Computer Networks 62 (2014) 122–136

Contents lists available at ScienceDirect

Computer Networks

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

Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments

K. Giotis, C. Argyropoulos, G. Androulidakis [*], D. Kalogeras, V. Maglaris

Network Management & Optimal Design Laboratory (NETMODE), School of Electrical & Computer Engineering, National Technical University of Athens (NTUA), Greece

ARTICLE INFO

Article history:
Received 16 November 2012
Received in revised form 24 October 2013
Accepted 28 October 2013
Available online 4 December 2013

Keywords:
Software Defined Networking
SDN
OpenFlow
sFlow
Anomaly detection
Attack mitigation

ABSTRACT

Software Defined Networks (SDNs) based on the OpenFlow (OF) protocol export control-plane programmability of switched substrates. As a result, rich functionality in traffic management, load balancing, routing, firewall configuration, etc. that may pertain to specific flows they control, may be easily developed. In this paper we extend these functionalities with an efficient and scalable mechanism for performing anomaly detection and mitigation in SDN architectures. Flow statistics may reveal anomalies triggered by large scale malicious events (typically massive Distributed Denial of Service attacks) and subsequently assist networked resource owners/operators to raise mitigation policies against these threats. First, we demonstrate that OF statistics collection and processing overloads the centralized control plane, introducing scalability issues. Second, we propose a modular architecture for the separation of the data collection process from the SDN control plane with the employment of sFlow monitoring data. We then report experimental results that compare its performance against native OF approaches that use standard flow table statistics. Both alternatives are evaluated using an entropy-based method on high volume real network traffic data collected from a university campus network. The packet traces were fed to hardware and software OF devices in order to assess flow-based data-gathering and related anomaly detection options. We subsequently present experimental results that demonstrate the effectiveness of the proposed sFlow-based mechanism compared to the native OF approach, in terms of overhead imposed on usage of system resources. Finally, we conclude by demonstrating that once a network anomaly is detected and identified, the OF protocol can effectively mitigate it via flow table modifications.

1. Introduction

Data center operators and cloud service providers require flexibility and scalability in setting up programmable network environments. Traditional network architectures prove to be cumbersome, thus constricting innovations and hindering management and configuration procedures. Moreover, the software interfaces between control and data plane software elements inside network equipment still remain proprietary, leading to vendor lock-in phenomena.

Software-Defined Networks (SDNs) rise as a promising, alternative architecture where the control and data planes are decoupled. Network intelligence and state are logically centralized, while enforcement is distributed locally at network gear [1]. Among the first and most widespread

[*] Corresponding author. Tel.: +30 210 7721449.
E-mail addresses: coyiotis@netmode.ntua.gr (K. Giotis), cargious@netmode.ntua.gr (C. Argyropoulos), gandr@netmode.ntua.gr (G. Androulidakis), dkalo@netmode.ntua.gr (D. Kalogeras), maglaris@netmode.ntua.gr (V. Maglaris).

1389-1286/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.bjp.2013.10.014

mechanisms to support SDN architecture is the OpenFlow (OF) protocol [2]. OF enables exporting of control-plane programmability of flow-based switched substrates. As a result, OF can be effectively applied with a rich diversity in traffic management, load balancing, routing, firewall configuration, etc. that may pertain to specific flows they control. In this paper, we extend these functionalities with an efficient and scalable mechanism for performing anomaly detection and mitigation in OF SDN architectures. Our goal is to detect network attack patterns in real time, as well as, to enforce mitigation rules combining the OF technology and sFlow [3] data collection capabilities on edge switches.

Flow-based anomaly detection methods, as stated in [4–6], are well known for discovering all kinds of network anomaly patterns, whether they are malicious or benign. Furthermore, entropy-based methods have proven to be effective, particularly in determining the randomness of a data-set really, therefore pointing out potential network anomalies [7]. Flow-based monitoring can be employed to collect network flow statistics [8], while a method of normalized entropies may be used to perform detection of worm outbreaks. We apply this widely-used entropy-based anomaly detection method on SDN, by implementing and comparing native OF and sFlow based approaches for the traffic data collection process. The impact of each to the anomaly detection accuracy is evaluated through Receiver Operating Characteristic (ROC) curves. In addition, we investigate the applicability and performance of our proposed mechanism pertaining to other known anomaly detection algorithms reported in the literature. Finally, we exploit the flow instantiation and flow modification capabilities introduced by the OF Controller in order to effectively mitigate detected network anomalies.

Our main contributions are summarized in the following four points:

- Architectural design of a modular mechanism that permits anomaly detection and mitigation on SDN environments.
- Scalability improvements, in comparison with the already proposed native OF approaches, employing sFlow protocol for flow-based data gathering.
- Effective reduction of the required communication between switches and OF controllers, eliminating the potential control plane overloading, under anomaly conditions to the data plane
- Performance tests using real traffic traces in order to validate the scalability and the effectiveness of the proposed sFlow-based approach compared to the native OF mechanisms in terms of anomaly detection accuracy and overhead on system resources (CPU usage, flow table size).

The paper is structured as follows: In Section 2, we discuss relevant work describing the main differences with our proposed mechanism, while in Section 3 we describe the design principles of the introduced architecture. Section 4 presents the implementation details of the proposed anomaly detection and mitigation mechanism on SDNs with respect to data gathering, anomaly detection and mitigation process. Section 5 presents our performance evaluation experiments and Section 6 concludes the paper and discusses future work.

2. Related work

The emergence of a software programmable control plane idea led to the design of a platform, called Onix [9], currently operating in Google's internal Wide Area Network (WAN). Onix is a control plane implementation consisted of OF controllers that act as a distributed system. HyperFlow [10] is another distributed event-based control plane that is logically centralized, but physically distributed, which allows network operators to deploy any number of controllers in their networks. The SDN concepts brought to life high-level languages, such as Frenetic [11], adequate for programming distributed collections of network switches and frameworks (e.g. Odin [12]), that introduce programmability on wireless local area networks. It is worth pointing out that Frenetic presents some primitive conclusions regarding network monitoring in OF native environments.

Despite the extensive work in OF with respect to monitoring, routing, load-balancing, etc. the intrusion detection field has not yet attracted much attention. Early work on establishing a flow-based intrusion detection mechanism using OF was reported in [13]: OF was used to facilitate the handling of the forwarding plane of the network device and to gather flow statistics, without any analysis of the constraints introduced from the flow table size, or the control plane overloading, due to flow table lookups for counters retrieval. Moreover, the proposed method focuses only to the case of Distributed Denial of Service (DDoS) attacks to the data plane. In addition, the presented performance analysis is limited to the proposed intrusion detection algorithm and not to the overall system performance. Last but not least, the impact of data plane traffic anomalies to the OF control plane is not addressed.

Another attempt focusing on traffic anomaly detection on SDN environments is presented in [14]. Multiple anomaly detection algorithms were used in their experiments in order to validate their suitability on Small Office/Home Office (SOHO) environments. Authors introduce the idea that decentralized control of distributed low-end network devices using OF (e.g. devices in SOHO environments) can effectively detect and limit network security problems close to the source of the anomaly. Their work provides experimental results with respect to the efficiency of intrusion detection algorithms using only low network traffic rates (ranging from 60 to 12,000 packets per second) as they focus on Home and SOHO environments. Moreover, mitigation of the detected network anomalies is left as future work. On the contrary, we are investigating intrusion detection systems based on OF with much higher data rates (up to 130,000 packets per second) with analysis of the constraints that are introduced from the flow table size and of the impact of data plane traffic anomalies to the OF control plane. Such analysis does not exist in [14].

3. Design principles and overall architecture

3.1. Design principles

The design principles of the proposed architecture on anomaly detection and mitigation in SDN environments are based on the following key properties:

- *Modular design with data gathering, anomaly detection and anomaly mitigation function decoupling.*
  Our design includes three modules that logically distinguish the required functions, as it is depicted in Fig. 1. The first module is responsible for flow-based monitoring harvesting. Various methods may be used to collect the required flow statistics. The harvested flow data set is processed on the second module, which is responsible for the anomaly detection and identification process. The choice of the anomaly detection algorithm depends on the requirements of the network environment. The third module, responsible for attack mitigation, is based on OF mechanism.
- *Compatibility with any OF-enabled Layer 2 or Layer 3 device.*
  Our architecture keeps the harvested monitoring data decoupled from the anomaly detection and mitigation process. Hence, any type of OF network devices may be used either focusing on Layer 2 functions (often released with sFlow support), or on Layer 3 functions (often released with NetFlow [15] support).
- *Elimination of OF related constraints regarding statistics collection.*
  In a native OF environment flow statistics collection requires access to the counters of each flow entry inside the OF forwarding table; flow entries that correspond to aggregated flows cannot be used due to anomaly detection requirement of per (micro) flow statistics input. On the other hand, in cases of steep increase of the number of flow entries (e.g. during DDoS attacks), the use of aggregated flows is required. We propose a novel method, based on sFlow, which avoids the use of the OF flow statistics counters. Therefore, access to the packet counters of each flow entry is eliminated and aggregated flows can be used for forwarding purposes without affecting the operation of the anomaly detection mechanism. OF control plane is offloaded, as OF protocol is not used for statistics gathering. Otherwise, the network anomaly in data plane results to extensive message exchanging between the OF controller and OF enabled device, thus leading to flow table congestion.
- *Exploitation of data and control plane decoupling for fast anomaly detection and mitigation on real-time environments.*
  Anomaly detection is a CPU intensive process that is not feasible to be executed on the network gear. In parallel, anomaly mitigation on real-time environments requires a mechanism that is capable to manipulate the forwarding logic of the network gear. Therefore, an OF Controller manipulates the forwarding table of an OF-enabled network device (i.e. switch or/and router), establishing the anomaly related flow rules (e.g. dropping attack flows) which mitigates the anomaly under consideration.
- *Scalable traffic manipulation using packet sampling techniques.*
  Our approach permits traffic data counting process offloading to external components. As collateral benefit, possible resource depletion (i.e. CPU, memory of flow entry cache) is minimized during high traffic rates. We achieve this by using sFlow packet sampling to minimize the volume of flow-related data. High-end special purpose equipment is avoided.

3.2. Architectural components

Our work relies upon a subset of the 12-tuple flow definition (OF protocol v1.0 [16] as shown in Table 1), typically located as a flow entry in an OF switch, associated with four specific variables; (i) the action rule which consists the decision of how and whether any packet matched with the related flow entry will be forwarded, (ii) the soft-timeout variable representing the time needed for a flow to expire, since the last packet match, (iii) the number of packets that matched this specific flow since the flow entry establishment, and (iv) a specific priority value which is assigned to each flow entry, determining which flow rule will match in case of overlapping in the packet matching process.

The architecture of our proposed mechanism consists of three main modules shown in Fig. 1: (a) the Collector, (b) the Anomaly Detection and (c) the Anomaly Mitigation; each one responsible for what its name implies.

3.2.1. Collector

The Collector module is responsible for data gathering, a prerequisite in order to perform flow-based anomaly detection. This module collects flow information and periodically exports them to the Anomaly Detection module.

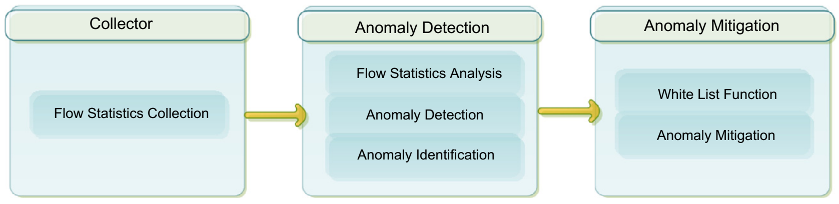


Fig. 1. Architectural view of the proposed IDS on SDN environments.

Table 1
Fields from packet headers used to match against OpenFlow flow entries.

LAYER 1	LAYER 2					LAYER 3				LAYER 4	
IN PORT	ETHER			VLAN		IP				PORT	
	src	dst	Type	id	PCP	src	dst	proto	TOS	src	dst

Regarding data gathering two different methods are investigated. The first one is based on an already proposed native OF approach [13,14], in which periodical queries and replies from flow-entries of the switch are collected. The second one leverages a flow monitoring mechanism utilizing packet samples. sFlow was chosen as it is highly vendor agnostic [17].

3.2.2. Anomaly Detection

Data produced by the previous module are subsequently fed to the Anomaly Detection module at periodic time intervals, thus creating discrete time windows. In our case, a time window of 30 s was chosen in order to accomplish near real-time detection consistent with similar studies reported in the literature [18,19]. For every time-window this module inspects all flow entries, exposing any flow-related network anomaly and identifying a potential attacker or the victim of the attack.

Our architecture can accept any algorithm such as statistical anomaly detection [20], machine learning based anomaly detection [21] and data mining based anomaly detection [22] as presented in [23,24] due to its modular design principle. We adopted a commonly used [5] entropy-based algorithm. Our mechanism can effectively detect DDoS, Worm propagation and Portscan attacks as shown in Section 5 of this paper. Moreover, it performs successfully not only in identifying the attack pattern, but also the attacker or the victim (i.e. host under attack). As soon as an anomaly is detected in the network, our algorithm inspects and correlates specific network metrics identifying the attack and exposing all related information to the Anomaly Mitigation module.

3.2.3. Anomaly Mitigation

The Anomaly Mitigation module aims to neutralize identified attacks, inserting flow-entries in the flow table of the OF switch (or modify existing ones) in order to block the desired malicious traffic. These flow-entries have higher priority than any other in the flow table. In order to avoid blocking benign network flows that resemble malicious anomalies behavior, we choose to insert specific host-related rules in a white list table.

4. Data gathering, anomaly detection and mitigation approaches for sdn environments

4.1. Approaches for flow-stats collection in SDN environments

4.1.1. The native OpenFlow approach

The native OF approach relies on the OF protocol for flow statistics collection. As the OF protocol mandates, periodic flow statistic requests are performed by the Controller for ordinary collection of all flow-entries residing in the OF switch, along with their corresponding counters.

Those flow counters are updated only when a forwarding lookup process matches a specific flow entry of the flow table. Hence, collecting flow statistics in a native OF environment is tightly connected with the packet forwarding logic, residing inside the OF controller. In our case, where the forwarding logic is dictated by the anomaly detection technique (which will be described in Section 4.2), Layers 3 and 4 protocol fields are employed. Thus, the required flow-entries of the flow table are a combination of variables {A, B, C, D, E, F, G, H, X, Y}, as shown in Table 2, where we present two flows signifying a bidirectional communication. The remaining fields of the flows are filled with wildcards, therefore matching any value of the respective fields.

Statistics collection with a native OF approach can be accomplished when a switch replies to a *flow-stats request* (which is the message that queries the switch for flow statistics) from the associated OF controller. As a consequence, the switch replies with large chunks of the flow table contents. Each chunk contains a portion of the flow-entries, along with the corresponding packet counters for each flow. However, the packet counters of the flow table records store the total number of packets since the instantiation of each rule, whereas the anomaly detection algorithm requires only the number of packets that matched each flow entry during the last time-window. Therefore, a record of the flow table's state for the previous time-window should be kept and compared to the current number of packets for each flow entry, in order to find the corresponding number of packets for the current time-window.

Table 2
Example of flow table entries.

LAYER 1	LAYER 2					LAYER 3				LAYER 4	
INPORT	ETHER			VLAN		IP				PORT	
	src	dst	type	id	PCP	src	dst	Proto	TOS	src	dst
X	A	D	G	*	*	B	E	H	*	C	F
Y	D	A	G	*	*	E	B	H	*	F	C

By utilizing the native OF collection method it is possible to gather and analyze the overall network traffic passing though the switch in full detail, as there is no sampling involved in the collection process. For that reason, this method has been successfully applied [13,14] to monitor low-to-medium traffic volume networks.

This monitoring capability of native OF approach does not come without major consequences. The number of flow-entries in the flow table of the edge switch may grow extensively (counted in tenths of thousands of flows per time-window), thus affecting the switching performance. For instance, assuming just 50 Mbps traffic, requires maintenance of about 24k flow-entries for at least 30 s in the flow table, whereas many commercial OF-enabled switches cannot maintain more than 4k flow-entries. Hence, this approach could be sufficiently used for software only OF implementation (e.g. Open vSwitch (OVS) [25]), which can carry a large flow table.

More importantly, a high rate attack in data plane may also cause a DoS attack to the control plane of the OF-enabled switch, since there will be thousands of *packet-in* events generated from the OF switch and directed to the OF controller, requiring an equal amount of flow-entries responses to be passed on and maintained in the flow table of the OF switch.

4.1.2. The sFlow-based approach

In order to overcome the hindrances presented in the aforementioned approach, we leveraged the packet sampling capability of sFlow in our proposed approach. The sFlow collection technique decouples entirely the flow collection process from the forwarding logic, because packet samples provide all the necessary flow-related statistics. Therefore the sFlow-based approach collects packet samples constructing the flow definitions and updating the packet counter of each flow entry centrally in the OF controller as a separate controller application. Afterwards, the sFlow Collector forwards all the necessary flow related statistics to the Anomaly Detection module.

This practice permits the use of more efficient and aggregated forwarding logic, since there is no longer a need for specific flow-entries as the ones used in the native approach. As a result, there is a significant reduction in the number of required flow entries, thus overcoming any possible flow table size limitations, which are typical for ordinary hardware OF switch implementations. At the same time, the sFlow collection process gathers sufficient information for a reliable anomaly detection process.

As the sFlow Collector receives packet samples on the fly, it updates the corresponding counters inside the monitoring module on a time-window basis. Hence, there is no need to constantly maintain and compare detailed flow statistics for each flow entry of consecutive time-windows. Consequently, this approach may reduce the complexity of the flow collection algorithm, thus requiring lower CPU resources. As it will be shown in the evaluation section by employing the sFlow approach, we effectively managed to increase tenfold the volume of monitored traffic compared to the previous native OF approach.

4.2. Entropy-based detection and classification of anomalies

We employed an entropy-based method, independent of network topology and traffic characteristics that can be applied to monitor every type of network for anomaly detection and classification purposes [7,26].

Entropy measures the randomness of a specific data set. High entropy values signify a more dispersed probability distribution, while low entropy values denote concentration of a distribution. In order to have a metric independent of the number of distinct values of the data set, we normalize the entropy by dividing it with the maximum entropy value of the data set, thus its values range in $(0,1)$.

The flow-related traffic feature distributions, that are valuable for our work, are the source IP address (srcIP), the destination IP address (dstIP), the source port (srcPort) and the destination port (dstPort). In this paper, we focus on three common types of anomalies, that are: (i) Distributed Denial of Service (DDoS), (ii) Worm Propagation and (iii) Portscan. Their inherent description and collateral entropy change behavior are shown in Table 3.

For instance, an anomaly of an infected host trying to infect other hosts in the Internet (worm propagation) results in decrease of the entropy of the source IP addresses. The infected machine produces a large number of flows, causing the same source IP address to dominate in the flow distribution of source IP addresses. On the other hand, during a port scanning activity, the entropy of the destination port increases due to the scan of random destination ports. Based on such fluctuations, the controller can identify the presence of an anomaly using predefined thresholds on the changes in the corresponding entropy values.

In Fig. 2 we present the variation of the aforementioned four entropy metrics during an indicative DDoS attack. Specifically, we inject a 6-min long DDoS attack, 10 min after initializing the monitoring process of our network

Table 3
Classification of anomalies based on entropy change.

Anomaly	Description	Entropy change
Distributed Denial of Service (DDoS) attack	An attack on a specific service, making the resource unavailable to its users	Significant decrease in dstIP and dstPort
Worm propagation	A self-replicating program that tries to infect other machines by exploiting a specific vulnerability	Significant decrease in srcIP and dstPort
Portscan	Sending probe packets to a wide range of ports in a specific host to check which services are available	Significant decrease in srcIP, dstIP Significant decrease in srcPort if attacker is not using random source ports

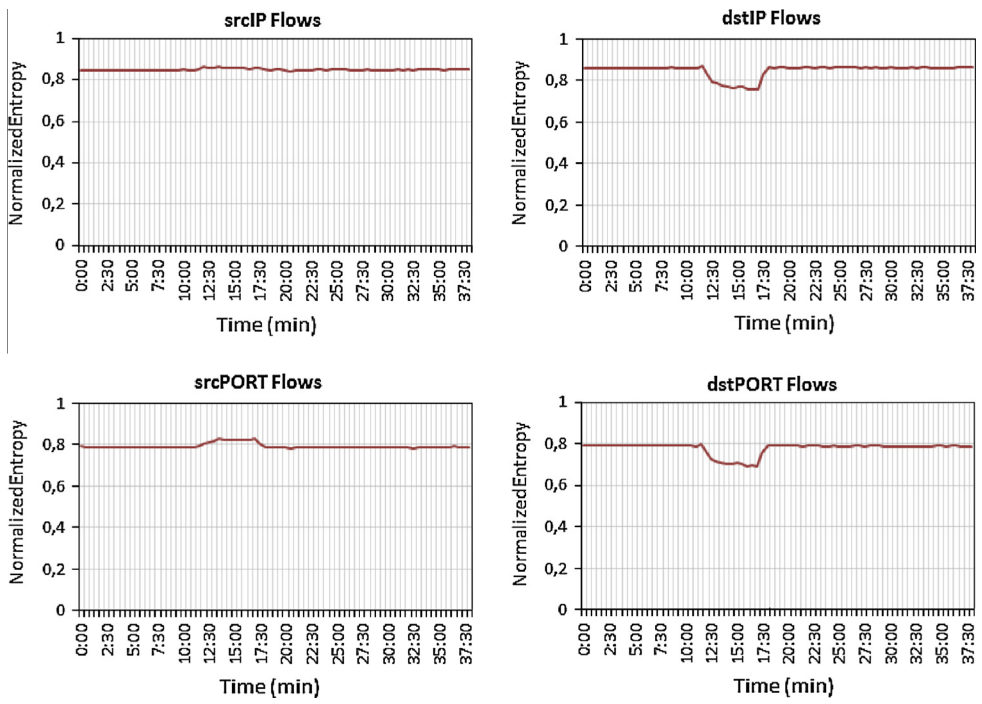


Fig. 2. Entropy values for flow-related traffic features.

traffic. As we can observe from Fig. 2, the destination IP and destination port entropy values decrease significantly during the attack, thus alerting us for the presence of an anomaly.

4.3. Attack identification and mitigation

In previous sections we have exploited OF and sFlow mechanisms to perform anomaly detection procedures on network traffic. In order to proceed to attack mitigation, we need to reveal the attacker and finally mitigate the result of the attack which is the malicious traffic. The huge advantage of OF is obvious on attack mitigation, because OF is tightly related to the forwarding functionality of any network component.

New flows are passed on to the switch flow table, along with a specific action. The most commonly used actions are the *Forward*, *Drop* and *Modify-field* actions [16]. *Forward* action is attached to each benign flow entry, while *Drop* action is used in order to block malicious traffic. Furthermore, we use predefined priority values for each different action. We assign a low priority value to flow-entries that are meant to forward packets matched to them, and a higher one to flow-entries meant to drop packets. Hence, a drop rule will always take precedence against a forwarding one.

Anomaly Detection module implements the observations of Table 3 and subsequently raises alarms. For example, if the alarm is triggered due to a decrease in the Destination IP and Destination port entropies, our algorithm presumes that the attack detected was actually a DDoS attack. The same applies to the other attack patterns. Afterwards, the same module analyzes the related with the attack pattern network metrics, from multiple time-windows to identify either the attacker, or the target(s) under attack. Hence, in case of a DDoS attack, we identify the host and the service under attack, while for the rest of the attack patterns we identify the attacker. In general, our purpose is to locate a specific service in a specific host, and apply policy rules to drop the malicious traffic.

Once the Anomaly Detection module has identified the host (whether this is the attacker or the host under attack), it passes the related information to the Anomaly Mitigation module. This module is able to instantiate a new flow in the OF-enabled switch or to modify an existing one, using the IP address and port number learned by the Anomaly Detection module and attaching a Drop action.

Although our algorithm can identify the attack patterns mentioned above, there are some benign network anomalies, which affect the network metrics the same way a malicious anomaly would have. For example a Flash Crowd anomaly would cause a heavy drop in the Destination IP and Destination port metrics, like a DDoS attack would have. In order to avoid cutting off a valuable service for our network (i.e. ftp), we also implemented a White List function that maintains a list of IP addresses/ports that their anomalous network behavior is related to legitimate network traffic. Hence, before the Anomaly Mitigation module inserts a drop rule in the switch, it first checks the combination of IP address and Port that was identified, against the White List table. The White List can be maintained via an automated mechanism or set manually by the network administrator. In our experiments, we implemented the latter for simplicity reasons.

5. Evaluation

In this section, we evaluate the combination of OF and sFlow protocols as an anomaly detection and mitigation mechanism. We mainly focus on the performance of OF and sFlow enabled switches and experiment with low to high traffic volumes. Moreover, we investigate the benefits of exploiting OF in order to identify and mitigate any detected malicious traffic, using the capabilities of the OF Controller (NOX) [27].

NOX is a modular event-based OF controller [28], which is exploited as a high-level programmatic interface upon network events. Through the API of the NOX Controller we implemented all three components as separate NOX applications, responsible for data gathering, periodical entropy-based calculations and flow table modification tasks.

We implemented the anomaly detection algorithm as a NOX application conducting periodically entropy-based calculations. The results of this procedure can then be passed to the Anomaly Mitigation module for enabling attack countermeasures. We developed a separate NOX application for mitigation purposes so that the mitigation process would not depend on the algorithm used for anomaly detection. This fact provides the user with the ability to develop or use the anomaly detection method of his preference, as long as he can then pass the necessary information to the Anomaly Mitigation module. Our performance evaluation experiments were based on production traffic collected from various locations of a university campus network.

5.1. Experimental setup

For our anomaly detection, identification and mitigation tasks we implemented all the necessary algorithms as NOX applications written in Python language. In order to conduct our experiments we required OF-enabled switches with sFlow support. For this purpose, we used Open vSwitch [29], a software switch that can handle the required traffic loads. The performance validation was made through NTOP [30], which monitored the egress ports of our software switch. This confirmed our expectations that Open vSwitch could reliably handle the forwarding of up to 100 Mbps traffic while exporting sFlow samples with 1/64 sampling rate, or up to 500 Mbps with 1/256 sampling rate. Our software switch was hosted on a Dual-Core 3 GHz with 8 GB RAM server. For 500 Mbps traffic rate, we also conducted the same experiments on a NEC IP8800/S3640 switch, in order to test our solution with hardware equipment and compare the capabilities of a software switch with a commercial hardware OF-enabled switch. Fig. 3 shows the experimental setup and the implemented NOX applications used to evaluate the two approaches mentioned above. For our evaluation, we injected the captured network traffic in a single switch (in our case, the software-based Open vSwitch or the hardware-based NEC-IP8800/S3640). Note that our proposed mechanism can be applied to more generic network topologies, with multiple OpenFlow-enabled switches and corresponding mitigation rules.

For low volume traffic networks, we evaluated and compared the native OF (OF Collector) and the sFlow-based (sFlow Collector) implementations. Specifically, for the first one, we had to set a specific value for the soft-timeout variable of each flow entry. This is due to the fact that we did not want any flow-entries to expire before the OF Collector module was able to gather the related flow statistics. Hence, we set the soft-timeout to 31 s, since our time-window was 30 s, as discussed in Section 3.2.

For the sFlow-based implementation we used the sFlow Collector implemented as a NOX application along with another NOX application responsible for controlling the forwarding logic of the OF switch, by performing a typical MAC learning process. As explained in Section 4.1.2, for the

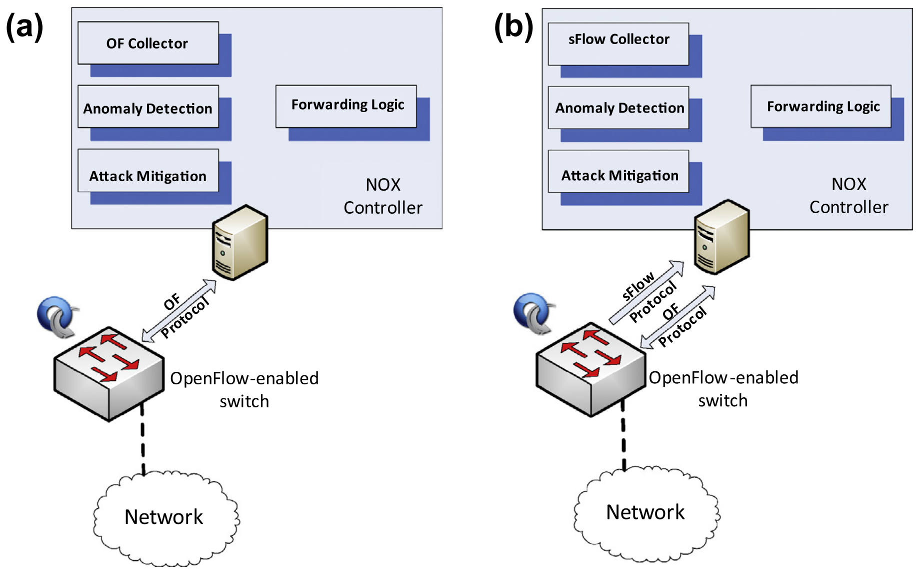


Fig. 3. Anomaly detection and mitigation architecture components for (a) native OpenFlow and (b) sFlow-based approach.

forwarding logic of this implementation we can just use any algorithm that fits best to our needs.

Table 4
Parameter values used in our experimentations.

	Average traffic rate (Mbps)	Sampling rate	Attack rate (pkts/s)
Exp. 1	50	No 1/64	50–200
Exp. 2	100	1/64	200–500
Exp. 3	500	1/256	1000–2500

5.2. Dataset and traffic generation

Aiming to evaluate our anomaly detection and mitigation mechanism's performance in a wide range of network environments, we collected benign traffic at three different locations from the campus network of the National Technical University of Athens (NTUA). Initially, we evaluated the performance in a low network traffic environment, by utilizing packet traces from our laboratory at the School of Electrical and Computer Engineering. In this way, we can emulate the network traffic conditions of a SOHO environment, with an average traffic load of 50 Mbps and about 35 active hosts.

For further evaluation of our proposed mechanism, we experimented with higher traffic of about 100 Mbps to emulate a larger enterprise environment. Hence, we captured network traffic from a part of the NTUA campus. Finally, in order to emulate the network traffic conditions of a small ISP, we captured packet traces from the entire NTUA campus. The average traffic load at the given time was around 500 Mbps. Trace files of 50 Mbps and 100 Mbps contain captured network traffic of a whole week, while the trace file of NTUA campus was two days long due to the significant size of files produced by capturing such high traffic rates. These trace files were employed in our experiments in order to evaluate the accuracy and detection capabilities of the native OpenFlow and sFlow approaches.

Tcpreplay tool [31] is used to replay captured packet trace files, injecting the generated traffic to a specific Ethernet port. Tcpreplay is able to replay the captured traffic at the speed it was captured. For our attack traces, we employed Scapy [32], a programmable tool that permits sending packet sequences with arbitrary protocol field values (e.g. src/dst IP address, port, etc.). Thus, in order to emulate a DDoS attack, we constructed SYN packets with a predefined set of destination IP address and port, and a random and constantly changed set of source IP address and port. Worm propagation scenario used the same generated traffic as the previous one, except from the fact that the destination IP address/port pair was random, while the source pair was predefined. Finally, for the Portscan scenario we constructed and injected packets with a specific source and destination IP address, while the source and destination ports were randomly selected in each packet.

5.3. Experiments – anomaly detection results at different traffic rates

In order to evaluate our proposed mechanism, we tested our algorithms under multiple and different network environments, with respect to the following three properties: (i) average normal traffic rate, (ii) sampling rate and (iii) attack rate as shown in Table 4. For the average network traffic rate, we used datasets of 50 Mbps, 100 Mbps and 500 Mbps. This practice proved that our method can be used for anomaly detection purposes in various environments, ranging from SOHO networks and campus departments to ISPs. Furthermore, we deployed our sFlow approach with sampling of 1/64 and 1/256 for the cases of 100 Mbps and 500 Mbps respectively. Finally, for each different network traffic rate, we injected packets emulating DDoS, Worm and Port scanning attacks, at varying packet rates.

We performed the following experiments:

- 50 Mbps of traffic monitored with the native OF implementation, as well as with the sFlow-based one, using 1/64 sampling rate. In both methods we utilized the OVS OF-enabled switch.
- 100 Mbps of traffic monitored using the sFlow-based implementation, with 1/64 sampling rate and the OVS OF-enabled switch.
- 500 Mbps of traffic monitored with the sFlow-based implementation, with 1/256 sampling rate, using the OVS OF-enabled switch. Then, we performed the same experiment with the NEC IP8800/S3640 switch, in order to evaluate whether our proposed mechanism could be used along with commercial hardware OF-enabled switches.

In our first experiment we aim to show that both our implementations can be used effectively to detect any anomalies in a small-to-medium network environment. We also want to compare the two methods and evaluate if the sFlow-based implementation can be used reliably for such a process, despite the sampling involved.

To this end, we replayed captured traffic of 50 Mbps, while injecting DDoS, Portscan and Worm attacks. In order to have varying attack rates, we injected attacks of 50, 100 and 200 packets per second. Considering that the legitimate packets per second had an average value of 12,000–13,000, the attack rate was significantly low.

The entropy values presented in Fig. 2 were a result of the native OF implementation used to perform anomaly detection. In Fig. 4, we compare the values of the four entropy metrics, shown in Fig. 2, during an indicative attack with the corresponding metrics from our sFlow-based implementation. From this comparison, we can observe that the results are almost identical and that the entropy values appear to have the same trend with or without sampling involved.

In the following, Receiver Operating Characteristic (ROC) curves are presented, with respect to true positive and false positive rates, for each of the attack patterns, under both implementations. To generate the aforementioned ROC curves we experimented with different attack (Table 4) and background traffic rates (Section 5.2).

Fig. 5 shows the ROC curves, for both native OpenFlow and sFlow implementations. As expected, the OF-based

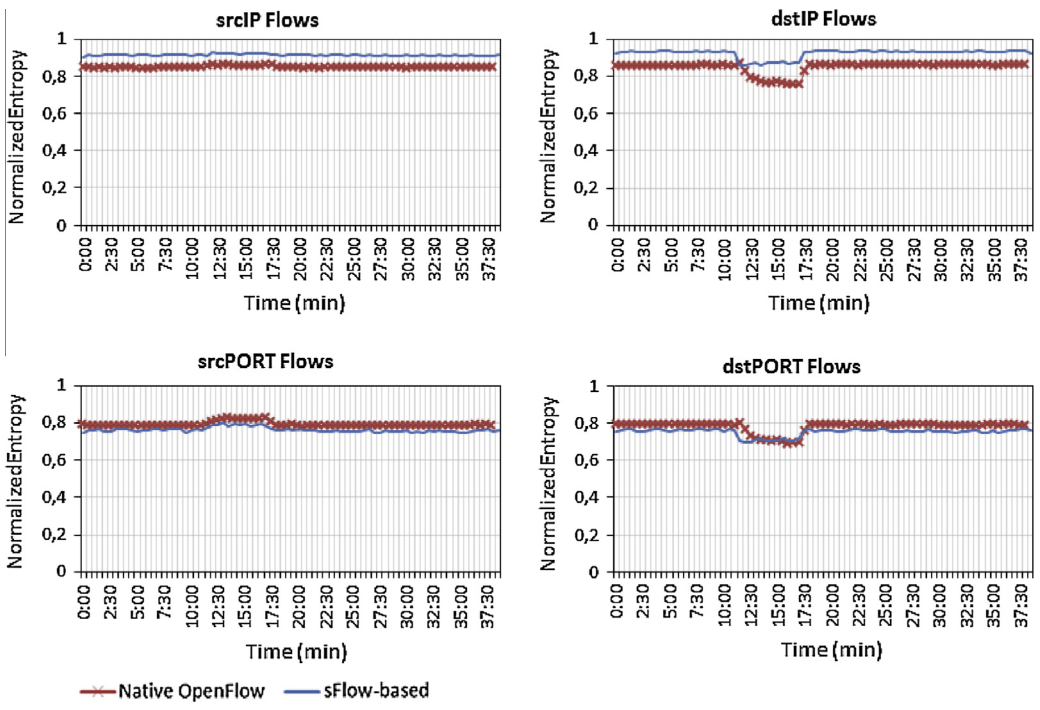


Fig. 4. Entropy values for native OpenFlow and sFlow-based implementations.

implementation performs better, due to the fact that there is no sampling. Thus, we could achieve 100% accuracy with a False Positive (FP) rate ranging from 23% to 39.3%, depending on the attack pattern. The sFlow-based implementation performed slightly worse, achieving 100% accuracy rate with a FP rate around 50% for each attack pattern. However, as it will be further discussed in Section 5.5, the OVS CPU load of the sFlow implementation was significantly less, compared to the native OF implementation.

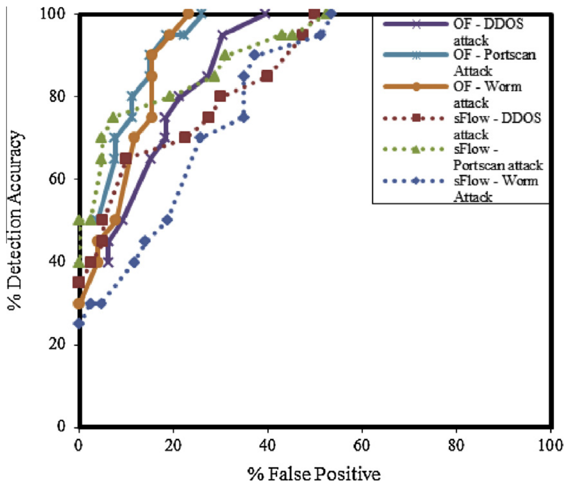


Fig. 5. ROC curves for native OpenFlow and sFlow-based implementations with respect to various network anomalies.

All native OF related issues presented in Section 4.1.1 may be overlooked while monitoring low traffic volumes such as the ones in our first experiment. In high traffic rates, such as 100–500 Mbps, anomaly detection tasks performed in a native OF manner (e.g. periodic processing of flow-stats requests/replies) does not perform well. Due to all aforementioned issues, any native OF approach requires a huge amount of CPU processing power from the OF Controller in order to handle both forwarding and anomaly detection tasks, therefore sometimes leading to significant packet forwarding delays. Furthermore, in many cases we observed that an attack on one or more hosts behind the OF-enabled switch could cause an actual DoS attack to the Control Plane of the network environment, due to CPU overloading caused by the intensive interaction between the OF switch and its Controller. Significant Control Plane overhead reductions were observed by employing the sFlow-based approach, especially in the two high traffic experiments (100 and 500 Mbps). More extensive measurements regarding the CPU usage in both approaches will be presented in Section 5.5.

In Fig. 6(a), we present the corresponding ROC curves for the second case of our experiments. The benign traffic was approximately 100 Mbps with an average of 25,000 packets per second. In this packet trace, we injected the same network attacks, using rates of 200, 350 and 500 packets per second. As we can observe from Fig. 6(a), using the sFlow-based implementation, we achieved 100% detection accuracy while having 40%, 42% and 50% FP rates, for the DDoS, Worm and Port scanning attack scenarios respectively.

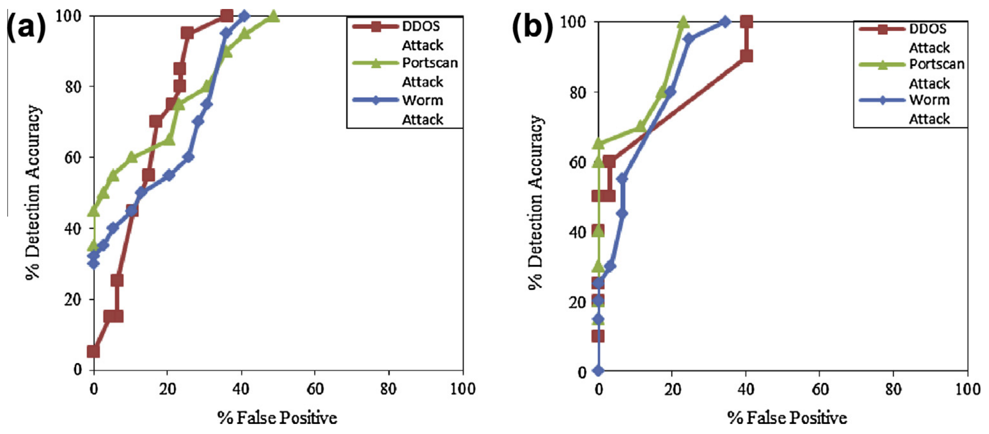


Fig. 6. (a) ROC curves for the 100 Mbps case and (b) ROC curves for the 500 Mbps case.

Fig. 6(b) shows the ROC curves for our last experiment. In this case, we tried to emulate a small ISP, with benign traffic of approximately 500 Mbps and an average of 130,000 packets per second. The attack rates used for this case were 1000, 2000 and 2500 packets per second. As it is depicted in Fig. 6(b), we achieved 100% detection accuracy while having 23%, 27% and 34% FP rates for the portscan, DDoS and worm attack scenarios respectively.

In Fig. 6, we observe that using 500 Mbps of traffic volume and 1/256 sampling rate, we achieve an equal, and in some cases better, detection accuracy rate, compared to the two aforementioned cases. The large traffic volume consists of several thousands of independent hosts, greatly randomizing the traffic characteristics. Thus, it is easier to detect network anomalies with our entropy-based method, although we used 1/256 sampling rate for this particular experiment.

Finally, we compared the results of the entropy values while using OVS and NEC IP8800/S3640 switch to monitor the same legitimate traffic volume of 500 Mbps. The values achieved with the OVS software switch were the same as with the NEC hardware switch, thus ensuring that the software switch can actually handle reliably such traffic volumes, while exporting sFlow samples at the same time.

5.4. Applicability and performance evaluation of other anomaly detection algorithms for the sFlow based approach

In the previous section, we evaluated the anomaly detection capability of the sFlow-based approach applied to OpenFlow-enabled SDNs, in terms of accuracy, employing entropy-based anomaly detection. In order to further evaluate the sFlow approach, we implemented the Threshold Random Walk with Credit Based connection rate limiting (TRW-CB) algorithm [33]. TRW-CB is well-known for its significant performance in detecting low-rate port scanning and worm attacks and has been applied in SDN environments (Section 2 – Related Work, [14]). The Entropy-based approach leverages baseline models of network traffic using predefined attributes and tries to detect deviations in values for these attributes. Unlike the Entropy-based method, TRW-CB performs statistics collection in a per host basis, therefore it is less sensitive to changes in the pattern of benign network traffic.

In this section, we present results from the implementation of the TRW-CB algorithm as the detection logic for the native OpenFlow and sFlow-based approach. The scope of such a comparison is to investigate the applicability of other anomaly detection algorithms to the sFlow approach, presenting the trade-off between accuracy and resource consumption when we favor sFlow.

The detection module implementation for the following experiment was based on the TRW-CB algorithm. Threshold Random Walk (TRW) [34] is an algorithm to rapidly detect portscanners based on observations of whether a given remote host connects successfully or unsuccessfully to newly-visited local addresses. TRW is exploiting the disparity between the frequency with which such connections are successful for benign hosts vs. for known-to-be malicious hosts. TRW-CB is an extension of TRW algorithm enforcing the continuous monitoring of a host, even after it has been found to be benign, in order to detect possible worm attacks. Furthermore, TRW-CB implements a Credit-Based connection rate limiting to ensure that worms cannot propagate rapidly in case of a late detection.

Fig. 7(a) and (b) depict the accuracy of the native OpenFlow and sFlow-based implementations of the TRW-CB algorithm for port scanning and worm attacks respectively. The benign traffic was approximately 50 Mbps, while the injected port scanning and worm attacks ranged from 50 to 100 packets per seconds, following the same principles presented in Section 5.3, and specifically in Table 4 (noted as Exp. 1). As we can observe, the results of the sFlow-based approach present a small increase in the false positives for detection rates higher than 85%, compared to the native OpenFlow approach. However, the sFlow approach has a superior performance in terms of system resources usage (CPU, number of flow-entries), as shown in the following section.

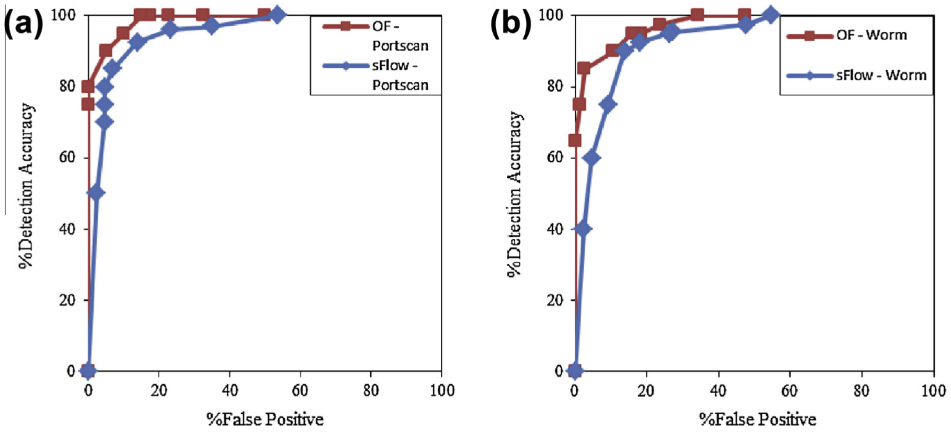


Fig. 7. ROC curves for the TRW-CB algorithm implementation under: (a) port scanning attacks and (b) worm attacks.

Table 5
Comparison of OpenFlow switch CPU usage and number of flows for the entropy-based and TRW-CB algorithms: sFlow-based versus native OpenFlow approaches.

	50 Mbps background traffic		50 Mbps background traffic with 200 pps attack injected	
	Average number of flows	CPU usage (%)	Average number of flows	CPU usage (%)
Entropy-based with sFlow	217	32	351	39
TRW-CB with sFlow	217	32	351	39
Entropy-based with Native OF	5184	47	7685	61
TRW-CB with Native OF	2022	42	2606	58

5.5. System performance evaluation

In this section, we evaluate system resources usage (CPU, number of flow-entries) for the OpenFlow Controller and the OpenFlow-enabled switches, in order to assess the capability of the sFlow-based approach to reduce the overhead introduced to the Control Plane in an SDN environment. The CPU usage of both the OpenFlow Controller and OpenFlow switches is a key performance indicator for the Control Plane operation; the number of flow entries in the table of the OpenFlow switch can impact the forwarding efficiency of the switch, as every incoming packet has to be matched against entries in the flow table.

In Table 5, we compare the average CPU usage of the switches for the TRW-CB and Entropy-based implementations, while using sFlow samples as input, against the case where a native OpenFlow approach is adopted for the same algorithms. Moreover, we show the average number of flows residing in the flow tables. Finally, we do the same comparisons in the case where an attack of 200 packets per second is injected and present the average number of flow entries observed during the attack period. The benign traffic at the time of measurement was slightly above 50 Mbps. As we can observe, the CPU utilization for the sFlow-based approach decreases from 61% to 39% for the case of the entropy-based algorithm, while the required amount of flow entries drops from 7685 to 351 entries.

As we can observe in Table 5, the Number of Flows and CPU usage for both algorithms, when the sFlow-based approach is used, are the same. This behavior is due to the fact that the packet forwarding application residing in NOX Controller is completely decoupled from the components used for anomaly detection. It is also worth noting that the increased value of the flow entries for the entropy-based algorithm compared to TRW-CB, is due to the fact that the first requires Layer 4 flow granularity, while the second utilizes Layer 3 flow information.

Table 6
Comparison of OpenFlow controller CPU usage for the entropy-based and TRW-CB algorithms: sFlow-based versus native OpenFlow approaches.

	50 Mbps traffic (%)	50 Mbps with 200 pps attack injected (%)
Entropy-based with sFlow	25	31
TRW-CB with sFlow	27	32
Entropy-based with Native OF	42	63
TRW-CB with Native OF	48	60

In addition to safeguarding the CPU usage of the OpenFlow switches, a major concern is the CPU overhead introduced to the OpenFlow Controller, as it can be severely affected from data gathering mechanisms. In Table 6, we depict the gain of the sFlow-based approach versus the native OpenFlow, in terms of CPU usage for the OpenFlow Controller. As we can observe, whether we use sFlow with the entropy-based or the TRW-CB algorithm, the required CPU cycles are reduced (e.g. from 42% to 25% for the entropy-based case), when compared to the respective native OpenFlow approaches. During the attack phase, the difference in the CPU utilization between the two approaches increases significantly (e.g. from 63% to 31% for the entropy-based case).

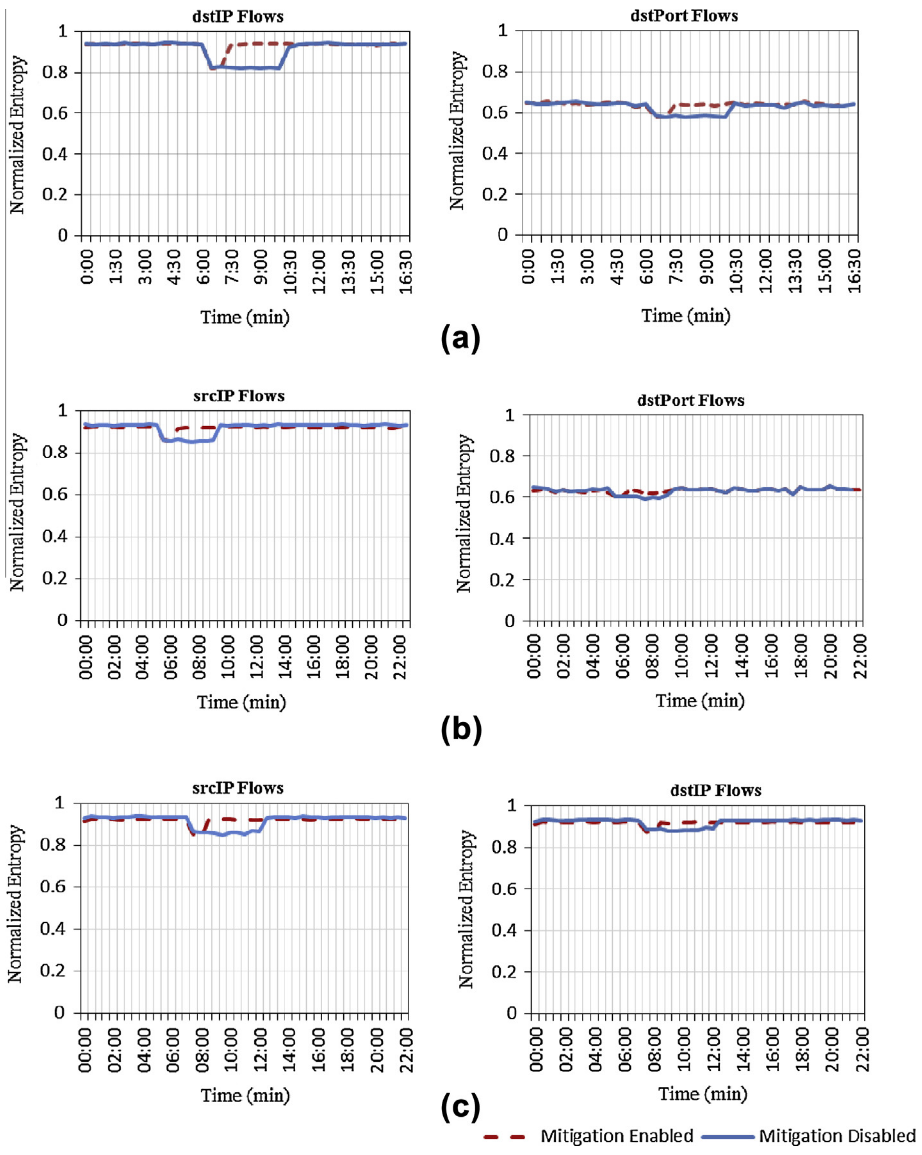


Fig. 8. Entropy values during indicative (a) DDoS, (b) worm and (c) port scanning attacks with and without the mitigation mechanism.

As shown in Tables 5 and 6, we can achieve a significant decrease in the CPU usage of the OpenFlow Controller and switches through the sFlow-based approach. Note that complete elimination of the CPU overhead introduced to the Control Plane from the data gathering process would be impossible, even in multi-core, multi-threaded systems.

5.6. Anomaly Mitigation

Up to this point, we have shown the results of using either native OF methods or sFlow-based methods to perform anomaly detection tasks in our network. In this section, we aim to demonstrate the added value of the OF protocol in such a mechanism and the reason to implement the sFlow-based approach as a module under the same SDN environment.

In this experiment, we took advantage of the same datasets, along with the same DDoS attack that was previously used to produce the entropy values shown in Fig. 4. Once the anomaly detection algorithm has detected the anomaly and identified it as a DDoS TCP attack to port N of host A, the Anomaly Mitigation module can drop the corresponding malicious traffic. Fig. 8(a) shows the entropy values of the corresponding network metrics when a DDoS attack is identified, in the case where the Attack Mitigation module is used (red[1] dotted line) or not (blue line). It is obvious that once the attack is identified and mitigated, the entropy values return to the expected range. In Fig. 8(b) and (c), we depict the mitigation of an indicative worm and port scanning attack respectively. As we can observe, the entropy

[1] For interpretation of color in Fig. 8, the reader is referred to the web version of this article.

Table 7
Drop rules in order to mitigate the different kinds of attacks.

ATTACK TYPE	LAYER 1	LAYER 2					LAYER 3				LAYER 4	
	IN PORT	ETHER			VLAN		IP				PORT	
		Src	dst	type	id	PCP	src	dst	proto	TOS	src	dst
DDoS	*	*	*	0×0800	*	*	*	A	0×06	*	*	N
DDoS	*	*	*	0×0800	*	*	A	*	0×06	*	N	*
WORM	*	*	*	0×0800	*	*	A	*	0×06	*	*	N
WORM	*	*	*	0×0800	*	*	*	A	0×06	*	N	*
SCAN	*	*	*	0×0800	*	*	A	B	0×06	*	*	*
SCAN	*	*	*	0×0800	*	*	B	A	0×06	*	*	*

values of Source-IP and Destination-Port for the worm attack, as well as the entropy values of Source-IP and Destination-IP for the port scanning attack, return to the expected range, once the attack has been detected and mitigated.

We have used a time-window of 30 s across all our experiments, at the end of which the anomaly detection process takes place, analyzing all the collected flow statistics until that point. Consequently, the Anomaly Mitigation is aware of the combination of variables {A, N} mentioned above. It is commonly accepted that a simplistic way to mitigate a detected DDoS attack is to cut-off the host under attack. This is due to the fact that attackers are counted in thousands and it would be inefficient to insert that many rules, cutting-off each attacker one by one. Hence, the Anomaly Mitigation module inserts a new flow-rule regarding the victim host, as shown in Table 7 with action Drop. The rule actually consists of two flow-entries representing a bidirectional connection. Along with these values, we assign a priority value higher than the priority of all forwarding flow-entries. Finally, we set a specific soft-timeout value, so that the two flow-entries will expire in a predefined amount of time, since they receive an idle state flag.

In case of a Worm or Port Scanning attack mitigation, we use the exact same logic mentioned above for the DDoS attack. The only difference is the wildcarded fields inserted in the flow for mitigating each type of attack. More specifically, in case of a worm attack, we aim to cut off the host "A" attacking a specific destination port "N". On the other hand, in case we target the mitigation of a Port Scanning attack, it is a common practice to cut off the host "A" performing the attack, and set the destination IP field to cover a whole subnet "B" (we employed classful subnets for simplicity). The reason behind setting the whole subnet and not just the detected destination IP(s) under attack is that, in all likelihood, the same attacker will try to scan the ports of more than one hosts of the same subnet, trying to find vulnerabilities.

6. Conclusions and future work

In this paper, we evaluated the applicability of native OF data gathering approach for anomaly detection purposes in SDN environments. Our study demonstrated that the periodic data gathering from the entire flow table that may contain tenths of thousands of entries, does not scale for high network traffic environments. Furthermore, this technique during an event of a medium rate DDoS attack in data plane could potentially end up as an actual denial of service attack in the control plane.

Hence, we proposed a combined mechanism comprised of: (a) reduced data gathering with sampling, implemented with the use of the sFlow protocol, (b) anomaly detection, implemented by entropy-based algorithm and (c) network-wide anomaly mitigation using OF. Our mechanism eliminates the flow statistics collection through forwarding tables' lookup and reduces the required communication between switches and OF controllers, thus easing the control plane overloading. We demonstrated that in low traffic environments the performance of our mechanism is actually comparable to the native OF implementation. Furthermore, the proposed mechanism can be reliably used to detect network anomalies in high traffic networks, while minimizing the false positive rates. Our implementation achieved to run efficiently, handling real-time traffic ten times bigger than that presented to the related work.

The design of our SDN oriented mechanism is modular, since the forwarding, data gathering, anomaly detection and anomaly mitigation functions, as well as their corresponding elements are decoupled and can easily be altered, optimized or adapted to future security threats. To this end, we demonstrated that other anomaly detection algorithms (e.g. TRW-CB) can be applied to our proposed architecture without affecting the anomaly detection accuracy. Finally, we evaluated our architecture in terms of system resources usage (CPU, flow table size) and demonstrated that the proposed sFlow-based approach presents superior behavior compared to the native OF mechanism.

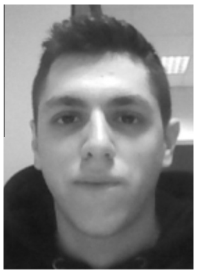
As future work, we plan to work towards the implementation and evaluation of other anomaly detection techniques as part of the Anomaly Detection module of our mechanism. For the attack mitigation process, we plan to study and experiment with inter-domain OF controllers' communication, in order to tackle attacks near their source.

Acknowledgement

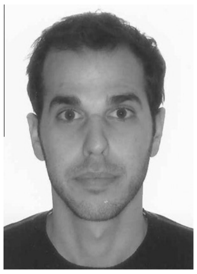
This work was partially supported by the European Commission, 7th Framework Programme for Research and Technological Development, Future Internet Research & Experimentation (FIRE), Grant No. 257867 – NOVI.

References

[1] Open Networking Foundation, Software-Defined Networking: The New Norm for Networks, ONF White Paper. <https://www.opennetworking.org/>.
[2] Nick McKeown et al., OpenFlow: enabling innovation in campus networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.
[3] P. Phaal, sFlow Specification Version 5, July 2004.
[4] M.S. Kim, H.J. Kong, S.C. Hong, S.H. Chung, J. Hong, A flow-based method for abnormal network traffic detection, in: Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP, vol. 1, pp. 599–612.
[5] A. Lakhina, M. Crovella, C. Diot, Mining anomalies using traffic feature distributions, in ACM SIGCOMM, 2005, pp. 217–228.
[6] Daniela Brauckhoff, Martin May, and Bernhard Plattner, Flow-Level anomaly detection – blessing or curse? in: IEEE INFOCOM 2007, Student Workshop, Anchorage, 2007.
[7] Georgios Androulidakis, Vassilis Chatzigiannakis, Symeon Papavassiliou, Network anomaly detection and classification via opportunistic sampling, IEEE Network: Mag. Global Internetworking – Special Issue Title Recent Develop. Network Intrusion Detect. 23 (1) (2009) 6–12.
[8] Arno Wagner, Bernhard Plattner, Entropy based worm and anomaly detection in fast IP networks, in: WETICE '05 Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise, 2005, pp. 178–177.
[9] Teemu Koponen et al., Onix: a distributed control platform for large-scale production networks, in: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, Vancouver, BC, Canada, 2012, pp. 1–6.
[10] Amin Tootoonchian, Yashar Ganjali, HyperFlow: a distributed control plane for OpenFlow, in: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking (INM/WREN'10), USENIX Association, Berkeley, CA, USA, 2010, p. 3.
[11] Nate Foster et al., Frenetic: a high-level language for OpenFlow networks, in: Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO '10), 2010.
[12] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feldmann, Teresa Vazao, Towards programmable enterprise WLANS with Odin, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HotSDN '12), 2012.
[13] Rodrigo Braga, Edjard Mota, Alexandre Passito, Lightweight DDoS flooding attack detection using NOX/OpenFlow, in: LCN '10 Proceedings of the 2010 IEEE 35th Conference on Local, Computer, 2010, pp. 408–415.
[14] Syed Akbar Mehdi, Junaid Khalid, Syed Ali Khayam, Revisiting traffic anomaly detection using software defined networking, in: RAID'11 Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection, 2011, pp. 161–180.
[15] Ed.B. Claise, Cisco Systems NetFlow Services Export Version 9, RFC 3954, October 2004.
[16] The OpenFlow specification version 1.0.0. <http://OpenFlowSwitch.org>.
[17] P. Phaal, S. Panchen, N. McKee, InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks, IETF, RFC 3176, 2001.
[18] Christos Siaterlis, Vasilis Maglaris, One step ahead to multisensor data fusion for DDoS detection, J. Comput. Sec. 13 (5) (2005) 779–806.
[19] Le The Quyen, Marat Zhanikeev, Yoshiaki Tanaka, Detecting and identifying network anomalies by component analysis, in: APNOMS'06 Proceedings of the 9th Asia–Pacific International Conference on Network Operations and Management: Management of Convergence Networks and Services, 2006, pp. 501–504.
[20] S. Staniford, J.A. Hoagland, J.M. McAlerney, Practical automated detection of stealthy portscans, J. Comput. Sec. 10 (2002) 105–136.
[21] Tarem Ahmed, Boris Oreshkin, Mark Coates, Machine learning approaches to network anomaly detection, in: SYSML'07 Proceedings of the 2nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques, 2007.
[22] Su-Yun Wu, Yen Ester, Data mining-based intrusion detectors, Expert Syst. Appl. 36 (3) (2009) 5605–5612.
[23] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, E. Vázquez, Anomaly-based network intrusion detection: techniques, systems and challenges, Comput. Sec. (Elsevier) 28 (1–2) (2009) 18–28.
[24] Animesh Patcha, Jung-Min Park, An overview of anomaly detection techniques: Existing solutions and latest technological trends, Comput. Netw.: Int. J. Comput. Telecommun. Netw. 51 (12) (2007) 3448–3470.
[25] Justin Pettit, Jesse Gross, Ben Plaff, Martin Casado, Simon Crosby, Virtual switching in an era of advanced Edges, in: 2nd Workshop on Data Center – Converged and Virtual Ethernet Switching (DC-CAVES), ITC 22, September 2010.
[26] S. Shah, A. Nucci, M. Munafo, R. Cruz, S. Muthukrishnan, DoWitcher: effective worm detection and containment in the internet core, in: INFOCOM 2007. 26th IEEE International Conference on Computer Communications, IEEE, 2007, pp. 2541–2545.
[27] Nox Repository Website. <http://www.noxrepo.org/>.
[28] Natasha Gude et al., NOX: towards an operating system for networks, ACM SIGCOMM Comput. Commun. Rev. 38 (3) (2008) 105–110.
[29] Ben Plaff et al., Extending networking into the virtualization layer, in: 8th ACM Workshop on Hot Topics in Networks (HotNets-VIII), New York, City, 2009.
[30] Luca Deri, Stefano Suin, Effective traffic measurement using ntop, IEEE Commun. Mag. 38 (5) (2000) 138–143.
[31] Tcpreplay. <http://tcpreplay.synfin.net>.
[32] SCAPY. <http://hg.secdev.org/scapy>.
[33] Stuart E. Schechter, Jaeyon Jung, Arthur W. Berger, Fast detection of scanning worm infections, in: Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID), 2004.
[34] Jaeyon Jung, Vern Paxson, Arthur W. Berger, Hari Balakrishnan, Fast portscan detection using sequential hypothesis testing, in: Proc. IEEE Symposium on Security and Privacy, 2004.



Kostas Giotis received his Diploma in Electrical and Computer Engineering from the National Technical University of Athens (NTUA), Greece in 2011. He is currently a Ph.D. Candidate at the Network Management and Optimal Design Laboratory (NETMODE) of the School of Electrical and Computer Engineering, National Technical University of Athens, Greece. His research interests lie in the area of computer networks with emphasis on network security, network virtualization and software defined networking.



Christos Argyropoulos received his Diploma in Electrical Engineering and Computer Science from the Polytechnic School of University of Patras. He is a Ph.D. Candidate at the ECE department of the National Technical University of Athens (NTUA) and a member of the Network Management and Optimal Design Laboratory (NETMODE) since 2008. He has been a teaching assistant in the course "Network Management and Intelligent Networks". He has been participating in EFIPSANS, NOVI and GÉANT (GN3) European FP7 projects. His main research interests lie in the area of computer networks with emphasis on network virtualization and software defined networking.

Georgios Androulidakis received his Ph.D. and Diploma in Electrical and Computer Engineering from the National Technical University of Athens (NTUA), Greece in 2009 and 2003, respectively. He is currently a Post-Doctoral Researcher at the Network Management and Optimal Design Laboratory (NETMODE) of the School of Electrical and Computer Engineering, National Technical University of Athens, Greece. His research interests lie in the area of computer networks with emphasis on network security, traffic analysis, network virtualization and software defined networking.

Dimitris Kalogeras is a Senior Researcher at ICCS within the NETMODE Laboratory. He obtained his Ph.D. (1996), Electrical & Computer Engineering from NTUA. His research spans several aspects of advanced network technologies and protocols. He is a senior consultant of GRNET (the Greek NREN) and the NTUA NOC. He was involved in several European RTD projects, e.g. on IPv6 (6Net) and on Network Security (GÉANT2 / GN2 – JRA2). He served in several EC technical panels and was for two terms with the Tech. Committee of TERENA.

Vasilis Maglaris, holds an Engineering Degree from the National Technical University of Athens – NTUA (1974) and a Ph.D. from Columbia University (1979). Until 1989 he held industrial and academic positions in the USA. He subsequently joined the School of Electrical & Computer Engineering at NTUA teaching and performing research on Internet technologies. He was responsible for establishing the NTUA LAN and the Greek National Research & Education Network GRNET. He served as Chairman of GÉANT, the European Research & Education Network (2006–2012). From July 2012 to June 2013 he served as the Secretary General for Research & Technology (GSRT) of Greece, responsible for planning and monitoring national research & innovation policies