consisting of 12000 samples in total for training SOM maps, in which 4000 patterns are randomly extracted from each dataset as mentioned in Section 4.1.1. Next, we generated two traffic levels (50 Mbps and 100 Mbps) using the BoNeSi tool to attack the Web server and measured the CPU utilization of the SDN controller in both two cases.

4.3.2. Attack reaction test

One of the most important criteria in security evaluation for every network system is the time of system reaction in case the system is under attacks. Hence, we carried out tests to measure the time, t_{re}, of the system reaction to flooding attacks in several cases. We determine the time t_{re} by calculating the total time of an abnormal client from being detected by the SOM map until policy rules are installed in flow-tables. We conducted the test with three scenarios as follows.

- SingleSOM: A single SOM with same four modules as described in Section 4.3.1. The controller collects flow information from OpenvSwitches and then detects attacking clients. If an anomaly client is detected, the Policy Enforcement will immediately apply rules to OpenvSwitches.
- ModifiedDSOM: The DSOM system *without* the Fast Policy Enforcement at OpenVswitches, and we replace the Policy Checking by a Policy Enforcement. This replacement makes the rule applying function as same as in the single SOM case. It means whenever the SOM map at a switch detects an abnormal client, the DSOM Switch Agent will only send client's information to the DSOM Controller to ask for policy enforcement.
- ProposedDSOM: In this case, we utilizes the Fast Policy Enforcement module inside the switch to apply rules as fast as possible when a client is detected as a DoS attacker. Then, the anomaly client's information will be forwarded to the Policy Checking in the DSOM Controller to verify installed rules.

Note that we used a same dataset consisting of 12000 samples in total for training SOM maps in three scenarios, in which each dataset, mentioned in Section 4.1.1, equally contributes 4000 patterns. In order to conduct the test, we generated three traffic levels (50 Mbps, 100 Mbps and 200 Mbps) running the BoNeSi tool to attack the Web server and computed the time t_{re} for each scenario.

4.3.3. SOM performance test

With regard to the detection performance of the SOM map in our introduced solution, we carried out the following setup cases to evaluate the DSOM system and prove that our proposal has the same performance with the traditional SOM mechanism:

- SingleSOM: A single SOM with the same configuration as the scenario 1 in Section 4.3.2.
- DSOM 1:1:1: This scenario is conducted in our introduced DSOM system with the ratio of training datasets in the *Initialization* process is 1:1:1.
- DSOM 1:2:3: This case is the same as Case 2, except that the ratio of training datasets in the *Initialization* process is 1:2:3.

In this experiment, we perform separately following tests for each case:

- The SOM map initialization and training processes are conducted by individual datasets(CAIDA, NSL-KDD and DARPA) as shown in Table 6. Then, the testing procedures are executed by the 5-fold cross-validation from the training dataset and tested with traffic from the BoNeSi DDoS Simulator.
- The SOM map is initialized and trained by a mixed dataset resulting

Table 6
Training and testing samples for SOM detection performance test.

Cases	Initialization	Training	Testing				
			CAIDA Dataset	NSL-KDD Dataset	DARPA Dataset	Mixed Dataset	BoNeSi Tool
Single SOM	4000	6000	30000	30000	30000	30000	6000
DSOM 1:1:1	4000:4000:4000	2000:2000:2000	30000	30000	30000	30000	6000
DSOM 1:2:3	4000:4000:4000	1000:2000:3000	30000	30000	30000	30000	6000

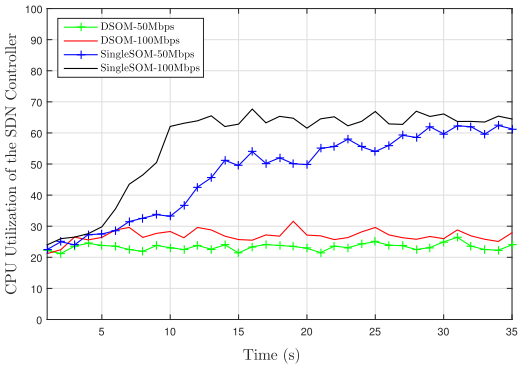

Fig. 11. The SDN Controller's CPU utilization.

from randomly merging three equal datasets. And, the testing processes are also performed by the 5-fold cross-validation and the BoNeSi DDoS Simulator.

We conducted the testing process many times for different values of the neuron number in the SOM map: 400, 900 and 1600.

5. Performance evaluation

In this section, we present the experiment results conducted in Section 4, and examine the performance of our proposed system.

5.1. Handling performance bottleneck

We firstly examine how our proposed system can solve the performance bottleneck issue as discussed in Section 1.1. From our implementation in Section 4.3.1, we achieved results as illustrated in Fig. 11. It is clear that there are significant differences among test cases, the single SOM mechanism always consumes much higher the CPU resource of the SDN Controller while the DSOM system usually keeps the controller system at a stable stage of CPU consumption. In case the generated traffic is 50 Mbps, the value of CPU utilization reaches 40% and 60% right after the attack launches just 12 and 33 s; meanwhile, the DSOM solution only consumes around 23% of the CPU resource in all testing time. The same difference also can be witnessed in the case of 100 Mbps, the single SOM scheme hit 60% of the SDN controller CPU consumption after just 10 s and it keeps this trend until at the end of the test. In contrast, the DSOM solution just shows a slight difference in terms of the CPU consumption from the previous measurement.

From the above results, the DSOM system can be seen as a great solution that totally outweighs the sing SOM scheme, and this can be explained by some following reasons:

- In the single SOM case, the controller has to frequently send messages to switches to get flow information, and then forward them to next modules for further processes. Moreover, processing flow information from all edge switches at the same time is a heavy workload for modules.
- The DSOM system delegates the traffic processing task to edge switches to reduce its workload, and it only process a little amount of information in case of policy checking. In addition, each edge switch only processes the traffic entering its ports from outside networks; thus, the pressure on the Switch DSOM Agent is not much considerable in comparison with the single SOM case.

5.2. Attack reaction performance

The second evaluation criterion of our introduced solution is the system reaction time to attacks in comparison with other approaches.

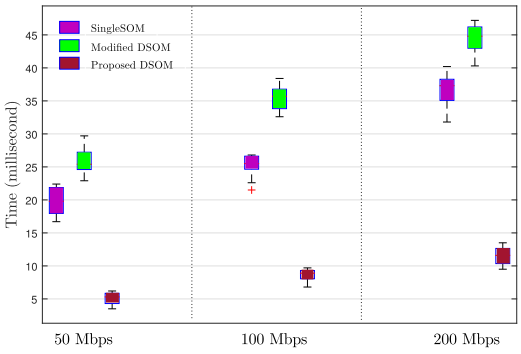

Fig. 12. The reaction time of the system in various attack levels.

Fig. 12 shows the distribution of reaction time to three flooding attack volumes (50 Mbps, 100 Mbps and 200 Mbps) of three scenarios as discussed in Section 4.3.2. Overall, our proposed method is always the best case in all test scenarios, which followed by the single SOM and the DSOM *without* the Fast Policy Enforcement at switches, respectively. To be more detailed, when the system is under 50 Mbps of the attack traffic, the DSOM solution only needs 5 ms to install a flow rule in the switch flow-table, while the single SOM spends almost 20 ms and the modified DSOM system is up to around 26 ms to finish this work. Similarity, in both cases 100 Mbps and 200 Mbps, the gap is still significant between the DSOM system and others in terms of reaction time to the attack traffic. These results are understandable because:

- In the single SOM case, SOM modules have to send *of − modification* messages to the switch to apply policy for each attack client, and this process takes a time to complete the task.
- In the modified DSOM system, whenever the SOM map at the switch detects a malicious client, it has to send the client's information to the Policy Enforcement at the DSOM Controller. Then, the DSOM Controller will send back the switch *of − modification* messages to apply rules. Therefore, these procedures are completed in a considerable time.
- Meanwhile, in our proposed DSOM system, if the SOM map at the switch detects an abnormal client, it will immediately send client's information to the Fast Policy Enforcement in order to install rules and also inform the Policy Checking module at the DSOM Controller Agent for further modification if needed.

In conclusion, the proposed DSOM system presented its efficiency as the best solution among others in reacting to flooding attacks.

5.3. SOM performance evaluation

Before we assess the DSOM approach performance, we note that: True Positive (TP) is the probability of attack clients that are classified as illegal users, True Negative (TN) presents the probability of legitimate clients that are considered as normal users, False Positive (FP) reflects the probability of abnormal users that are believed to be normal users, and False Negative (FN) shows the probability of legal users that are recognized as attack clients.

5.3.1. Detection rate

To evaluate the efficiency of the proposed DSOM system, we consider one of the crucial criteria, that is the Detection Rate. We performed several tests as discussed in Section 4.3.3 and achieved results. Then, we compute the Detection Rate as follows:

$$DR = \frac{TP}{TP + FN}, \tag{11}$$

Fig. 13 all depict the detection rate of the SOM maps in three cases with different four datasets. From the experiment, we can show that

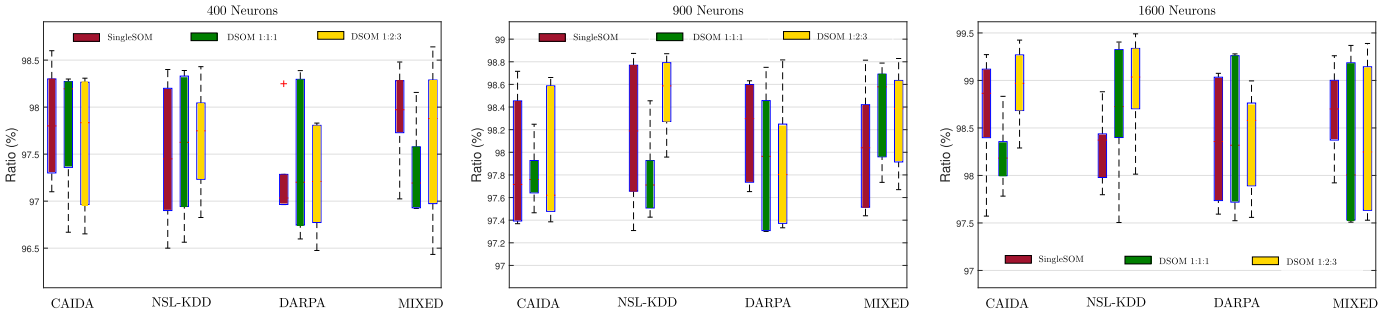

Fig. 13. The detection rate of the SOM maps with three different neuron numbers and four datasets.

with the same number of neurons in the SOM map, three cases only have slight differences from each other in all four datasets. This is because we use the technique k-fold cross validation to assess the detection rate on the datasets themselves. Even if we use the attack tool to generate traffic; however, the trained SOM can easily detect attack clients because of the high number of training samples. Furthermore, these experiments also show that if the SOM map has more neurons, this implies that the detection performance is higher, and there is no great difference in comparison between the proposed DSOM and the single SOM systems.

5.3.2. Accuracy

The second criterion used to assess the DSOM system and the single SOM is the Accuracy. This criterion presents how accurately the SOM map makes a decision, and is formed as

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{12}$$

Fig. 14 demonstrates that in general, the Accuracy of all three experiments is always accountable for high ratios. In the case of 400 neurons, the lowest rate of accuracy is around 96.5%, while it is above 97% for both case of 900 and 1600 neurons. The reason is that when the number of learning samples for the SOM map is increasing, the accuracy of making a decision on a client is also more accurate. And, when the SOM map reaches to a threshold where it can make totally accurate decisions, the number of neurons does not greatly affect to the accuracy. That is the reason why there are only minor differences between tests in terms of the accuracy.

5.3.3. False alarm rate

The next criterion that should be taken into account is the False Alarm Rate, which depicts the probability of falsely making a decision. This criterion plays a key role in evaluating both the DSOM system and the single SOM performance. The false alarm rate is calculated as:

$$FA = \frac{FP}{FP + TN} \tag{13}$$

Fig. 15 illustrates the results of the false alarm rate for the experiments. At a glance, the figure shows a decreasing trend of the false alarm rate with the increasing number of neurons. This is reasonable because with the same testing conditions, the map having the higher number of neurons produces more mathematical calculation; therefore; it will make less wrong decisions on users. In addition, there is no significant difference between two systems in case of having the same neuron number in the SOM map.

5.3.4. The DSOM system overhead

We additionally value the performance of the DSOM mechanism and the single SOM by measuring the processing and classification times for both systems. For the DSOM system, we assume that the times taken to transfer data via socket connections are just around some milliseconds as we examined in Section 5.2; consequently, the processing time is mainly calculated by the summary of two procedures: the first is training DSOM maps at OpenvSwitches, which takes the highest time value, and the latter is the merging time at the DSOM Controller. Meanwhile, the processing time of the single SOM is just its training time. The DSOM and single SOM maps are trained with input sets as shown in Table 6 of Section 4.3.3.

The results in Table 7 show that the processing time of the DSOM system is faster than the single SOM, if we use a greater number of neurons in the SOM maps. This is easy to understand because the DSOMs at OpenvSwitches use smaller sets for training, while the single SOM always trains its maps with larger input sets. Regarding the classification time, all three cases are approximately equal to each other, and they increase in direct proportion to the neuron numbers. This can be explained by using the same configuration for the Controller and OpenvSwitches; thus, the CPU processing time shows no great difference between the three cases.

In conclusion, through several evaluation criteria: handling performance bottleneck, attack reaction performance, SOM detection rate and the system overhead, we now can assess the DSOM system performance in Software-Defined Networks in comparison with the single SOM. Firstly, the DSOM system can be considered as a great solution that totally outweighs the sing SOM scheme in terms of handling performance bottleneck problem and reducing time of attack reaction. Moreover, both systems show equal achievements regarding the detection rate, accuracy and false alarm rate for abnormal traffic.

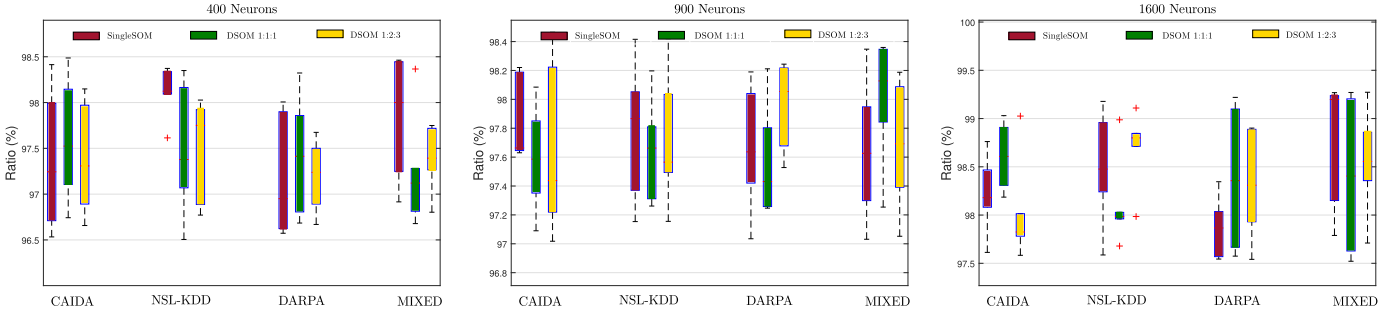

Fig. 14. The accuracy of the SOM maps with three different neuron numbers and four datasets.

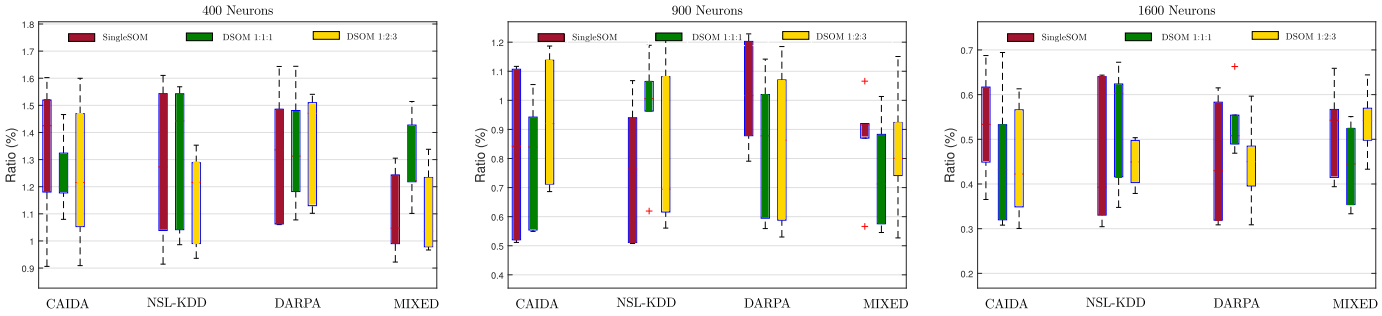

Fig. 15. The False Alarm Rate of the SOM maps with three different neuron numbers and four datasets.

Table 7
Processing and Classification Time.

Neurons	Cases	Processing (*s*)	Classification (*ms*)
400	SingleSOM	10.14	0.956
	DSOM 1:1:1	9.16	0.932
	DSOM 1:2:3	9.81	0.895
900	SingleSOM	25.95	1.82
	DSOM 1:1:1	15.71	1.81
	DSOM 1:2:3	18.04	1.70
1600	SingleSOM	37.77	4.55
	DSOM 1:1:1	23.99	4.03
	DSOM 1:2:3	28.72	4.45

One more crucial point is that the proposed DSOM overhead is less than the single SOM in making the system ready for the *Classifying* procedure. Overall, the DSOM system in Software-Defined Networks can effectively solve the bottleneck problem under flooding attacks in a large-sized network and provide some advantages if compared to the traditional method.

6. Related work

In this section, we look at SDN security studies related to detection and prevention of flooding attack mechanisms. Note that while all the researches discussed below motivate us, our proposed mechanism is completely different and it is originally extended from our previous work (Minhoe et al., 2015).

In the SDN, the detachment of the control and data planes creates some weaknesses for the SDN model, which can result in a serious flooding attack not only on the controller, but also on the flow-table of the switch. Numerous methods have been proposed to tackle and minimize these weaknesses. As with the bottleneck problem, AVANT-GUARD (Shin et al., 2013) is proposed to overcome the mentioned issue by using a connection migration tool to limit the amount of traffic between the control and data planes while the system is under flooding attacks. Meanwhile, Fonseca et al. created CPRecovery (Fonseca et al., 2012), which provides a backup solution for the SDN controller when the primary controller fails under flooding attacks. The ident++ protocol illustrated in Naous et al. (2009) could provide the controller with an effective approach against saturation attacks. VAVE (Yao et al., 2011) is implemented in OpenFlow protocol to resolve IP spoofing issues in notorious attacks such as TCP SYN flooding (Geetha and Sreenath, 2014) and DNS amplification (Fachkha et al., 2014) using a global view. Another solution called CONA is presented by Suh et al. (2010), in which an OpenFlow agent is considered as a mediator between host content requests and a content server. The CONA is responsible for limiting the rate of requests to the content server in flooding attacks. One more feasible method is the usage of OpenFlow and LISP (YuHunag et al., 2010) to record authorized and malicious sources in flooding attacks. If the volume of traffic is over a predetermined threshold, the controller takes drop actions to attack flows. Van Trung et al. (2015) introduced a new DDoS prevention mechanism based on Fuzzy Interference System, and he also proposed an optimal solution for handling flooding attacks in SDN using Support Vector Machine appliance (Phan et al.). Yunhe et al. (2016) proposed a novel mechanism, namely SD-Anti-DDoS: Fast and Efficient DDoS Defense in Software-Defined Networks.

In Sonchack et al. (2016), Sonchack introduced a framework named OFX, which allows extension modules to integrate into a hardware OpenFlow Switch. This proposal also motivates our introduced DSOM mechanism to be practical and feasible for implementation in the foreseeable future, with large-scale cooperation between software and hardware vendors.

7. Conclusion

In this paper, we propose a Distributed Self Organizing Map system for handling the bottleneck issues because of aggregation caused by flooding attacks at the upper layers in the large-scale Software-Defined Networks by using a distributed system instead of a centralized one. We conduct our experiments by using multiple datasets and several evaluation criteria. Through our experimental results, we show that the proposed approach totally outweighs the traditional method in handling performance bottleneck and increasing the system reaction speed to the attack. Moreover, the experiments not only show the performance of our mechanism is the same as that of the single system, but also indicate that the DSOM system overhead outperforms the single SOM. We hope that our research will introduce an effective and feasible approach to the SDN security community in the future.

Acknowledgements

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2016-H8501-16-1008) supervised by the IITP(Institute for Information & communications Technology Promotion)

References

Alsulaiman, M., Alyahya, A., Alkharboush, R., Alghafis, N., 2009. Intrusion Detection System Using Self-Organizing Maps. In: IEEE Proceedings of the 3rd International Conference on Network and System Security; p.397-402.
BoNeSi. the DDoS Botnet Simulator, ⟨https://github.com/markus-go/bonesi⟩.
Braga, Rodrigo, Mota, Edjard, Passito, Alexandre, 2010. Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow. In: Proceedings of the 35th Annual IEEE Conference on Local Computer Networks, Denver, Colorado; p.408-415.
CAIDA Dataset. DDoS Attack 2007, ⟨https://data.caida.org/datasets/security/ddos-20070804/⟩.
CAIDA Dataset. Anonymized Internet Traces 2015, ⟨https://data.caida.org/datasets/passive-2015/⟩.
Cisco IOS Embedded Event Manager (EEM) ⟨http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-embedded-event-manageeem/⟩.
KDD Cup 1999 Data. ⟨http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html⟩.

Fachkha, Claude, Bou-Harb, Elias, Debbabi, Mourad, 2014. Fingerprinting Internet DNS Amplification DDoS Activities. In: Proceedings of the 6th International Conference on New Technologies, Mobility and Security (NTMS); p.1-5.
FERRO, G. Big Switch Networks Launches Mature Hardware-Centric Data Centre SDN Solution, July 2014. ⟨http://etherealmind.com/big-switchnetworks-launches-hardware-centre-mature-data-centre-sdn-solution/⟩.
Floodlight is an Open SDN Controller, 2017. ⟨http://www.projectfloodlight.org/floodlight/⟩.
Fonseca, P., Bennesby, R., Mota, E., Passito, A., 2012. A replication component for resilient OpenFlow-based networking. In: IEEE Network Operations and Management Symposium (NOMS); p.933-939.
Geetha, K., Sreenath, N., 2014. SYN flooding attack – Identification and analysis. Inf. Commun. Embed. Syst. (ICICES), 1–7, (p).
Gu, Qijun, Liu, Peng, 2012. Denial of Service Attacks, John Wiley and Sons, p.454-468, Part II (in book: Handbook of Computer Networks: Distributed Networks, Network Planning, Control, Management, and New Trends and Applications).
Gunes Kayacik, H., Zincir-Heywood, N., 2007. A hierarchical SOMbased intrusion detection system. Eng. Appl. Artif. Intell. 20 (4), 439–451.
Hassas Yeganeh, S., Ganjali, Y., 2012. Kadoo: A Framework for Efficient and Scalable Offloading of Control Application. In: Proceeding HotSDN Conference; p.19-24.
HP 3800 Switch Series. ⟨https://www.hpe.com/h20195/v2/getpdf.aspx/c04111485.pdf?Ver=19⟩.
Jain et al., S., 2013. B4: Experience with a Globally-Deployed Software Defined WAN. In: Proceeding ACM SIGCOMM 2013 Conference; p.3-14.
Jankowski, D., Amanowicz, M., 2015. Intrusion detection in software defined networks with self-organized maps. J. Telecommun. Inf. Technol. 1 (4), 3–9.
Jiang, D., Yang, Y., Xia, M., 2009. Research on Intrusion Detection Based on an Improved SOM Neural Network. In: IEEE Proceedings of the Fifth International Conference on Information Assurance and Security; p.400-403.
John Pescatore. DDoS Attacks Advancing and Enduring: A SANS Survey 2014, ⟨https://www.sans.org/reading-room/whitepapers/analyst/ddos-attacks-advancing-enduring-survey-34700⟩.
Kaspersky Lab. Statistics on botnet-assisted DDoS attacks in Q1 2015, Technical Report, 2015, https://securelist.com/blog/research/70071/statistics-on-botnet-assisted-ddos-attacks-in-q1-2015.
Kiziloren, T., Germen, E., 2007. Network traffic classification with Self Organizing Maps. In: Proceedings of the 22nd International symposium on computer and information sciences; p.1-5.
Kohonen, T., 1990. The self-organizing map. In: Proceedings of the IEEE, p.1464–1480.
Kohonen, Teuvo, 2001. The Basic SOM. Springer-Verlag Berlin Heidelberg, p.105-176, Chapter 3 (in book: Self-Organizing Maps).
Koponen et al., T., 2010. Onix: A Distributed Control Platform for Large-Scale Production Networks. In: Proceeding OSDI Conference; p.1-6.
LANDER:DARPA 2009 DDoS attack-20091105. ⟨https://ant.isi.edu/datasets/readmes/DARPA-2009-DDoS-attack-20091105.README.txt⟩.
Langin, C., Zhou, H., Rahimi, S., Gupta, B., Zargham, M., Sayeh, M., 2009. A Self-Organizing Map and its Modeling for Discovering Malignant Network Traffic. In: IEEE Symposium onComputational Intelligence in Cyber Security; p.122-129.
Masoudi, Rahim, Ghaffari, Ali, 2016. Software defined networks: a survey. J. Netw. Comput. Appl. 1 (67), 1–25. http://dx.doi.org/10.1016/j.jnca.2016.03.016.
Medhi, D., 2007. Network Routing: Algorithms, Protocols, and Architectures 1st ed.. Morgan Kaufmann.
Kim, Minhoe, Souhwan Jung, Minho Park, 2015. A Distributed Self Organizing Map for DoS Attack Detection. In: Proceedings of the 7th International Conference on Ubiquitous and Future Networks (ICUFN); , p.19-22.
Mitrokotsa, A., Douligeris, C., 2005. Detecting denial of service attacks using emergent self-organizing maps. In: IEEE Signal Processing and Information Technology; p. 375-380.
Naous, J., Stutsman, R., Mazieres, D., McKeown, N., Zeldovich, N., 2009. Delegating network security with more information. In: Proceedings of the 1st ACM workshop on Research on enterprise networking; p.19-26.
Nascimento, Z., Sadok, D., Fernandes, S., 2013. A Hybrid Model for Network Traffic Identification Based on Association Rules and Self-Organizing Maps (SOM). In: Proceedings of the Ninth International Conference on Networking and Services; p. 213-219.
NSL-KDD Data Set for Network-based Intrusion Detection Systems, 2009. ⟨http://iscx.cs.unb.ca/NSL-KDD/⟩.
Omnipeek Network Analysis, 2017. ⟨https://www.savvius.com/products/network-visibility-performance-diagnostics/omnipeek-family/omnipeek-network-analysis⟩.
Open network linux, 2017 ⟨http://opennetlinux.org⟩.
OpenFlow. OpenFlow Switch Specification Version 1.4.0, 2013 ⟨https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/op-enflow/openflow-spec-v1.4.0.pdf⟩.
OpenvSwitch. OpenvSwitch version 2.3.0, 2014 ⟨http://openvswitch.org/releases/openvswitch-2.3.0.tar.gz⟩.
Park, T., Kim, Y., Shin, S., 2016. UNISAFE: A Union of Security Actions for Software Switches. In: the SDN-NFVSec Workshop, p.13–18.
Pica os white box switch os, 2017. ⟨http://www.pica8.com/white-box-switches/white-box-switch-os.php⟩.
Ramadas, M., Ostermann, S., Tjaden, B., 2003. Detecting anomalous network traffic with self-organizing maps, Recent Advances in Intrusion Detection, Springer-Verlag Berlin Heidelberg, p.36–54, Chapter 1 (in book: Recent Advances in Intrusion Dection).
Shin, S., Yegneswaran, V., Porras, P., Gu, G., 2013. AVANT-GUARD: scalable and vigilant switch flow managament in software-defined networks. In: Proceeding of the 2013 ACM SIGSAC Conference on Computer and Commications Security; p.413-424.
Sonchack, J., Aviv, Adam J., Keller, E., Smith, M., 2016. Enabling Practical Software-defined Networking Security Applications with OFX. In: The Network and Distributed System Security Symposium (NDSS); p.181-195.
Suh, J., Choi, H., Yoon, W., You, T., Kwon, T., Choi, Y., 2010. Implementation of Content-oriented Networking Architecture (CONA): A Focus on DDoS Countermeasure. In: European NetFPGA Developers Workshop.
Kanungo, Tapas, Mount, David M., Netanyahu, Nathan S., Piatko, Christine D., Silverman, Ruth, Wu, Angela Y., 2002. An efficient k-means clustering algorithm: analysis and Implementation. IEEE Trans. Pat. Ana Ma Intel. 24 (7), 881–892.
Ting, B., Yong, W., Xiaoling, T., 2010. Network Traffic Classification Based on Kernel Self-Organizing Maps. In: International Conference on Intelligent Computing and Integrated Systems; p.310-314.
Trung, V.Phan, Van Toan, T., Van Tuyen, D., Huong, T.T., Thanh, N.H.,. OpenFlowSIA: An optimized protection scheme for software-defined networks from flooding attacks. In: Proceedings of the Sixth International Conference on Communications and Electronics, IEEE, p. 13-18.
Van Trung et al., P., 2015. A Multi-criteria-based DDoS attack prevention solution using Software Defined Networking. In: IEEE The International Conference on Advanced Technologies for Communications, p.308-313.
Wang, C., Yu, H., Wang, H., 2009. Grey self-organizing map based intrusion detection. Optoelectron. Lett. 5 (1), 64–68.
Wang, H., Xu, L., Gu, G., 2015. FloodGuard: A DoS Attack Prevention Extension in Software-Defined Networks. In: Proceeding IEEE/IFIP Proceedings of the 45th International Conference on Dependable Systems and Networks (DSN); p.239-250.
Yao, G., Bi, J., Xiao, P., 2011. Source address validation solution with OpenFlow/NOX architecture. In: Proceedings of the 19th IEEE International Conference on Network Protocols (ICNP); p.7–12.
YuHunag, C., MinChi, T., YaoTing, C., YuChieh, C., YanRen, C., 2010. A novel design for future on-demand service and security. In: IEEE International Conference Communication Technology (ICCT); p.385-388.
Yunhe Cui, Yan, Lianshan, Li, Saifei, Xing, Huanlai, Pan, Wei, Zhu, Jian, Zheng, Xiaoyang, 2016. SD-Anti-DDoS: Fast and Efficient DDoS Defense in Software-Defined Networks. Journal of of Network and Computer Applications, http://dx.doi.org/10.1016/j.jnca.2016.04.005.
Zanero, Stefano, 2005. Analyzing TCP Traffic Patterns Using Self Organizing Maps. In: Proceedings of the 13th International Conference on Image Analysis and Processing; p.83-90.