Journal of Network and Computer Applications 80 (2017) 152–164

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

Identifying cyber-attacks on software defined networks: An inference-based intrusion detection approach

Ahmed AlEroud[a,*], Izzat Alsmadi[b]

[a] Yarmouk University, Jordan
[b] University of Texas A&M, San Antonio, USA

ARTICLE INFO

Keywords:
Software defined networks
Information security
Intrusion detection
Graph mining, Denial of service attacks
Security architecture

ABSTRACT

Software Defined Networking is an emerging architecture which focuses on the role of software to manage computer networks. Software Defined Networks (SDNs) introduce several mechanisms to detect specific types of attacks such as Denial of Service (DoS). Nevertheless, they are vulnerable to similar attacks that occur in traditional networks, such as the attacks that target control and data plane. Several techniques are proposed to handle the security vulnerabilities in SDNs. However, it is fairly challenging to create attack signatures, scenarios, or even intrusion detection rules that are applicable to dynamic environments such SDNs. This paper introduces a new approach to identify attacks on SDNs that uses: (1) similarity with existing attacks that target traditional networks, (2) an inference mechanism to avoid false positives and negatives during the prediction process, and (3) a packet aggregation technique which aims at creating attack signatures and use them to predict attacks on SDNs. We validated our approach on two datasets and showed that it yields promising results.

1. Introduction

In network security, autonomous network agents learn their network topology and the nature of threats in their network to write or update their roles with the least amount of human effort. There are many learning and evolutionary activities that the complete network agent may require. The amount of automation in those activities depends on the complexity of the design of those agents, their network, goals, etc. The recent evolution in programmable networks such as Software Defined Networks (SDNs) opens the possibility to extend the automated activities to ultimately build network agents that are fully autonomous in the network.

Despite the opportunities and benefits of SDNs, practitioners and researchers argue that SDNs are vulnerable and easier to target (Kreutz et al., 2013; Sezer et al., 2013; Nunes et al., 2014). When the logic of the forwarding behavior is centralized and allocated in the controller, a single point of failure and attack is created. Both exploiting the vulnerabilities in the controller or the communication links between the switch and controller can lead to several attacks such Denial of Service (DoS) (Cui et al., 2016) and Host Location Hijacking Attacks (Hong et al., 2015). Man in the Middle (MIM) is yet another potential attack in which the adversary may break the link between the controller and its switches, then claim the control of such connection (Hong et al., 2015). To summarize, there are possible attack scenarios that make the current architecture of SDN non-secure, which requires more attention to various security aspects of SDNs.

Yet, there are several challenges when creating security techniques to mitigate such attacks on SDNs. First, there is a need to create sophisticated security measures which do more than just discarding or forwarding flows. Specifically, statistical and signature matching techniques need to be combined with the flow rule production. In particular there is a need to “implement complex quarantine procedures of the flow producer, or they could migrate a malicious connection into a counter-intelligence application in a manner not easily perceived by the flow participants” (Hong et al., 2015). Second, the existing procedures which are used to create security applications depend on the architecture of the controllers used to manage the network. The implications of the current controller architectures are equally problematic for implementing security mediation services. In particular, the element responsible for security mediation should operate independently from those elements it mediates (Hong et al., 2015). As such, there is a need to create platform independent security applications for SDNs. Third, since handling traffic in SDNs is quite dynamic, it becomes difficult to discover different types of attacks using anomaly detection techniques since they will lead to high percentage of false positives. Finally, there is an increasing number of Zero-day attacks and SDNs are not protected against such attacks. There have been some attempts to create anomaly detection techniques that discover Zero-day attacks such as the one

* Corresponding author.
E-mail addresses: Ahmed.aleroud@yu.edu.jo (A. AlEroud), ialsmadi@tamusa.edu (I. Alsmadi).

http://dx.doi.org/10.1016/j.jnca.2016.12.024
Received 1 July 2016; Received in revised form 8 November 2016; Accepted 15 December 2016
Available online 22 December 2016
1084-8045/

created by AlEroud and Karabatis (2012), however, the main challenge remains if it is possible to rely solely on anomaly detection techniques to identify such attacks.

Addressing these challenges requires mitigation techniques that work on top of the SDN controller. In addition, such techniques need to provide reasoning mechanisms to discover relationships between attacks that occur in similar context. This paper introduces a novel intrusion detection approach that can be used to identify attacks on SDNs. We designed an approach that is consistent with the existing OpenFlow models. A Markov-based graph model, with known DoS attacks as nodes and relationships between them as edges, is created and used to discover attacks that may target SDNs.

Research on security of SDNs focuses on utilizing data mining techniques to create attack prediction models (Choi, 2010; Yuzawa, 2013; Schehlmann and Baier, 2013; Giotis et al., 2014). However, since only the header information is available at the time of prediction, such techniques lead to high rate of false positives. Our approach handles such a limitation using the relation context, not only the features of the flow are important to classify it as an attack, but also the relationship between that predicted attack and other attacks that occur in similar contexts. The proposed approach aggregates traditional packets to create flow-based datasets that contain attack signatures, which are also used to create attack prediction techniques. At run time, the OpenFlow-based flows are initially classified using a k-NN classifier which discovers the k-similar flows. Graph nodes are also represented as feature vectors that have the same structure of flows. Therefore, each OpenFlow-based flow is analyzed to discover the most similar node from the created graph using feature similarity. Given that node, the graph is traversed to discover other k nodes that may occur in similar context. Finally, the flow is classified as an attack or a benign activity depending on the types of traversed nodes.

We tested our approach on two datasets. Our graph-based mechanism outperforms the typical distance-based classification approaches such as k-NN. A comparison with several classification approaches proved the enhancements achieved using our approach. In particular, this work contains several significant contributions:

1. We proved the feasibility of using the existing attack signatures to identify attacks on SDNs.
2. We created a flexible approach which can uses datasets that contain labeled events at different levels to create attack prediction models that identify attacks on SDNs. Specifically, we introduced a flow creation scheme which aggregates packets to create precise flow-based attack signatures and coherent events that are contextually related, thus, identify potential attacks which cannot be discovered using the traditional anomaly detection techniques.
3. We showed that our approach can be efficiently applied to identify attacks on SDNs by querying pre-created graphs at run time. Our results have shown that our approach outperforms the traditional classifiers and the existing SDNs' attack classification techniques.
4. We proposed a semi-automated technique to expand the created graph by adding new types of nodes using their relationships with the existing nodes.

The paper is organized as the following: Section 2 discusses the motivations of this research. Section 3 discusses several existing techniques on the security of SDNs. Section 4 introduces the proposed approach. Section 5 discusses the results of our experiments. Section 6 discusses the implications of our research. Finally, Section 7 concludes the paper and discusses the future work.

2. Research motivations

Research has shown that it is feasible to attack SDNs. Shin et al. (2015) categorize attacks on SDN into three categories as shown in Fig. 1: *Control plane specific*, *Control channel specific*, and *Data plane specific*. Several attacks are possible according to this categorization. For instance one possible attack is to exploit the vulnerabilities in the packet forwarding process to decide if a specific network uses OpenFlow Switches, then generate crafted flow requests from the data plane to the control plane. Such an attack consumes the resources of the control plane and makes it hard to handle all requests, leading to a DoS attack. DoS attacks can focus on either the controller and overwhelm it with artificial (typically large in numbers and size) service calls. They can also target the communication between SDN and its switches. Several useless flow rules that need to be stored by the data plane make it harder to manage the resources of the data plane which also leads to other forms of attacks such as Freeloading, where attackers try to bypass the rule installation step in order to target the network (Park et al., 2016).

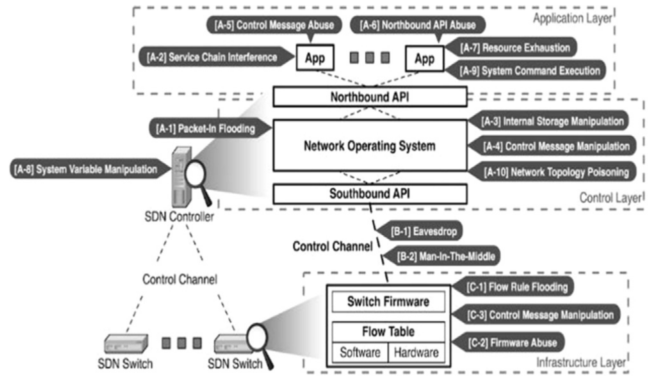


Fig. 1. SDN misuse/attack cases: reproduced from Shin et al. (2015)

Another feasible attack scenario is the Fingerprinting in which the attacker needs to decide if the packets are sent to a SDN. In particular, when the data is sent to a SDN users may experience differences in the response time since the new flows require a setup time before they can be added to existing flows. To initiate such an attack, attackers need to collect data about the targeted network. Statistical tests that show the differences between the response time between new and existing flows can tell if the network has a controller. Using specific types of network scanners that work in SDNs, the attacker may also decide to further attack the network and initiate DoS, MIM (Wang et al., 2014), Firewall Policy Violation (Hu et al., 2014), or even Poisoning Network Visibility attacks (Hong et al., 2015). Several DoS attack scenarios can be handled using multiple controllers approach, however, multiple controllers are still vulnerable to distributed DoS attacks (Yan and Yu, 2015). Another possible solution to DoS attacks is to drop all table miss packets. Yet, by simply dropping those packets we may also drop the benign ones.

To mitigate DoS attacks IDSs can be developed on top of the SDN controller in order to make intelligent and real time decisions for network traces as they go through the network. The controller takes two actions based on incoming traffic. The first action is the actual decision on what to do with the subject traffic (e.g. drop, forward, flood) and the second is writing a flow rule in the switch flow table as a response to the subject traffic. We believe that IDSs for SDN should perform similar tasks but on a broader context. The single or simple flow in the case of control-flow tables should be extended to include complex flow in an inference engine. This motivates us to introduce a graph-based inference mechanism which does not only identify potential DoS attacks on SDNs but is also extendable to identify other types of attacks.

3. Related work

3.1. SDNs

Earlier motivations for the proposal of SDNs were related to the need for students, educators or researchers to evaluate and experiment different protocols and their impact on the network. Another motivation is to create networks that can respond to incidents and other changes on the network traffic at real time (Yoon et al., 2015; Alsmadi and Xu, 2015). Kim and Feamster (2013) discuss the complexity of managing classical firewalls. In a middle to large size company, a typical firewall can have hundreds of rules written by different network administrators over a significantly long period. There are no practical solutions to continuously screen those firewall rules for possible rules' obsolescence, conflicts, etc. Classical network components (e.g. routers, switches, etc.) are also controlled by software. However, this software is typically embedded within the network component and is vendor closed (McKeown, 2009). In other words, users do not have the ability to view, customize or interact with the underlying network at run time and control for example traffic flow or access control.

3.2. Security weaknesses and opportunities in SDN

In SDN security, research papers focus on two different aspects: security challenges and problems related to SDN, and on the other side security opportunities that SDN creates in terms of changing the way security controls are developed (Shin et al., 2013; Liu et al., 2016; Wang et al., 2015; Ding et al., 2014; Skowyra et al., 2013; Tsugawa et al., 2014; Van Trung et al., 2015; Hall et al., 2016). SDN manages the network resources in a way that can be used when creating attack detection and prevention systems (Mehdi et al., 2011; Dabbagh et al., 2015). A new aspect SDN brings to network monitoring and control is the ability to control traffic at the fine-grained level. Yet, there have been several concerns about the feasibility of using SDNs for network security. The SDN itself is vulnerable making it not highly reliable for creating security applications (Zaalouk et al., 2014). SDN is seen as a vulnerable network or architecture to cyber-attacks especially in reactive modes. SDN allows flow rules to be written in switches manually similar to classical networks or in a reactive mode by the SDN controller based on real time traffic. While it is an important feature in SDN to support dynamic and responsive security controls, this is also considered as a vulnerability that can be exploited by attackers or abused by legitimate users.

3.3. Security threats in SDN

Shin and Gu (2013) and YuHunag et al. (2010) have shown that it is quite feasible to attack SDNs. In addition, Alsmadi and Xu (2015) have conducted a survey on the security aspects of SDNs. They categorize those aspects into attacking techniques and security controls. They found that several attacks are feasible when initiated on SDNs including: Spoofing, DoS, Privilege Escalation, Tampering, Information Disclosure, and Repudiation (Kloti et al., 2013). In Tasch et al. (2014), the author found several vulnerabilities that are related to SDNs. In one aspect, the evaluation of the encryption of communication between the controller and switches showed that such an encryption is ineffective and optional. In principle, intruders can have the possibility to hijack the relation between controller and switch and claim controlling the switches. If a switch crashes or is forced to reboot, illegitimate identity of the controller can be claimed by intruders.

Benton et al. (2013) conducted a vulnerability assessment for SDN. The authors pointed to several possible vulnerabilities in SDN. In terms of DoS, the centrality of the controller seems to get a lot of concern since it creates a single point of attack where attackers can flood the controller with traffic which will eventually leads to flood tables in switches and also the communication between the controller and switches. The idea of SDN based on demand security services is introduced in several papers (Shin et al., 2013; YuHunag et al., 2010). It is envisioned that in future, security services can be called on demand, whenever required and for the exact problem or requirement purposes. For example, DoS security control can be developed to only detect and counter possible DoS and flooding attacks. If a new virtual image, network, data center, etc. is initiated, customized security services can be developed to only take care of this new installation. Similarly, once this network or computing resources is removed, security services allocated to those resources can be automatically removed.

3.4. DoS attacks

Several related works discuss the mitigations against DoS and botnet attacks. There are few techniques which have been proposed to detect botnets and DoS attacks on SDNs (Choi, 2010; Yuzawa, 2013; Schehlmann and Baier, 2013). Choi (2010) propose a content-based networking architecture which can be implemented in an OpenFlow platform. The approach introduces a content-based mechanism to deal with accountability. A router, which works as an agent, is used to figure-out what content is requested. The content is received from the source host, then the agent forwards it to the destination host. The approach has been used to mitigate resource consumption attacks such as DoS and botnet attacks. While such approaches address accountability, they introduce more overhead in attack mitigation procedures, which is very costly in a dynamic environment such as SDNs.

3.5. Flow-based intrusion detection

One of the promising techniques in cyber-attack detection is using Netflow instead of packet content analysis for attack discovery. In Sperotto et al. (2010) introduce a comprehensive survey on Netflow-based intrusion detection techniques. Braga et al. (2010) introduce a method to identify DoS attacks in SDNs using feature similarity and signature matching. The proposed technique requires normal and suspicious traces to identify attacks by analyzing incoming flows. DoS flow-based security control and detection is also introduced in several other research papers. Flow-based detection methods seem to have better visibility to the attacks compared to classical content based detection methods (Zhang, 2016). There are some authors who argue about the benefits of using flow features for traffic monitoring. In traditional networks it has been shown that analyzing netFlows to discover cyber-attacks introduces several challenges since packet content is not available at the time of analysis. Similarly, packet content is not directly accessible when using the current OpenFlow standards, which introduces several difficulties when creating IDSs for SDNs or OpenFlow security applications.

Shirali-Shahreza et al. (2013), have introduced an approach which extends the existing OpenFlow standards to access packet payload information by the controller. The controller selects which parts of packets need to be sent and where to send them. In addition, packets can be sampled with pre-identified probability or deterministically to provide more flexibility when analyzing the selected traces. There is a major limitation in such techniques. In fact, the majority of those techniques rely on statistical anomaly detection models to identify attacks. Such models require large number of existing benign and suspicious activities. However, in new environments such as SDNs, the signatures of the discovered attacks are still not sufficient to create accurate attack prediction models.

3.6. Using existing IDSs to identify attacks on SDNs

The recent trend in research on SDNs security is to adapt existing IDSs to identify attacks on SDNs (e.g. Ballard et al., 2008; Xing et al., 2013). IDSs such as Snort need to access packet payload, which is impractical and contradicts with the motivations of introducing SDNs. Handling every single packet to the controller is not feasible and has significant effects on the efficiency of SDNs. Controllers can only handle the first few packets of each flow; then the rest of packets are

Table 1
Detecting DoS in SDNs.

Approach	Category	Detection Technique	Algorithm
Braga et al. (2010)	Machine learning	Signature/ Misuse Detection	Self-Organizing Map
Delgado et al. (2014)	Network Re-configuration	High level security policies	Congestion triggers
Shin et al., (2013)	Connection migration	Data plane's statistics	Actuating triggers
Giotis et al. (2014)	Machine learning	Anomaly detection	Entropy-based anomaly detection
da Silva et al. (2015)	Machine learning	Signature/Misuse Detection	Principal Component Analysis (PCA) and Support Vector Machines (SVMs)
Phan et al. (2016)	Machine learning	Signature/Misuse Detection	Support Vector Machines

handled by switches which contain specific rules with an idle timeout if the rule is not used. In addition, Snort cannot be directly integrated with the controller without having significant effects on the performance of the network. There are some proposals to include basic security mechanisms including IDS in switches. For instance, ALARMS language (A Language for Arbitrary Route Management for Security) is introduced by Ballard et al. (2008) to steer traffic for monitoring and security purposes. Table 1 provides a summary of the approaches that are used to identify DoS in SDNs, including the category of each approach, the detection techniques used, and the created algorithms.

There are some proposals on a small scale (e.g. Skowyra et al., 2013; Chung et al., 2013) to implement SDN based IDS where attacks are detected and mitigated dynamically without human intervention. However, extending this to a large scale IDS that can handle a large spectrum of different attacks is still a long pending research area. A report by Kerner (2012) discussed the feasibility of implementing an SDN based IDS. The author showed the utilization of SDN for a university IDS.

There have been some attempts to mitigate attacks on SDNs using graph-based techniques. For instance in Chung et al. (2013) a network-based IDS called NICE that uses the functionalities of OpenFlow is presented to mitigate attacks on virtual networks. An attack graph is created periodically using the output of vulnerability scanners and other reasoning mechanisms. At run-time the system may enter the inspection state if an attack path is triggered. In the inspection state Deep Packet Inspection (DPI) is conducted and decisions can be made based on the results. The significant contribution of the paper is the utilization of SDN in creating network-based intrusion detection systems. An extension to NICE has been proposed by Xing et al. (2013) where an approach to integrate Snort with OpenFlow networks is proposed to reconfigure the network so that it can re-route or discard suspicious packets that are discovered by Snort. Several components between the two systems are the same. The earlier system (i.e. NICE) was missing a module that acts as Snort for sniffing and monitoring. The authors used Snort for coordinated attacks' detection which was not possible in early system.

3.7. Using packet payload to identify attacks on SDNs

Recently, there are several researchers who focus on layer 2 and layer 3 information without going into deep packet inspection. In Shirali-Shahreza et al. (2013) and Shirali-Shahreza and Ganjali (2013) authors propose an approach which lets the controller to access packet payload. In the current specification of SDNs, the controller can only access header information that is only related to routing. In addition, the controller receives only samples of the traffic. While there are several sampling approach in literature (Ha et al., 2016), some extensions are still needed to handle these two aspects when creating security applications for SDNs. Extensions include accessing packet content under specific criteria, such as when an initial investigation shows that a specific flow is suspicious (Zhang et al., 2016; Shirali-Shahreza et al., 2013). Traffic Monitoring in SDN is another area of research. In fact, one of the major objectives for utilizing SDNs is packet monitoring and control (Hogg, 2013). For IDSs that work on top of SDNs, customized (Rothenberg et al., 2012) and real time (Chowdhury et al., 2014) traffic monitoring can be the most significant contribution that SDNs can bring to IDSs.

4. The proposed approach

We followed an approach that uses regular labeled flows to analyze different samples of incoming OpenFlow-based flows that are generated by SDNs. The objectives of our approach are as the following:

1) Examining if the existing attack signatures can be used to analyze traces that are generated by SDNs.
2) Finding out how the traditional IDS behaves when handling large number of the traces generated by SDNs.

Fig. 2 shows the following major modules in our attack prediction system.

4.1. Traffic monitor

This module communicates with OpenFlow switches to make customized traffic queries and receive customized traffic that is requested from the network or the switches. This is largely a customized traffic-monitoring module. Hence, this task can be outsourced to applications such as sFlow that can interact with SDN controllers for flow monitoring purposes. Flow records can be then exported and analyzed at run time using attack prediction models.

4.2. Attack signatures

This module is used to represent the attack signatures that are extracted from the traditional network datasets. The network data that

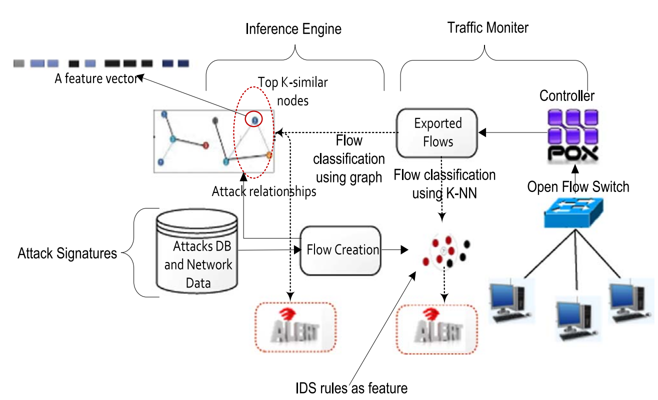


Fig. 2. The research approach.

is stored in attack DB is aggregated to create netFlows using the flow creation module.

4.3. Flow creation

There are several hardware and software specifications to collect and export flows from packet traces. Flows consist of packets that share specific features such as the time of occurrence, source, and destination IP. In particular, the flows created using our approach consist of ten features: source and destination IP, source and destination ports, packet and octet counts, start and end time, TCP flags, and flow duration.

Algorithm 1. Creating a flow, based on aggregating OpenFlow messages.

```
Inputs: duration: specific duration for one flow, default 30 s
  features: a list of features extracted from packet data to create
  one flowpackets: OpenFlow network traces
procedure createFlow(duration, features, packets)
   Initialize a list of packet S to empty
   Initialize a hashmap M, key is flow id, value is a list of
   packet belonging to this flow
   For all packet: packet in packets do
   If S is empty then
   add the packet in S
   Else
   If duration between current packet's start time and start
   time of first packet of S is greater than duration then
   Print out first flow, get first flow from M
   Remove first packet from S
   Remove first flow from M
   End if
   Initialize flag to False
   For each packet startPacket in S do
   If packet has same values on features with startPacket
   then
   M.get(id of startPacket).add(packet)
   Set flag to True
   End if
   End For
   If flag is False then
   add packet to S
   M.put(id of packet, add packet to the list)
   End if
   End if
   End For
If M is not empty then
   print out all flows from M
End if
end procedure
```

In Algorithm 1 we show the steps of aggregating packets to create flows. This process is a preprocessing step and it is only executed when the datasets to be stored in the attack DB are not flow-based. The duration of each flow is adjusted using the duration parameter. The packets are aggregated when they are close based on time stamps. Each created flow has the start time as the first packet's start time in that flow.

Flow-based labeled datasets are rare, therefore, our approach provides a technique to label flows using the corresponding pre-existed, labeled, and packet-based network datasets. The technique is discussed in Ding et al. (2015) and it needs the labels that are produced by IDSs. Using this technique, flows are labeled using the correlation between time stamps of flows and the corresponding alerts that are produced by IDSs. Algorithm 2 shows the steps.

Algorithm 2. Labeling flow by majority rule.

```
Inputs: flow: a flow
Return: type: suspicious or benign type
      procedure Label_Flow(flow)
         For all packet: packet in flow do
         if packet.type is not benign then
         Count each type
         Flow is not normal type;
         End if
         End For
      If there is no suspicious type then
         Return benign
      Else
      Return type with max_count
      end procedure
```

The algorithm takes unlabeled flows as input and outputs labeled flows. Each packet that belongs to a particular flow might be a suspicious or a benign activity. Using the majority rule, the flow takes a single label (the type which is a majority in the flow). For instance, if the flow contains three packets one of them is a benign activity and the other two are TCP flooding attempts, the resulting flow will be labeled as a TCP flooding attempt.

4.4. IDS/rules

IDS rules are accessed and updated by network administrators or using specific machine learning techniques. Our approach performs the required analysis on the exported OpenFlow-based flows in order to decide if they contain suspicious activities. Each flow fl_i is classified as a benign or suspicious activity. In this work we use a flow classification model to create this set of rules. Such rules are created using the regular flows that are stored in the attack DB. Let $\vec{V}=[f_1:d_1, \ \ldots, f_n:d_n]$ be a set of features that are extracted from a network trace (as shown in Table 2). Suppose that a flow classification model m examines $\vec{V}$. An

Table 2
Features extracted from a network trace and their values.

ID	SRC IP	DST IP	PACKETS	SRC PORT	DST PORT	TCP FLAGS	Node (alert) type n_1
155649	108.239.60.192	83.39.140.125	4	113	59346	20	2
155650	113.69.150.12	7.6.81.7	4	113	58085	20	2
155651	72.159.16.47	17.130.149.225	6980	113	42461	20	3
155652	225.101.113.49	165.132.147.120	4	64221	113	2	2
155653	12.160.24.12	29.107.15.54	3069	53152	80	2	3
155654	148.67.0.23	43.48.244.67	14	22	1454	27	1
155655	244.144.214.239	49.129.28.253	1	113	51192	20	2

alert is raised if $\vec{V}$ matches any signature defined by m. Suppose that $N = \{n_1, \dots, n_p\}$ is the set of all probable alerts identified by m. Suppose that NS is the set of labels of pre-existing suspicious flows and NB is the set of labels of pre-existing benign flows, using any flow classification model such as m, two main objectives need to be achieved at run time:

1. Increasing the *Detection Rate* of suspicious activities given the correct labels in the set NS.
2. Decreasing the rate of *False Positives* by correctly identifying benign flows given the correct labels in the set NB.

4.5. Inference engine

This is the core module of the SDN IDS. Rules can vary of complexity between simple flows related ones to more complex pattern recognition algorithms. This module can be further divided into several sub modules in which each sub module is dedicated to one way of detecting security attacks (e.g. signature or inference-based). For example, a dictionary-based module may include a large dataset of known threats in terms of their known ports, IP addresses, etc. There are several problems with traditional IDS inference engines. For example, there is a need for fine grained decisions beyond the (Block/Permit) binary decisions. This is since the majority of IDS decisions may fall in the gray area between those two binary decisions. This is why false positive and negative alarms are considered the most significant problems related to those decisions. When OpenFlow is used, the switch only sends the packet header (first 160 bytes) to the controller. Therefore, we use a graph-based prediction model to compensate the absence of packet content during the prediction process. Given the type (label) of each flow, the relationships between those types are identified on graphs. Each node in the graph represents a specific type of alert or a benign activity. The relationships between nodes are weighted. The weights denote the similarity between nodes in term of their features. The semantics of the created graph are augmented based on a Markov process which discovers the indirect relationships between nodes. Two major steps are used to create those relationships:

4.5.1. Weighted paths are created between nodes

Each node n_i is represented using specific features (i.e. the features of the corresponding flows). Each label in the collected data is treated as a node in the graph. The features of the flows labeled as n_i are used to describe that node. A single feature vector V_{ni} is used to represent each node n_i as shown in Table 3.

Each vector contains the aggregated weight of the features source and destination IP, port numbers, the number of packets, TCP flags, protocol type and alert description. The normalized frequency of each feature f_{i_bj} with each node is used as a weight of f_{i_bj} in that node. Since some of these features are numerical they are discretized using the equal width binning approach which creates the bins $b_1, \dots b_j$ for each feature f_i.

The correlation between each pair of nodes is calculated using the Pearson Correlation as shown in formula 1. Pearson correlation is a widely used similarity measure in intrusion detection (Qishi et al., 2009; Hassanzadeh and Sadeghian, 2008; Beauquier and Hu, 2007). Therefore, we used it to create our similarity-based graph. The resulting weights are normalized and then assigned on edges that connect nodes.

Table 3
Feature vectors to discover correlation between nodes.

	f_{1_b1}	f_{1_b2}	f_{2_b1}	f_{2_b2}	f_{n_b1}	f_{n_bj}
V_{ni}	0.92	0.67	0.38	0.45	0.55	0.65
V_{nj}	0. 83	0.43	0.56	0.43	0.43	0.63

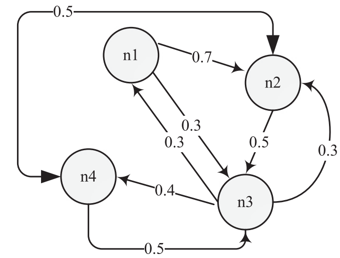


Fig. 3. Node/edge representation as a Markov Chain-based graph.

$$Sim_{n_i,\ n_j} = \frac{\left(cov\,(\sigma_{V_{n_i}}, \sigma_{V_{nj}})\right)}{\sigma_{V_{n_i}} \times \ \sigma_{V_{nj}}} = \frac{e\left[(V_{n_i} - \mu_{V_{n_i}})\ \ (V_{nj} - \mu_{V_{nj}})\right]}{\sigma_{V_{n_i}} \times \ \sigma_{V_{nj}}} \tag{1}$$

4.5.2. An inference procedure is performed to discover indirect paths between nodes

In this step the created similarity-based graphs are traversed based on the path length. The most feasible path is selected to find a numerical score that represents the most probable relationship between any two nodes.

Fig. 3 shows an example of a graph with four nodes. The values shown on the edges represent the transition probabilities from one node to another. The most probable relationship is identified based on the following steps:

1. Calculating the score l which is the sum of products of weights on all paths with length $|t|$ that connect nodes (n_i, n_j)
2. Depending on how many paths of length $|t|$ that exist between nodes (n_i, n_j), $l = N^{|t|}_{n_i \to n_j}$, there will be several values of l based on the number of reasoning steps ($|t|$), however, we selected the maximum value as the most probable link between nodes based on a previous work by Karabatis et al. (2009).

When a new node n_i is discovered, there is no need to regenerate the entire graph. The following steps are used to add the new discovered node.

1. *Find similarity* sim *between node* n_i *and all other nodes in the existing graph.*
2. *Using similarity values calculated in 1, select the k-nearest neighbors to* n_i *from the existing graph*
3. *Find* l_{avg}*(the average of* l *values between* k *nodes.*
4. *If all* k *selected nodes* $\in c_i$ *(i.e. all nodes belong to the same attack category)*

Calculate $l_{(n_i \to n_k)} = l_{avg} + (|1 - l_{avg}| \times Sim\,(n_i,\ n_k)$
Else.
Calculate$l_{(n_i \to n_k)} = l_{avg} - (|1 - l_{avg}| \times Sim\,(n_i,\ n_k)$

Fig. 4 shows an example of augmenting a graph with a new node n_i, the red nodes are the k-nearest neighbors to n_i, therefore n_i will be connected to these nodes, assuming that

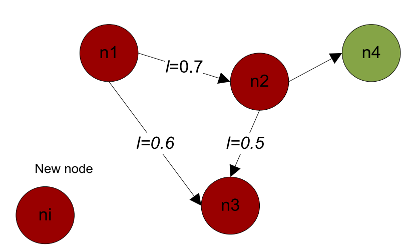


Fig. 4. Graph with l between nodes n_1, n_2, n_3.

$Sim\ (n_i,\ n_1) = 0.9$

$Sim\ (n_i,\ n_2) = 0.8$

$Sim\ (n_i,\ n_3) = 0.7$

Since all nodes in red belong to the same category and using the procedure above, the value of l between the new node n_i and n_1, n_2, n_3 will be

$$l_{(n_i \to n_1)} = l_{avg} + (|1 - l_{avg}| \times Sim(n_i, n_1) = 0.6 + (1 - 0.6) \times 0.9$$

$$l_{(n_i \to n_2)} = l_{avg} + (|1 - l_{avg}| \times Sim(n_i, n_2) = 0.6 + (1 - 0.6) \times 0.8$$

$$l_{(n_i \to n_3)} = l_{avg} + (|1 - l_{avg}| \times Sim(n_i, n_3) = 0.6 + (1 - 0.6) \times 0.7$$

Note: l_{avg} =(0.7+0.6+0.5)/3=0.6.

4.6. Flow classification

This module is responsible for processing OpenFlow-based flows and generating predictions that identify the type of the flow under analysis at run time. The created graph is used to model the relationships between nodes. Since each node represents a specific type of attack or a benign activity, the signature of each node is defined as a feature vector which contains the same features of incoming OpenFlow-based flows that are used for testing. This vector contains the highly weighted features for that node. The k-Nearest Neighbor and our graph model are used to classify incoming Open flow-based flows as follows:

4.6.1. Using K-NN for flow classification

The k-NN classifier, uses a similarity function to discover the k-similar nodes with the flow under analysis, given that each node is represented as feature vector as shown in Fig. 5. Each feature vector represents a profile that contains the features with the highest weights for that node. Conditional Entropy is used for feature weighting in order to create such profiles.

The flow is classified based on the type which is a majority in its neighbors. When the value of $k = 1$, the flow is classified based on the class of its single neighbor. Each flow is classified as a specific type of DoS attacks or a benign activity.

Using the graph model for flow classification: Algorithm 3 shows the steps required to classify incoming flows using our graph model.

The algorithm needs three inputs, the set of flows to be processed FL, the number of nearest neighbors k and the values of l. The features of the flow under analysis are used to find the k-nearest neighbors in the graph. Each node in the graph is represented as a feature vector which is used to discover the similarity with the features of incoming flows (line 3). Compared to k-NN classifier, once the first nearest neighbor n_i to an incoming flow fl_i is discovered using feature similarity, the k-nearest neighbors to n_i are queried from the graph using the values of l (line 12 and Fig. 6). The inquired nearest neighbors are stored in R which contains the suspicious and benign types of nodes. This set contains the predictions made to that particular flow. In practice, this set should only contain either suspicious or benign nodes, therefore, the filtering function *classifyF* is used. The function *classifyF* takes the set R as an input and produces the final prediction as an output. For each flow the number of suspicious and benign predictions in R is counted using two separate counts. The majority rule is used to classify that flow. If the count of suspicious predictions is greater, the flow is classified as suspicious, or else, the flow is classified as a benign activity. If the counts are equal, then the best way to classify the flow is to remove the least similar node from R until the tie is broken.

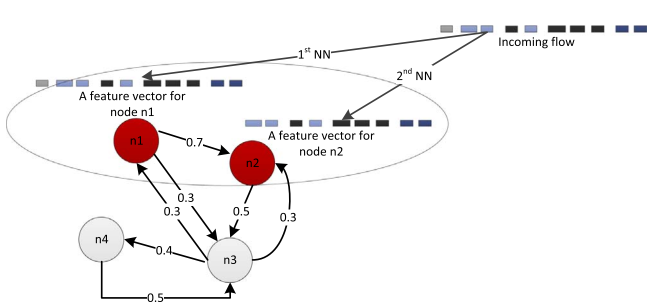


Fig. 5. Discovering the NNs using simple K-NN.

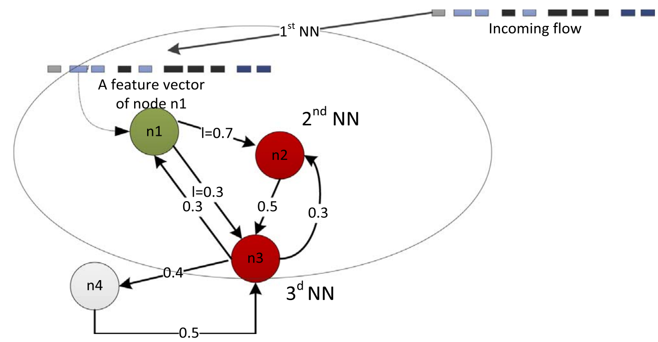


Fig. 6. Attack prediction and retrieval of the k-NNs by querying the created graph.

Algorithm 3. Flow classification using the graph model.

Inputs: k , A set of flows $FL=\{fl_1, \ldots, fl_n\}$, $l(n_i, n_j)\ \forall\ n_i, n_j \in N$
Procedure Flow classification(FL, k,l)
1. *For* i=1 *to* n *do*
2. $R = \{\varnothing\}$, *suspiciouscount*=0, *benigncount* = 0
3. *Discover the most similar node* n_i *to* fl_i *using feature similarity*
4. $R = \{n_i\}$
5. *If* n_i *is suspicious then*
6. *suspiciouscount* = *suspiciouscount* + 1
7. *Else*
8. *benigncount* = *benigncount* + 1
9. *End if*
10. *For* j=1 *to* k *do*
11. *Retrieve top* $kl(n_i, n_j)$
12. $R=\{R\} \cup n_j$
13. *If* n_j *is a DoS alert then*
14. *suspiciouscount* = *suspiciouscount* + 1
15. *Else*
16. *benigncount* = *benigncount* + 1
17. *End if*
18. *End for*
19. *classifyF(R,*fl_i*)*
20. *End for*
21. *End procedure*

Function *classifyF(R,* fl_i*)*
1. **Input:** A set R with suspicious and/or benign nodes, flow fl_i
2. **Return:** the type for each flow fl_i
3. *Check benigncount and suspiciouscount in R*
4. *If benigncount>suspiciouscount then*
5. *Return benign flow*
6. *Else if benigncount* < *suspiciouscount*
7. *return DoS alert*
8. *Else break a tie by removing the K-NN from R*
9. $R = (R/k)$
10. *While the tie is not broken*

11. *classifyF (R/k)*
12. *end loop*
End function

4.7. Graph model representation

The results of the similarity calculation are represented using the structure shown in Table 4. We used the power method on the similarity matrix to perform the reasoning step and calculate l with a time complexity of $O(N^3)$. The size of the graph has minor effect on the performance of the entire system. First, the representation used to model the created graph is the relational as shown in Table 5 where each node is connected to at most $N-1$ nodes. At runtime the relationships between nodes are scanned, this requires $O(N)$ where N is the total number of nodes. We used a relational approach to model the relationships instead of other graph modeling techniques. Therefore, the expansion step does not significantly affect the computational complexity and the time required to make predictions.

Table 4
Similarity score.

N1	N2	Similarity score
1	6	0.9
2	5	0.2
3	4	0.2
4	5	0.2
6	2	0.9
6	3	0.2

Table 5
Representation of our graph model.

N1	N2	l
1	2	0.8
1	3	0.1
1	5	0.1
1	6	*0.9*
2	5	0.2
3	4	0.2
4	5	0.2
6	2	0.9
6	3	0.2
…	…	…

5. Experiments and analysis

5.1. Network topology

Fig. 7 shows the network that is created to run our experiments. We used three hosts that run Linux OS which are also connected to an Open Virtual Switch. The latter works as a software which imitates the role of a regular switch. The communication between the hosts occurs only via the Virtual Switch. GENI experimentation environment is used to host the created network (Berman et al., 2014). We used a VM with a public IP as a controller, another VM works as a virtual switch. The three other VMs represent the attacker and two other hosts. Since the controller has a public IP it can communicate with a virtual switch over the internet.

There are many controllers that work in SDNs including Floodlight, Trema, and POX. We used POX which is an Open Source controller that uses Python as a programming language for network prototyping, management, and monitoring. Before running the data collection and experimentation, the virtual switch required several configuration

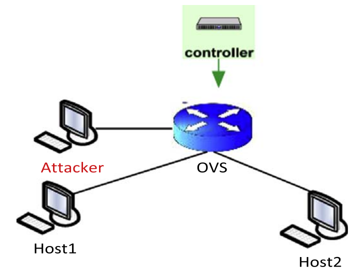


Fig. 7. Network topology used to generate suspicious and benign traffic.

steps:

1. Configuring the connection between the switch and the controller: Once the switch is pointed to the controller, no packet forwarding occurs by the switch unless otherwise allowed by the controller. The packet forwarding behavior is determined by rules that are inserted in the packet forwarding table which is an initially empty table.
2. The Virtual switch's interfaces work as an Ethernet bridge with interfaces as ports. The virtual switch can run as a L2-switch with no IP for ports. This is not a desired behavior when the controller is down. Due to DoS attacks, there will be no packet forwarding at all. As such, the switch should be converted into L-2 switch when the controller is not running.

To run our experiments several DoS attacks are initiated using custom scripts that are created using hping3 (Sanfilippo) tool which provides several commands to implement TCP SYN floods. The OpenFlow traces are generated by monitoring the messages exchanged between the controller and the OpenFlow switch. Monitoring those connections is important to distinguish between suspicious and benign traces. We used Wireshark connection monitoring system for capturing OpenFlow traffic between the controller and OVS hosts. Capturing open-flow traffic is done by sshing the controller machine. We also simulate the generation of the normal traffic using iPerf tool (iPerf3, 2014). The attacker host is used to generate TCP, ICMP, and UDP DoS attempts on hosts 1 and 2. To generate ICMP flooding attacks, the victim machine sends ICMP echo requests with no delay in between. As such, there is no need to wait for replies when the flood option of ping is used. The attack is considered successful if the CPU cycles of the target system are consumed slowly in a manner that a regular user will not notice. The attacks are initiated by generating a traffic with a rate of eight thousand packets per second. The packets sent do not match any rule. Consequently, the memory buffer of network devices is overloaded with large number of packet-in message to the controller. As such, the bandwidth of the data-control plane is occupied and the controller becomes less responsive due to resource consumption. A suspicious TCP/SYN DoS attempts are also generated. In such attacks, SYN requests are sent to the victim. The controller resources are consumed to the point that it does not respond to legitimate connection attempts. Similarly, suspicious UDP-based traffic is generated to create UDP flooding attack scenarios. UDP traffic is sent to random ports on the victims. We initiated 100 DoS attacks by changing various traffic parameters including the delay between packets and the duration of flows. We have also generated benign flows to use them in our experiments. Both benign and suspicious flows are used to test the effectiveness of the proposed approach. Examples on the initiated attacks are shown in Fig. 8a–c.

5.2. Implementation and datasets

Using a Dual Core processer which runs a 64 windows with 8 GB RAM, we evaluated the effectiveness of the proposed approach. Our graph model is implemented in Matlab. The resulting relationships and

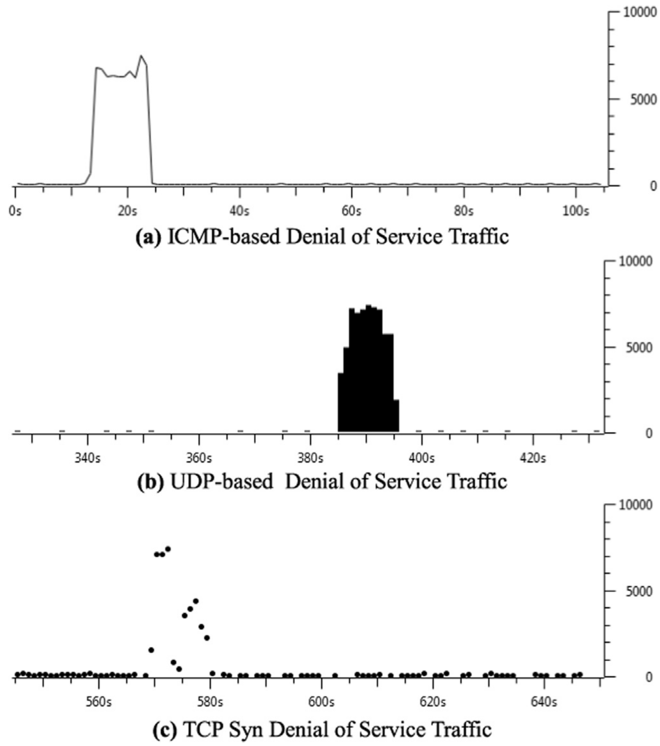


Fig. 8. Denial of Service attack scenarios on SDNs. (a) ICMP-based Denial of Service Traffic. (b) UDP-based Denial of Service Traffic. (c) TCP Syn Denial of Service Traffic.

the values of l are stored in an Oracle database. We implemented the k-NN algorithm using PL-SQL. In addition, we created flows using the approach implemented in (Ding and Aleroud, Karabatis). We used two datasets in our experiments:

1. **A CAIDA dataset that contains DoS attacks:** This dataset contains anonymized network traces (The CAIDA "DDoS Attack, 2007). It consists of attacks that are initiated to consume computing resources on the targeted servers to block all future connections on the victims. This dataset does not contain labels for each packet, therefore, an analysis is performed using Snort IDS on this dataset to identify the type of each packet. The dataset contains DoS attacks, therefore, the types of alerts raised by Snort are compatible with the suspicious traces that are generated to test our approach.
2. **A dataset that contains suspicious network connections:** This dataset consists of network connections with packets that have features such as source and destination IP, source and destination ports, protocol, and packet type (Syed Ali Khayam). This dataset contains labels per connection. Therefore, only flow aggregation is needed to create flows.

Pre-processing steps are performed on the flows generated from each dataset. By varying traffic parameters, we selected different types of attacks and benign activities for our test scenarios. The data used for testing consists of suspicious and benign OpenFlow-based flows. A summary of the flows used in our experiments, the number and types of nodes are shown in Table 6.

Three types of experiments are conducted.

- First, we measured the effectiveness of the proposed approach in terms of Precision (P), Recall (R), and F-score (F) when varying the percentage of training regular flows with respect to the testing OpenFlow-based flows. We used six samples of regular training flows. With each sample we selected a specific percentage of the testing OpenFlow-based flows, as shown in Fig. 9. The sample S_1 contains 90% Non-OpenFlow traces for training compared to 10% OpenFlow traces for testing. The sample S_6 contains 10% Non-OpenFlow flows for training compared to 90% OpenFlow flows for testing. The objective of such a variation is to examine if increasing the percentage of Non-OpenFlow training flows has a significant effect on correctly recognizing the types of testing OpenFlow-based flows. We used k-NN as a baseline classifier to run this experiment.
- Second, we compared between the effectiveness of $k-$ NN and our graph model based on the correct label of the suspicious and benign testing flows.
- Third, we measured the elapsed time needed to process the testing flows using each approach.

Table 6
Data used for training and testing.

Activity type	Dataset 1 Training	Testing	# of n	Dataset 2 Training	Testing	# of *n*
TCP/SYN flood	6214	3216	1	8316	1214	1
UDP Flood	–	2512	–	5322		4
ICMP flood	6230	3590	2	7255	2612	2
Total	**12,444**	**9318**	–	**20,893**	**3826**	–
TCP/SYN Benign traffic	2712	1100	1	4512	3650	1
UDP benign traffic	–	–	–	6622		4
ICMP benign traffic	5112	4100	2	7213	2212	2
Total	**7824**	**5200**	–	**18,347**	**5862**	–
Suspicious and benign	**20,268**	**14,518**	–	**39,240**	**9688**	–
% of suspicious flows	**0.61**	**0.64**	–	**0.46**	**0.60**	–
% of benign flows	**0.38**	**0.35**	–	**0.53**	**0.39**	–

Note: # of n is the number of nodes in the graph

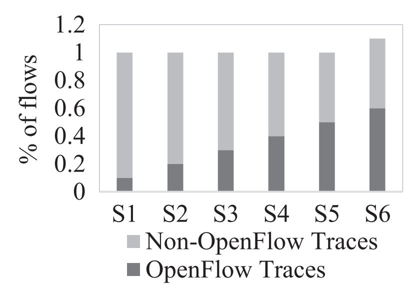
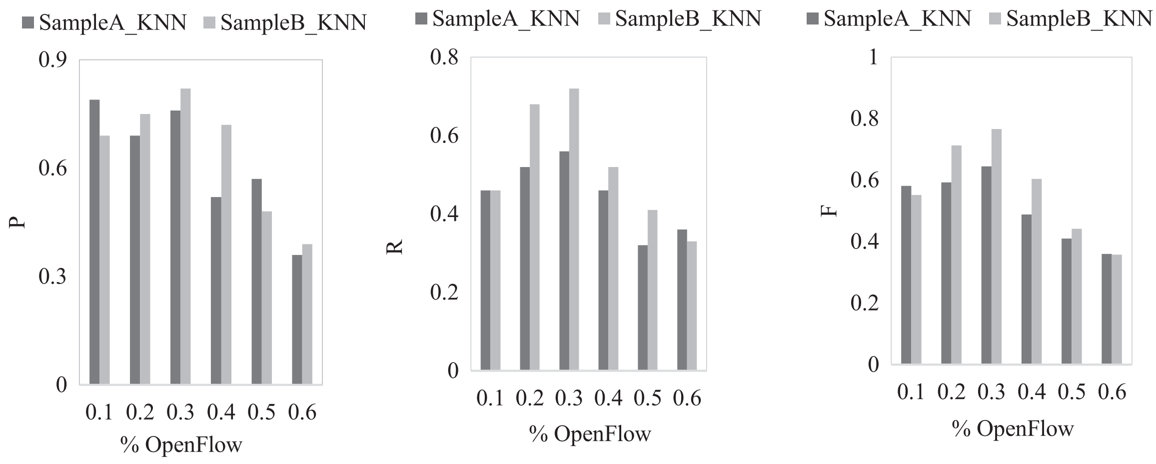


Fig. 9. % of OpenFlow testing traces to non-OpenFlow training traces.

Both R and P are calculated based on the values of True Positive (TP), False Positive (FP), and False Negative (FN) rates which are calculated using the testing flows. The value of TP rate represents the ratio between the suspicious flows that are predicted as suspicious and all known suspicious flows. The FP rate represents the ratio between the number of benign flows that are predicted as suspicious and the total number of known benign flows. A FN rate represents the ratio between the number of suspicious flows that are predicted as benign and the total number of known suspicious flows. Formulas 2, 3 and 4 are used to calculate the values of P, R and F-score.

$$P=\frac{TP}{TP+FP} \tag{2}$$

$$R=\frac{TP}{TP+FN} \tag{3}$$



(a) Average Precision (b) Average Recall (c) Average F

Fig. 10. Average Precision, Recall, and *F*-score when varying the percentage of OpenFlow to non-OpenFlow flows. (a) Average Precision. (b) Average Recall. (c) Average F.

$$F-score=\frac{(1+\beta^2)\times P\times R}{\beta^2\times(P+R)}\beta^2=1 \quad (4)$$

The analysis of the collected OpenFlow-based flows emphasizes on: (1) discovering similarity between such flows and the non-OpenFlow flows, and (2) classifying them using an appropriate classification technique. We hypothesized that *sampling OpenFlow-based flows and classifying them using an appropriate intrusion detection mechanism, can be effectively used to discover threats on SDNs.*

5.3. Changing the percentages of OpenFlow to non-OpenFlow flows

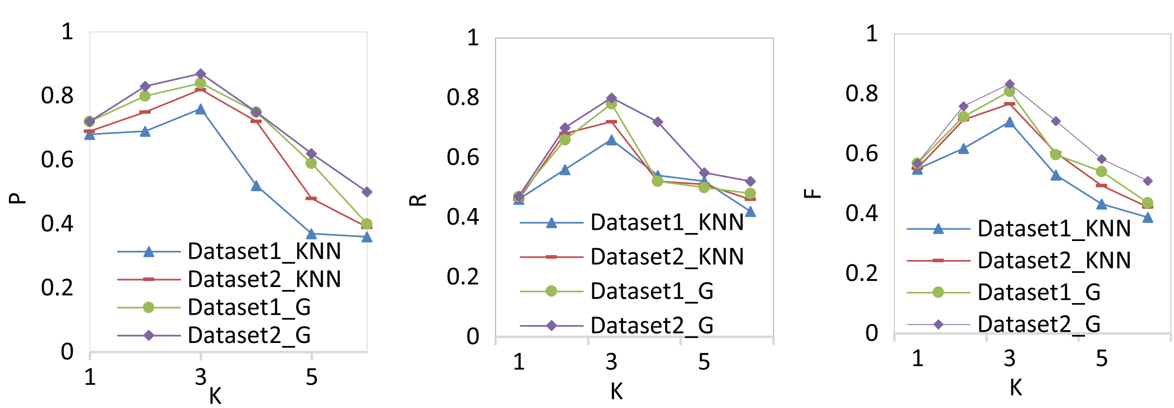
Fig. 10a–c shows the results of our experiments when changing the percentage of OpenFlow to non-OpenFlow flows. The results show that increasing the percentage of OpenFlow-based flows, which represent the testing sample while decreasing the percentage of non-OpenFlow-based flows, which represent the training sample, lowers the attack detection rate in terms of P, R and F-score. This indicates that there is a need for a sufficient number of attack signatures (i.e., the non-OpenFlow training sample) in order to detect attacks on SDNs. In other words, when using a sufficient number of attack signatures, our approach recognizes different types of attacks on SDNs, which explains the higher attack detection rate when more non-OpenFlow flows are used for training.

5.4. The effectiveness of K-NN vs. graph model

The second experiment measures the difference between the effectiveness of the $k-$NN classifier and our graph model. As shown in Fig. 11a–c, the values of P, R and F increase before they decrease when the value of k is greater than 3. In addition, there is a significant difference between the effectiveness of k-NN and the graph model. The dynamic nature of SDNs makes it challenging to identify attacks using traditional classifiers. The graph model provides an advanced methodology to retrieve related nodes which leads to higher detection rate when handling attacks on SDNs.

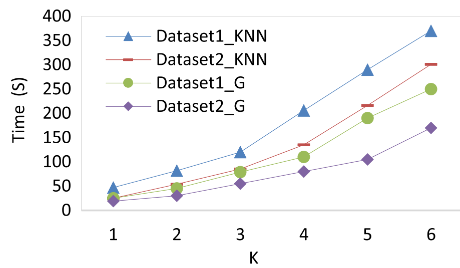
5.5. Scalability with higher value of k

The results of the experiments on testing (elapsed) time are affected by changing the value of k and the amount of testing data. Suppose that we have n flows each of a dimensionality d. $O(d)$ is the complexity to compute distance to one flow, $O(nd)$ is the complexity to find one nearest neighbor. The basic k-NN has the complexity of $O(ndk)$ to find k closest examples. This is prohibitively expensive for large number of flows. For the graph model, the complexity is $O(n'kd)$, however, $|n'| < |n|$ where $|n'|$ is the total number of nodes in the graph. In fact, when $k > 1$, only the first nearest neighbor needs to be calculated; other $k-1$ nearest neighbors are queried from the graph. Therefore, using the graph model is more scalable than k-NN. We compared the effectiveness of both techniques by measuring the testing time needed to classify flows as shown in Fig. 12a. The testing time in seconds varies depending on the value of k. Another observation is the less testing time when the graph model is used. Our graph based model, which includes an inference mechanism, limits the number of unnecessary comparisons and narrows down the number of nodes that need to be

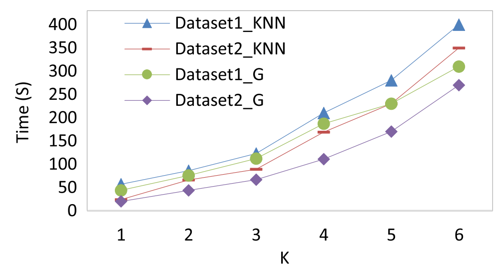


(a) Average precision (b) Average Recall (c) Average F-score

Fig. 11. Results of K-NN and our graph-based technique. (a) Average precision. (b) Average Recall. (c) Average *F*-score.



(a) Testing (elapsed) time using three nodes to generate testing flows



(b) Testing (elapsed) time using five nodes to generate testing flows

Fig. 12. Testing (elapsed) time using two different topologies to generate flows. (a) Testing (elapsed) time using three nodes to generate testing flows. (b) Testing (elapsed) time using five nodes to generate testing flows.

scanned which eventually affects the total time needed to classify incoming flows.

5.6. Scalability with larger topology

For this part of the experiment we increased the number of hosts to five (two attackers and three victims). We created two larger testing datasets. The first one consists of 24874 suspicious and benign flows while the second one consists of 17954 suspicious and benign. An interesting observation is the sublinear nature of the curves in Fig. 12b when we concurrently execute two instances of the attack prediction procedure.

5.7. Adding new nodes to graph

The last part of this experiment measures the effectiveness of our graph model when new nodes are added using the technique discussed in Section 4. We added one to six new types of nodes to the created graph, then the average values of P, R, and F-score are re-calculated as shown in Fig. 13. The values of P, R, and F-score are not significantly affected by adding those nodes. We noticed a minor decline in the values of P and a minor increase in the values of R. Adding new nodes results in increasing the number of retrieved nodes when querying the graph, thus, there is a greater likelihood to include false positives in the retrieved results, which eventually lowers the values of P.

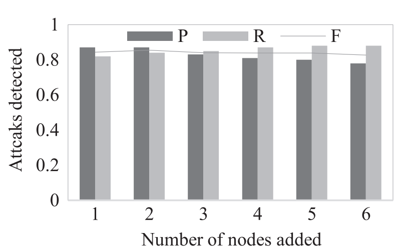


Fig. 13. Values of P, R, and F-Score when adding new graph nodes.

6. Comparison with other approaches

We compare our approach with the existing detection mechanisms that are created to identify DoS attacks on SDNs. It is quite challenging to find existing techniques that work on a standard intrusion detection dataset for SDNs. Therefore, we regenerated the results of the existing techniques using our datasets. We compared our approach with three existing techniques. The first technique was proposed by Braga et al. (2010). The second uses SVM and PCA to create a classifier that identifies flooding attacks on SDNs (da Silva et al., 2015). The third technique is also a SVM-based classification technique (Phan et al., 2016).

As Table 7 shows, our improved graph-based approach performed the best with P value between 0.80 and 0.84 using the first dataset and 0.82–0.87 using the second dataset. Similarly, the results reported in this table show that our approach achieves the best R values which range between 0.79 using the first data set and 0.80 using the second one. The F values follow the same pattern and they range between 0.81 using the first dataset and 0.83 when using the second dataset.

Table 7
A comparison with existing techniques.

Technique	***Dataset 1***			***Dataset 2***		
	P	R	F	P	R	F
K-NN	0.68	0.46	0.55	0.69	0.46	0.55
	0.69	0.56	0.62	0.75	0.68	0.71
	0.76	0.66	0.71	**0.82**	0.72	**0.77**
Graph-based	0.72	0.47	0.57	0.72	0.47	0.57
	0.80	0.66	0.72	**0.83**	0.70	0.76
	0.84	**0.78**	**0.81**	**0.87**	**0.80**	**0.83**
SOM (Braga et al., 2010)	0.71	0.66	0.68	0.75	0.74	0.74
SVM & PCA (da Silva et al., 2015)	0.78	0.71	0.74	0.80	0.74	0.77
SVM (Phan et al., 2016)	0.77	0.66	0.71	0.78	0.76	0.77

7. Research implications

One of the features that SDN has, which may support SDN tasks, is the controller global view and knowledge of the network. The design of SDNs emerges because there is a need to handle traffic in computer networks in a manner that is dynamic and programmable (Kreutz et al., 2015). This is done by embedding source code that controls the process of packet forwarding across the network. SDNs separate the data layer from the control layer. The earlier deals with the data exchanged during network connections. In addition it includes the switches that forward packets. The latter provides the logic and procedures used to decide where the packets are sent. With such an architecture, the network becomes more flexible in terms of traffic control and network management. In addition, it is easier to manage, update and change decisions on packet forwarding (Koponen et al., 2011; Li and Liao, 2013), However, this architecture introduce several security challenges.

Our approach handles such a challenges by designing and implementing one or more IDS appliances connected to the controller which can receive overall global traffic and information related to the network. This may facilitate some tasks that were much harder and complex to accomplish in IDS given the traditional networking architecture. This architecture can contribute also to solving earlier problems or challenges related to packets or traffic collection. This was related to the ability to collect all network traffic and monitor the network without impacting its performance.

The experiments conducted using our approach provide an evidence that sampled OpenFlow traffic can be efficiently tested using the proposed technique. Under some circumstances, traffic samples are needed for analysis instead of the entire traffic, however, there is a need to select the sampling method based on the security applications and the characteristics of data under analysis. For instance if the data is highly sensitive, full packets can be analyzed instead of sampling certain packets. The evolution of OpenFlow protocol should take into consideration all quality factors and not only security. This is since such addition of information to be exchanged with the controller is expected to cause a significant overhead for an already exhausted controller.

In addition, the fact the SDN controller can aggregate data from different switches in one location can also be an important characteristic since our approach relies on information aggregation to identify attacks. The central NIDS module periodically collects information about traffic and policies. It can be also triggered by certain traffic changes or events. The traffic itself needs to be classified or categorized into different classes. Each class can be subjected to different types of NIDS analysis. Our engine can be used for an initial analysis of traffic to specify the type of analysis to subject the traffic to. This can be an evolutionary process that learns from past experience or traffic and improve accuracy in future.

Information about the entity relations is very important to realize entity situation in a particular location, at a particular point of time. Each entity (e.g. network host, attack), in the network environment, has one or more static or dynamic relations with other entities. In this work we consider the relationship between attacks to detect them at real time on SDN. The relation context information is modeled using a graph model with nodes representing attacks, and edges representing the strength of relationships between them. The closer the relationship between attacks n_i and n_j, the higher the possibility that such attacks would co-occur in a particular context, and consequently, detecting attack n_i might help in proactively, avoiding attack n_j. Strongly related attacks, in terms of similarity and source of the attacks, will be near each other in the graph. Defining such relations using our graph model can be used in predicting the future attacks based on their relevancy to ones predicted by IDS.

Finally, there are several design goals that are fulfilled in our approach:

- Openness and Extensibility: While there are some open source IDS such as Snort, however, in most cases, those are not easy to extend. SDN open vendor programmable architecture allows us to develop algorithms that interact with the flows generated in an Openflow-based networks.
- Flexibility: The developed approach is flexible to use, configure, and interact with. Users, network administrators and programmers can still access network functionalities. Our system is flexible to allow adding new rules, datasets of known vulnerabilities, malwares, etc.
- Dynamic: This is one of the most ambitious goals of developing IDSs for SDNs. Our system is able to interact with the network with the least human intervention. Our system, adjusts its own inference rules in response to network attacks. For example, when a new type of threat is identified, some rules will be adjusted to accommodate this new threat (e.g. adding a new node to the created graph.

8. Concluding remarks and future work

SDNs bring new advantages over the traditional network infrastructure as they separate data from control plane. This separation introduces several security challenges and opens the door to different types of security threats, including DoS attacks. This paper presents several related studies which show that it is feasible to initiate such attacks on SDNs. We proposed a novel approach to mitigate DoS attacks on SDNs. Our approach is driven by the Graph Theory and it focuses on the role of relation context to identify such attacks on SDNs. This work helps not only to understand the role of contextual relationships for identifying attacks on SDNs, but also highlights the role similarity of features of incoming OpenFlow traffic with the features of traditional attacks in identifying attacks on SDNs. This work will be extended to consider other types of attacks on SDNs. Another direction is to create our prediction models as an OpenFlow application where some functionalities may be embedded in the controller. Finally, there is a need to consider zero-day attacks that may target SDNs.

Acknowledgement

We wish to acknowledge the assistance of CAIDA and GENI for their support in providing experiments' resources and datasets.

References

AlEroud, A., Karabatis, G., 2012. A contextual anomaly detection approach to discover zero-day attacks. In: International Conference on Cyber Security (CyberSecurity), pp. 40–45.

Alsmadi, I., Xu, D., 2015. Security of software defined networks: a survey. Comput. Secur. 53, 79–108.

Ballard, J.R., Rae, I., Akella, A., 2008. Extensible and Scalable Network Monitoring Using OpenSAFEINM/WREN.

Beauquier, J., Hu, Y., 2007. Intrusion detection based on distance combination. World Acacemy Sci., WAS 1 (7), 72–80.

Benton, K., Camp, L.J., Small, C., 2013. Openflow vulnerability assessment. In: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined, 151–152.

Berman, M., Chase, J.S., Landweber, L., Nakao, A., Ott, M., Raychaudhuri, D., et al., 2014. GENI: a federated testbed for innovative network experimentsComput. Netw. 61, 5–23.

Braga, R., Mota, E., Passito, A., 2010. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In: Proceedings of the 35th Conference on Local Computer Networks (LCN), IEEE, pp. 408–415.

Braga, R., Mota, E., Passito, A., 2010. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In: Proceedings of the 35th Conference on Local Computer Networks, IEEE, pp. 408–415.

Choi, Y., 2010. Implementation of Content-oriented Networking Architecture (CONA): A Focus on DDoS Countermeasure. In: Proceedings of the 1st European NetFPGA Developers Workshop, Cambridge, UK.

Chowdhury, S.R., Bari, M.F., Ahmed, R., Boutaba, R., 2014. Payless: a low cost network monitoring framework for software defined networks. In: IEEE Network Operations and Management Symposium (NOMS), pp. 1–9.

Chung, C.-J., Cui, J., Khatkar, P., Huang, D., 2013. Non-intrusive process-based monitoring system to mitigate and prevent VM vulnerability explorations. In: Proceedings of the 9th International Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), pp. 21–30.

Chung, C.-J., Khatkar, P., Xing, T., Lee, J., Huang, D., 2013. NICE: network intrusion detection and countermeasure selection in virtual network systemsIEEE Trans. Dependable and Secure Computing 10, 198–211.

Cui, Y., Yan, L., Li, S., Xing, H., Pan, W., Zhu, J., et al., 2016. SD-Anti-DDoS: fast and efficient DDoS defense in software-defined networks. J. Netw. Comput. Appl. 68, 65–79.

Dabbagh, M., Hamdaoui, B., Guizani, M., Rayes, A., 2015. Software-defined networking security: pros and cons. IEEE Commun. Mag. 53 (6), 73–79.

Delgado, J.S.S., Peñuela, D.J.M., Medina, L.V.M., Rodríguez, S.J.R., 2014. Automatic network reconfiguration because of security events. In: IEEE Colombian Conference on Communications and Computing (COLCOM), pp. 1–6.

Ding, A.Y., Crowcroft, J., Tarkoma, S., Flinck, H., 2014. Software defined networking for security enhancement in wireless mobile networks. Comput. Netw. 66 (1), 94–101.

Ding, T., Aleroud, A., Karabatis, G.. Contextually Revealing Attacks over IP flows(CRAFT). Available: ⟨http://userpages.umbc.edu/~cq13453/Craft/⟩

Ding, T., AlEroud, A., Karabatis, G., 2015. Multi-granular aggregation of network flows for security analysis. In: IEEE International Conference on Intelligence and Security Informatics (ISI), pp. 173–175.

Giotis, K., Argyropoulos, C., Androulidakis, G., Kalogeras, D., Maglaris, V., 2014. Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. Comput. Netw. 62, 122–136.

Ha, T., Kim, S., An, N., Narantuya, J., Jeong, C., Kim, J., et al., 2016. Suspicious traffic sampling for intrusion detection in software-defined networks. Comput. Netw. 101 (2).

Hall, J., Ramsey, B., Rice, M., Lacey, T., 2016. Z-wave network reconnaissance and transceiver fingerprinting using software-defined radios. In: International Conference on Cyber Warfare and Security.

Hassanzadeh, A., Sadeghian, B., 2008. Intrusion detection with data correlation relation graph. In: Proceedings of the Third International Conference on Availability, Reliability and Security (ARES '08), Barcelona, Spain, pp. 982–989.

Hogg, S., 2013. Using SDN to Create a Packet Monitoring System: Packet-level Monitoring Use Case with Cisco XNC and Monitor Manager, (Available)⟨http://www.networkworld.com/article/2226003/cisco-subnet/using-sdn-to-create-a-packet-monitoring-system.html⟩.

Hong, S., Xu, L., Wang, H., Gu, G., 2015. Poisoning Network Visibility in Software-Defined Networks: NEW Attacks and Countermeasures. NDSS.

Hu, H., Han, W., Ahn, G.-J., Zhao, Z., 2014. FLOWGUARD: building robust firewalls for software-defined networks. In: Proceedings of the third workshop on Hot topics in software defined networking, 97–102.

iPerf3, 2014. ⟨https://iperf.fr/⟩.

Karabatis, G., Chen, Z., Janeja, V., Lobo, T., Advani, M., Lindvall, M., et al., 2009. Using semantic networks and context in search for relevant software engineering artifacts. J. Data Semant., LNCS 5880 14, 74–104.

Kerner, S.M., 2012. OpenFlow Can Provide Security, Too., (Available)⟨http://www.enterprisenetworkingplanet.com/datacenter/openflow-can-provide-security-too.html⟩.

Kim, H., Feamster, N., 2013. Improving network management with software defined networking. IEEE Commun. Mag. 51 (2), 114–119.

Kloti, R., Kotronis, V., Smith, P., 2013. Openflow: a security analysis. In: 21st IEEE International Conference on Network Protocols (ICNP), 1–6.

Koponen, T., Shenker, S., Balakrishnan, H., Feamster, N., Ganichev, I., Ghodsi, A., et al., 2011. Architecting for innovation. ACM SIGCOMM Comput. Commun. Rev. 41 (3), 24–36.

Kreutz, D., Ramos, F., Verissimo, P., 2013. Towards secure and dependable software-defined networks. Proceedings of the second ACM SIGCOMM Workshop on Hot topics in software defined networking, 55–60.

Kreutz, D., Ramos, F.M., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., Uhlig, S., 2015. Software-defined networking: a comprehensive survey. Proc. IEEE 103 (1), 14–76.

Li, C.-S., Liao, W., 2013. Software defined networks. IEEE Commun. Mag. 51 (2), 113.

Liu, J., Li, Y., Wang, H., Jin, D., Su, L., Zeng, L., et al., 2016. Leveraging software-defined networking for security policy enforcement. Inform. Sci. 327, 288–299.

McKeown, N., 2009. Software-defined networking. INFOCOM Keynote Talk 17 (2), 30–32.

Mehdi, S.A., Khalid, J., Khayam, S.A., 2011. Revisiting traffic anomaly detection using software defined networking. Recent Adv. Intrusion Detect., 161–180.

Nunes, B.A., Mendonca, M., Nguyen, X.-N., Obraczka, K., Turletti, T., 2014. A survey of software-defined networking: past, present, and future of programmable networks. IEEE Commun. Surv. Tutor. 16 (3), 1617–1634.

Park, Y., Chang, S.-Y., Krishnamurthy, L.M., 2016. Watermarking for detecting freeloader misbehavior in software-defined networks In: 2016 International Conference on Computing, Networking and Communications (ICNC), pp. 1–6.

Phan, T.V., Van Toan, T., Van Tuyen, D., Huong, T.T., Thanh, N.H., 2016. OpenFlowSIA: An optimized protection scheme for software-defined networks from flooding attacks. In: Proceedings of the Sixth International Conference on Communications and Electronics, IEEE, pp. 13–18.

Qishi, W., Ferebee, D., Yunyue, L., Dasgupta, D., 2009. An integrated cyber security monitoring system using correlation-based techniques. In: IEEE International Conference on System of Systems Engineering (SoSE'09), Albuquerque, NM, pp. 1–6.

Rothenberg, C.E., Nascimento, M.R., Salvador, M.R., Corrêa, C.N.A., Cunha de Lucena, S., Raszuk, R., 2012. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, 13–18.

Skowyra, R.S., Bahargam, S., Bestavros, A., 2013. Software-defined IDS for securing embedded mobile devices. In: High Performance Extreme Computing Conference (HPEC), IEEE, pp. 1–7.

Sanfilippo, S.. hping3. ⟨https://linux.die.net/man/8/hping3⟩.

Schehlmann, L., Baier, H., 2013. COFFEE: A Concept based on OpenFlow to Filter and Erase Events of botnet activity at high-speed nodesGI-Jahrestagung, 2225–2239.

Sezer, S., Scott-Hayward, S., Chouhan, P.-K., Fraser, B., Lake, D., Finnegan, J., et al., 2013. Are we ready for SDN? Implementation challenges for software-defined networks. IEEE Commun. Mag. 51 (7), 36–43.

Shin, S., Gu, G., 2013. Attacking software-defined networks: a first feasibility study. In: Proceedings of the second Workshop on Hot Topics in Software Defined Networking, 165–166.

Shin, S., Wang, H., Gu, G., 2015. SDN misuse/attack cases. Available: ⟨http://sdnsecurity.org/project_SDN-Security-Vulnerbility-attack-list.html⟩

Shin, S., Yegneswaran, V., Porras, P., Gu, G., , 2013. AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pp. 413–424.

Shin, S., Porras, P.A., Yegneswaran, V., Fong, M.W., Gu, G., Tyson, M., 2013. FRESCO: modular Composable Security Services for Software-Defined Networks. NDSS.

Shirali-Shahreza, S., Ganjali, Y., 2013. Empowering software defined network controller with packet-level information. In: 2013 IEEE International Conference on Communications Workshops (ICC), pp. 1335–1339.

Shirali-Shahreza, S., Ganjali, Y., 2013. Flexam: flexible sampling extension for monitoring and security applications in openflow. In: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, 167–168.

Shirali-Shahreza, S., Ganjali, Y., 2013. Efficient implementation of security applications in openflow controller with flexam. In: Proceedings of the 21st Annual Symposium on High-Performance Interconnects (HOTI), IEEE, pp. 49–54.

da Silva, A.S., Machado, C.C., Bisol, R.V., Granville, L.Z., Schaeffer-Filho, A., 2015. Identification and selection of flow features for accurate traffic classification in SDN. In: Proceedings of the 14th International Symposium on Network Computing and Applications (NCA), 2015 IEEE, pp. 134–141.

Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., Stiller, B., 2010. An overview of IP flow-based intrusion detection. IEEE Commun. Surv. Tutor. 12 (3), 343–356.

Syed Ali Khayam, F.M.. Network-Embedded Security Using In-Network Packet Marking. Available: ⟨http://ictrdf.org.pk/beta/administrator/components/com_tprojects/images/download/progress_reports/2c5acbfb5d6757cfe7f31c6137e7464a-13-Project-Completion-Report-Open-Source-Enterprise.pdf⟩

Tasch, M., Khondoker, R., Marx, R., Bayarou, K., 2014. Security analysis of security applications for software defined networks. In: Proceedings of the AINTEC 2014 on Asian Internet Engineering Conference, p. 23.

The CAIDA DDoS Attack2007 Dataset. Available: ⟨http://www.caida.org/data/passive/ddos-20070804_dataset.xml⟩

Tsugawa, M., Matsunaga, A., Fortes, J.A., 2014. Cloud Computing Security: What Changes with Software-Defined Networking?Secure Cloud Computing 1st ed.. Springer, 77–93.

Van Trung, P., Huong, T.T., Van Tuyen, D., Duc, D.M., Thanh, N.H., Marshall, A., 2015. A multi-criteria-based DDoS-attack prevention solution using software defined networking. In: International Conference on Advanced Technologies for Communications (ATC), 308–313.

Wang, B., Zheng, Y., Lou, W., Hou, Y.T., 2015. DDoS attack protection in the era of cloud computing and software-defined networking. Comput. Netw. 81 (1), 308–319.

Wang, H., Xu, L., Gu, G., 2014. Of-guard: A DoS Attack Prevention Extension in Software-defined NetworksThe Open Network Summit (ONS).

Xing, T., Huang, D., Xu, L., Chung, C.-J., Khatkar, P., 2013. Snortflow: A openflow-based intrusion prevention system in cloud environment. In: Research and Educational Experiment Workshop (GREE)s GENI, pp. 89–92.

Yan, Q., Yu, F., 2015. Distributed denial of service attacks in software-defined networking with cloud computing. IEEE Commun. Mag. 53 (4), 52–59.

Yoon, C., Park, T., Lee, S., Kang, H., Shin, S., Zhang, Z., 2015. Enabling security functions with SDN: a feasibility study. Comput. Netw. 85 (1), 19–35.

YuHunag, C., MinChi, T., YaoTing, C., YuChieh, C., YanRen, C., 2010. A novel design for future on-demand service and security. In: Proceedings of the 12th International Conference on Communication Technology, IEEE pp. 385–388.

Yuzawa, T., 2013. *OpenFlow 1.0 actual use-case: RTBH of DDoS Traffic While Keeping the Target*, (Available)⟨http://packetpushers.net/openflow-1-0-actual-use-case-rtbh-of-ddos-traffic-while-keeping-the-target-online⟩.

Zaalouk, A., Khondoker, R., Marx, R., Bayarou, K., 2014. OrchSec: an orchestrator-based architecture for enhancing network-security using Network Monitoring and SDN Control functions. In: IEEE Network Operations and Management Symposium (NOMS), pp. 1–9.

Zhang, W., Liu, G., Wood, T., Ramakrishnan, K., Hwang, J., 2016. Flexible and Dynamic Software Defined Control of an Application-and Flow-AwareData Plane. arXiv preprint arXiv:1606.02599

Zhang, Y., 2016. Method and system to dynamically collect statistics of traffic flows in a software-defined networking (SDN) system. ed: US Patent 20,160,050,132.

Ahmed AlEroud Assistant Professor of Computer Information Systems, Yarmouk University, Jordan. Visiting Associate Research Scientist, Department of Information Systems, University of Maryland, Baltimore County. Dr. Ahmed AlEroud is currently an Assistant Professor of Computer Information Systems, in the department of Computer Information Systems at Yarmouk University in Jordan. He has recently joined UMBC as a Visiting Associate Research Scientist in the Department of Information Systems. Dr. AlEroud has received his PhD in Information Systems from the University of Maryland, Baltimore County (UMBC). He also received a Master degree in Information Systems from UMBC. He received his B.Sc. degree in Software Engineering from Hashemite University, Jordan. He worked as a lecturer in computer information Systems department at Yarmouk University in Jordan during 2009–2011 academic years. His major research area is Cyber-Security. He mainly focuses on fusing contextual information to create graph-based techniques to detect cyber-attacks. His research appears in several cyber security and information systems conferences with high reputation, such as the IEEE/ASE International conference in Cyber-Security, the IEEE conference on software Security and Reliability, and the International Conference on Semantic Computing. He has several Journal Publications in the areas of cyber-Security, such as in the Journal of Information Science and the International Journal on Information and Computer Security. He has served as a committee member and a reviewer in some conferences in several areas such as the first International Conference on Anti-Cybercrime (ICACC-2015) and the entropy Journal. He has participated with his PhD Advisor in writing several proposals that were funded by Northrop Grumman Corporation and the State of Maryland.

Izzat Alsmadi Izzat Alsmadi is currently working as an Assistant Professor in the department of Computer Science at University of Texas A & M, San Antonio. He has his master and PhD in Software Engineering from North Dakota State University. He has more than 100 conference and journal publications. His research interests include: Software security, software engineering, software testing, social networks and software defined networking. Research Projects: Izzat Alsmadi is a member or CO-PI of several projects and pending publications including: Security and trust in social networks (Izzat Alsmadi, Dianxiang Xu and Jin-Hee Cho, University of New Haven, Boise State University, Army Research lab). Using SDN to enhance HPC (Izzat Alsmadi, Dianxiang Xu and Mishra Vinod, University of New Haven, Boise State University, Army Research lab). Using SDN to build dynamic security controls, Izzat Alsmadi, University of New Haven.