

## RESEARCH ARTICLE

# DeMi: A Solution to Detect and Mitigate DoS Attacks in SDN

**LUBNA FAYEZ ELIYAN**<sup>1</sup> AND **ROBERTO DI PIETRO**<sup>2</sup>, (Fellow, IEEE)<sup>1</sup>College of Science and Engineering, ICT Division, Hamad Bin Khalifa University, Doha, Qatar<sup>2</sup>CEMSE Division, RC3 Center, King Abdullah University of Science and Technology, Thuwal 23955, Saudi Arabia

Corresponding author: Lubna Fayez Eliyan (leliyan@hbku.edu.qa)


This work was supported by the Award Thematic Research Grant Program from Hamad Bin Khalifa University (HBKU), Office of the Vice President for Research, Doha, Qatar, under Grant VPR-TG01-009.

**ABSTRACT** Software-defined networking (SDN) is becoming more and more popular due to its key features of scalability and flexibility, simplifying network management and enabling innovations in the network architecture and protocols. In SDNs, the most crucial part is the controller, tasked with managing the entire network and configuring routes. Given its critical role, a failure or problem occurring at the controller may degrade and even collapse the entire SDN. A typical threat controllers are subject to is a Denial of Service (DoS) attack. To cope with the above-introduced threat, in this paper we propose a lightweight DoS attack detection and mitigation method (DeMi) as well as a heavy-load management module. The proposed solution for detection leverages a sample entropy approach coupled with an adaptive dynamic threshold considering an exponentially weighted moving average (EWMA); the mitigation approach is based on proof of work (PoW) combined with flow rule installations; and, the heavy-load management method implements a scheduling approach at the SDN controller. Results are staggering: for instance, when DeMi is deployed, in an attack scenario the number of exchanged control packets is roughly similar to the attack-free scenario—without DeMi, the number of control packets in the network is 2,7 times more than what experienced in an attack-free setting. As per the number of re-transmitted packets, again, DeMi is able to achieve a re-transmission rate similar to an attack-free scenario—without DeMi the of packets that need to be re-transmitted is roughly 3,7 times the number of packets re-transmission occurring in an attack-free scenario. Moreover, DeMi does not block legitimate traffic, contrary to other solutions in the literature. The novelty of the approach, the demonstrated complete end-to-end solution, and the quality of the achieved experimental results, other than being interesting on their own, do pave the way for further research in this field.

**INDEX TERMS** SDN, DoS, DDoS, security, detection, mitigation, load balancing, proof-of-work.

## I. INTRODUCTION

The internet has revolutionized the development of communication and computer technologies. Cisco predicted that, by 2023, the number of devices connected to the network would increase from 18.4 billion in 2018 to almost 30 billion devices [1]. Coupling what before with the estimate that more than 50 billion devices will connect to the internet by 2025, [2], it becomes evident that there is a need for a networking infrastructure that keeps up with the

The associate editor coordinating the review of this manuscript and approving it for publication was Dominik Strzalka .

ever-changing landscape of users, resources, and services. Such an infrastructure has several new demands, such as scalability, security, flexibility, and reliability [3]. Traditional hardware-based networks operate inadequately because of constantly changing computing and storage needs. Consequently, SDNs gained significant traction as a novel network architecture in which several characteristics demand a more flexible and dynamic approach [4], [5].

The main difference between SDN and traditional hardware-based networks is the separation of control and data planes. The control plane is managed by the SDN controller, which has essential functions. These functions include

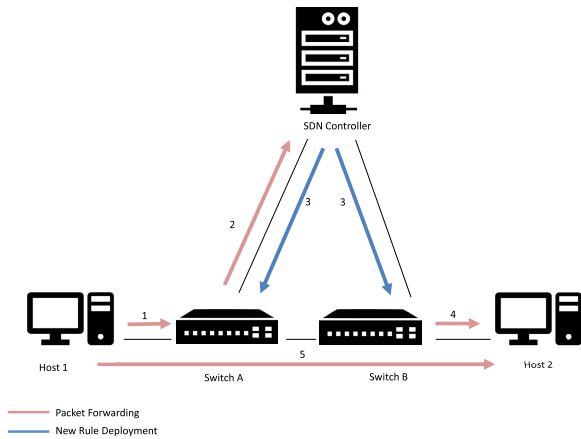


FIGURE 1. SDN operation.

flow table management, link discovery, topology, and storage management [6]. On the other hand, the data plane consists of forwarding devices known as OpenFlow switches. Each OpenFlow switch consists of one or more flow tables that perform packet lookups and forwarding to route the incoming data streams [7], [8]. Fig. 1 shows the core elements as well as the operation procedure of the SDN.

The nature of the SDN enables the data plane only to forward the incoming data stream according to the flow table rules created by the SDN controller. When there is no matching flow entry for the incoming data stream, a table-miss process occurs. In such a process, a “packet\_in” message request is sent to the controller from the OpenFlow switch to resolve the query of the new stream. Such a process could involve sending the header of the packet of the new stream or the complete packet, depending on the available buffer of the OpenFlow switch. Indeed, with a huge volume of network traffic, the switch may send the entire packet to the controller, which may consume high data bandwidth [9]. Once the controller receives the “packet\_in” request, it evaluates the routing path of the new stream and installs a flow rule at the involved switch to instruct how to forward the sequel of the data stream. By default, the requests are queued in the controller’s buffer upon arrival and are served in a first-come-first-serve manner. If the buffer is full, new requests will be dropped and will never be served [10].

The controller can handle a large but limited number of simultaneous requests at a time. Under a DoS/DDoS attack, the attackers generate a massive number of packets with randomly forged headers, making them hardly match the existing flow rules of the OpenFlow switches. This results in initiating multiple table-miss processes that, in turn, generate massive “packet\_in” requests. Such requests overload the controller, resulting in resource exhaustion and bandwidth/memory saturation [11]. Moreover, such overload floods the communication channel between the control and data planes, resulting in dropped communications between the planes under heavy attack loads [12]. Eventually, the controller will fail and will

not be able to serve legitimate hosts or traffic. Thus, the new flow requests of the legitimate hosts are delayed or dropped together with other attacking requests as they are not handled differently by the controller [13]. Therefore, a severe impact on the network’s availability, reliability, and efficiency occurs due to the attack.

As it can be seen, a failure or problem occurring at the controller may degrade and even collapse the entire SDN network. The above-discussed threat triggers the need for an efficient and high-performance DoS detection and mitigation approach to identify and respond to incidents before they might negatively affect the network [14]. Securing the SDN controller from a DoS attack is a challenging and resource-intensive task that, if not carefully engineered, could reduce the effectiveness of the controller in managing the network. This is even more so given that there are different types of DoS attacks on SDN [11]. Such attacks are easier to introduce, yet, more difficult to prevent and more destructive when compared with other attacks [15]. Therefore, any effort to secure the SDN infrastructure against DoS attacks requires a comprehensive understanding of SDN characteristics and how those DoS attacks affect the internals of SDNs [16].

Based on the existing technology, this paper proposes a lightweight DoS detection and mitigation method as well as heavy-load management. Deploying methods inherited from traditional networks to handle DoS/DDoS attacks in SDNs is neither straightforward nor (in many cases) feasible. The main reason for that is the architecture and working scheme of the SDN, which is different from the traditional networks. In SDNs, the network logic is abstracted in the controller, whereas in traditional networks, it can be distributed among different devices. Thus, in this paper, the proposed approach employs the main properties of the SDN that do not exist in traditional networks. More specifically, the approach utilizes the SDN controller that has an entire view of the network. Thus, deploying the solution on it makes it very effective for attack detection, mitigation, and management of heavy loads. The second main property of the SDN environment is the forwarding functionality of the new flows to the controller. Such functionality enables the controller to inspect the flows and identify suspicious behavior in the network at a higher and more manageable level compared to deploying the solution at the network devices level—like it is sometimes the case for traditional networks.

**Contribution:** Our solution provides the following features:

- a DoS detection method using sample entropy with an adaptive dynamic threshold for DoS attack detection;
- a mitigation method using the proof of work (PoW) approach combined with flow rule installations;
- a heavy-load management method using a scheduling approach at the SDN controller; and,
- an end-to-end solution to secure the components of the SDN to protect it against the discussed DoS attacks. More specifically, these components are:

- OpenFlow switches—the proposed approach prevents overflow of the forwarding tables by preventing the SDN controller from installing flow rules of the attacking traffic into the OpenFlow switches;
- communication channels—the proposed approach reduces overhead on the communication channel by controlling the number of requests sent by each OpenFlow switch to the SDN controller; and,
- SDN controller—the proposed approach protects the controller from a massive number of new requests that cause the controller to fail.

**Roadmap:** This paper is organized as follows: In Section II, we discuss related work about DoS/DDoS attacks on SDNs and existing solutions to address such attacks as well as the main components used in the corresponding implementations. Then, in Section III, we discuss the adversary model considered for the proposed solution of this work, followed by the details of the proposed solution in Section IV. In Section V, we evaluate and show the experimental results of the proposed solution, while Section VI draws some conclusions.

## II. RELATED WORK

Many techniques have been proposed to detect DoS attacks against the SDN controller to secure the network. In the following such techniques are classified into five main categories: entropy-based approaches, machine learning approaches, statistics-based approaches, scheduling-based approaches, and data plane-based (stateful) approaches. Each of these approaches has its pros and cons. This section will shed light on some of these adopted approaches and set the ground for our contribution.

Starting with the entropy-based approaches, information entropy is a statistical measure that shows the randomness in a data set. A higher entropy value indicates that the distribution of measured attributes is relatively scattered, while a low entropy value indicates a more concentrated distribution. Many approaches have adopted the entropy approach for DoS/DDoS attack detection and mitigation. For example, Mousavi and St-Hilaire [17] adopted Shannon entropy [18] over destination IP addresses. Once the evaluated entropy decreases below a predefined static threshold for a given number of consecutive time windows, an attack is declared. The same approach is adopted by Wang et al. [19] with a variation of evaluating the entropy at each edge OpenFlow switch, making the detection distributed. However, placing the detection mechanism inside the OpenFlow switch requires upgrading them to support such a feature. The entropy-based approach considering the destination IP address is also adopted in [12] and [20] with a static threshold value. No mitigation process is proposed in [12], while the targeted port of the specified switch is blocked in [20]. Other authors have considered different features to detect DoS attacks in entropy-based approaches, such as source IP address [21], [22] and the number of flows [23]. The attack is detected when the evaluated entropy exceeds a predefined

static threshold in [21], a dynamic threshold in [22], or is lower than a predefined static threshold [23].

For a more precise entropy evaluation, the authors in [24] and [25] considered more than one feature for attack detection. These features are pairs of source IP and port addresses and their corresponding destination addresses. The attack is mitigated by limiting the inflow rate from the switch to the controller. Instead, the authors in [25] considered the destination IP and port addresses as features. The attack is mitigated by installing flow rules that prevent all packets from reaching the target host, which might affect the legitimate hosts.

Other works adopted the entropy-based approach for DoS attack detection in flash events. For example, the authors in [26] utilized general entropy (GE) and generalized information distance (GID), initially proposed by [27], to identify the variations in the traffic behavior of flash events. However, no mitigation approach is proposed. Other authors, as in [28] and [29], used  $\phi$  entropy, which is an enhancement of Renyi's GE [30] and GID metrics proposed by [31]. The attack is detected when the value of the  $\phi$  entropy is lower than a static threshold for several consecutive windows. However, the proposed approach in [28] is for DDoS detection in traditional networks, not considering DDoS attacks in the SDN network environment. The same approach and limitations can be found in [29].

Other works consider joint entropy to have a more accurate attack detection. For example, the authors in [32] used flow duration and packet length. The evaluated entropy is compared to a pre-defined static threshold for attack detection with no proposal of the mitigation approach. Similarly, the authors in [33] utilize joint entropy, evaluated for all possible combinations (in pairs) of the transmission control protocol (TCP) layer attributes. The mitigation is achieved by dropping the entries from the controller.

Other researchers have considered the adaptive threshold value for the entropy approach to reflect the fluctuation in the network traffic. In [34], [35], [36], and [37] the adaptive threshold is updated each time the evaluated entropy reaches the threshold. Other approaches, as in [38], consider the adaptive threshold using the exponential weighted moving average (EWMA) algorithm proposed by [39] while, in [40], the threshold is updated based on CPU utilization factor.

Other authors have combined the entropy-based approach with classification algorithms to enhance attack detection. The entropy is used as an initial identification step of the abnormal behavior, followed by triggering the classification algorithms. For example, the authors in [41], [42], [43], [44], [45], and [46] trigger the classification algorithm for extracting additional flow features, while the particle swarm optimization (PSO)-BP neural network algorithm is triggered in [14], and the BiLSTM-RNN neural network algorithm is triggered in [47]. Similarly, authors in [48], [49], and [50] combine the entropy approach with a convolutional neural network (CNN) model to distinguish normal traffic from suspicious traffic. Deep learning approaches are also considered

in [51], [52], [53], and [54] for more precise attack detection, where authors in [53] considered neural networks while authors in [54] utilized the information gain (IG) and random forest (RF) in order to analyze the most comprehensive relevant features of the attack.

Finally, the authors in [55] propose a machine learning-based misbehavior detection system that considers machine learning algorithms to detect nine types of attacks on SDNs, including DoS attacks.

Another well-known approach for addressing DoS/DDoS attacks in SDNs is the statistics-based approach. Such an approach involves collecting and analyzing specific network properties to detect abnormal behavior in the traffic. These statistics include but are not limited to type, size, number of packets, number of half-open connections, and rate of packets associated with a particular application or port number. For example, the work in [56] collects statistics related to packet characteristics, while in [57], the authors collect the number of packets exchanged by sources and destinations. In comparison, the statistics collected in [58] is the number of packets that do not have valid IP destination addresses. The authors in [59] and [60] collect statistics related to the rate of incoming packets within specific time periods, while the mitigation is done by dropping incoming packets of the connected host in [59]. In [60], the mitigation process blocks the attack source by tracing back over the network connection.

Other contributions are based on the collection of statistics related to the number of incomplete connections and compare them to a specific threshold within a time window, as in [61], [62], and [63]. However, considering pre-defined thresholds in attack detection might generate false positives and inaccurate detection as the behavior of the users might change during the network operation. Additionally, the cited contributions involve the victim host (server on the network) in the detection process—requiring additional overhead on the host.

Finally, the solution discussed in [64] collects statistics about the number of connections per host and compares them against a threshold for attack detection, while the mitigation is achieved by installing a flow rule to drop the packets of the attacking host.

Other approaches collect statistics of specific parameters of each unique source IP address as in [65]. The collected statistics are the number of packets transmitted, their size per flow (in bytes), and the amount of time the new flow entry has spent in the OpenFlow switch.

Some works extract more detailed properties related to the traffic for attack detection. For example, the approaches proposed by [66] and [67] bind the MAC/IP addresses of the connected hosts and the OpenFlow switches. Similarly, the authors in [13] and [68] collect statistics related to “packet\_in” messages, including source and destination MAC addresses and source and destination IP address properties.

Some works address the DoS/DDoS attacks by maintaining the network’s availability through scheduling approaches on the controller’s side. The scheduling aims to provide a fair share of the resources and handle legitimate hosts’ requests while dropping attacking hosts’ requests through the starvation process. For example, the work in [69] prioritizes the requests’ handling based on the trust levels of the hosts, where highly trusted hosts have the highest priority. The trust level is also considered in [70] with a variation of assigning less timeout to the flow rules of the malicious switch. Other works deploy multiple queues to schedule the incoming requests. For example, the authors in [71] assign each OpenFlow switch a separate queue while inserting delays between the incoming requests to limit the effect of the DDoS attack. The approach in [72] defines functional and user-related request queues. The user-related queues are served after the functional queue is empty. This approach protects critical functional requests from starvation due to DDoS attacks, where plenty of user-related requests are received. Finally, the work in [73] proposes a multi-layer fair-queuing approach where each OpenFlow switch is assigned a separate queue being served considering the weighted round-robin (WRR) algorithm.

Other solutions employ the OpenFlow switches to detect and mitigate DoS/DDoS attacks. For example, Avant-Guard [74] and OF-Guard [75] add a packet filtering function on the switch side. Similarly, the authors in [76] employ the OpenFlow switches for traffic monitoring and entropy evaluation while [77] and [78] employ OpenFlow switches to make decisions for attack detection. On the one hand, these approaches could have a fast reaction time and prevent aggregation of the attack to the controller. On the other hand, they may lead to delays because of the added functionalities of the OpenFlow switches. Such functionalities involve the OpenFlow switches in more complex operations such as statistics collection, packet processing, and decision-making [79]. Therefore, employing switch-side solutions requires the OpenFlow switches to be more intelligent, violating the SDN’s design principles. Thus, having a switch-side solution is controversial in the literature since it brings some capabilities to switches. In contrast, the SDN’s key rationale is to provide decisions in a centralized manner with a very streamlined data plane [33].

Finally, some researchers introduced solutions that use proof of work (PoW) techniques to penalize hosts performing (alleged) suspicious activities. For example, in [80] and [81], all the connected hosts need to spend a considerable amount of computational resources (PoW) before obtaining a connection with the controller. The controller, in turn, can easily verify the PoW with relatively low overhead compared to the host. The solutions can limit the effect of initiating DoS/DDoS attacks on the controller. Nevertheless, the legitimate hosts are affected in terms of spending some resources before obtaining the connection to the network. Additionally, the two cited works fail in handling network availability when



the latter one is under attack and do not support host promotion from suspicious to legitimate. Additionally, the solution in [81] is implemented on the OpenFlow switches of the SDN, which violates the key rationale in providing decisions in a centralized manner with a very streamlined data plane, as it is discussed earlier. Table 1 shows a summary of the most relevant related works compared against our solution, where the considered features focus on detection, mitigation, or graceful degradation. We also list major highlights for reported proposals.

Through the above literature analysis, it emerges that detection schemes based on information entropy are simple and take up fewer resources with a relatively low computation overhead. The machine learning method is highly accurate but complex with regard to detecting the switch to which the victim host is connected to. Additionally, it needs pre-sets and training for the specified attack patterns. The statistics-based approaches require the controller to collect the flow table information of each switch, extract the features, and then perform detection cyclically and periodically. The cited operations might consume the controller's resources and impose delays over the attack detection and mitigation. Scheduling-based approaches can help maintain network availability, while the switch-side approaches are against the key principle of the SDN design. As it can be seen from Table 1, most of the existing works do not consider all the requirements needed to provide full protection to the network when addressing DoS/DDoS attacks. The main requirements are detection, mitigation, and maintenance of network availability—fully isolating the victim host or OpenFlow switch cannot be considered a viable mitigation approach. Thus, our approach described in the following sections aims to detect and mitigate the attack and maintain the network's availability under attack scenarios, including promoting hosts back to full legit node status while at the same time managing heavy-load conditions.

### III. ADVERSARY MODEL

The nature of the reactive packet processing mechanism in SDN can be a vulnerability exploited by the attackers to initiate the DoS/DDoS attacks. Such a mechanism is based on sending “packet\_in” requests to the controller for the flaws that do not have matches at the flow tables of the OpenFlow switches. The controller, in turn, installs a flow table entry (rule) on the involved OpenFlow switch to instruct it on how to route the new flow in the network. Such a mechanism can be effective for the centralized and managed control nature of the SDN. Yet, it can be a vulnerability when the number of new flows targeting the network is massive and overload the controller's processing power. Thus, the DoS/DDoS attacks can be achieved in SDN by initiating a massive number of new flows that cause a high volume of “packet\_in” requests within a relatively short time. In order to initiate such new flows from a single source (DoS) or multiple sources (DDoS), the attackers need to make the headers of such flaws hardly match the entries of the flow tables of the OpenFlow switches.

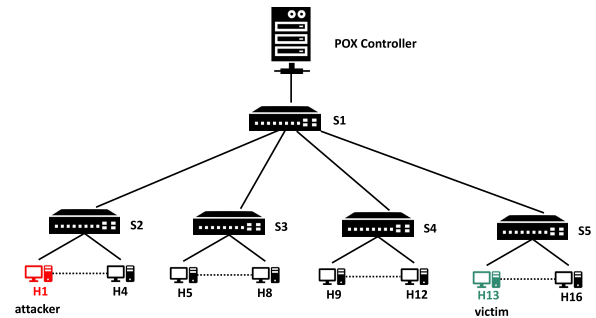


FIGURE 2. Considered adversary model.

This can be done by forging some or all fields of the packets in the flow randomly, making it hardly match any existing flow entry of the OpenFlow switch. In our attacker model, we assume that the attackers have successfully connected to the SDN through the three-way traditional handshake TCP procedure. After that, the attackers exploit the reactive packet processing mechanism's vulnerability by flooding malicious packet requests to the OpenFlow switches. Such requests are addressed to a specific host in the network. At the same time, the used source IP address and the rest of the header fields are forged with deliberately random values, making it almost impossible to match the OpenFlow switch table entries. This results in numerous table miss-matches in the forwarding tables, and many “packet-in” messages are flooded to the controller, resulting in the resource exhaustion of the entire SDN. More specifically, under the specified DoS attack, the massive number of new flows saturates the OpenFlow switches (data plane), the communication channel, and the SDN controller (control plane). Therefore, the SDN would not be able to provide any service for legitimate hosts or traffic. Fig.2 shows the considered adversary model.

### IV. THE PROPOSED SOLUTION

In this section, we discuss the details of the proposed solution. We start with an overview of the solution, followed by the details of the different modules that build the solution.

#### A. OUR SOLUTION AT GLANCE

DeMi maintains the network's availability and ensures the legitimate hosts' connectivity. DeMi is an application that runs on the SDN control plane and communicates with the data plane through a southbound interface via the OpenFlow protocol. DeMi is composed of a multi-stage process responsible for ensuring the network's availability, observing and controlling the connected hosts' activity, and maintaining the network's resources in terms of computation, such as memory, central processing unit (CPU), and buffer.

The main stages of DeMi are attack detection, mitigation, heavy-load, and system management. These stages are achieved by a composition of modules running inside the network. Such modules aim to detect hosts that initiate a

TABLE 1. Summary of related work in terms of focus.

| Reference  | Detection | Mitigation | Graceful degradation | Highlights  |
|--|-----------|------------|----------------------|---|
| [12](2019), [17](2015), [23](2019), [26](2018), [32](2018) | ✓         |            |                      | Considered static threshold for entropy evaluation in detection   |
| [19](2015), [21](2020)                                     | ✓         | ✓          |                      | Solution is deployed at the edge OpenFlow switches  |
| [20](2019), [24](2019)                                     | ✓         | ✓          |                      | Considered static threshold value for entropy evaluation in the detection   |
| [25](2014)   | ✓         | ✓          |                      | Considered static threshold for entropy evaluation<br>Introduced additional overhead due to the extra security layer  |
| [33](2018)   | ✓         | ✓          |                      | Attack-free traffic is used to establish nominal profiles which are difficult to obtain in real networks  |
| [41](2021), [45](2020), [46](2023), [47](2019)             | ✓         |            |                      | Solution is based on a machine-learning approach requires training and testing<br>Requires additional processing and data analysis  |
| [14](2019)   | ✓         |            |                      | Solution is based on a machine-learning approach<br>Solution partially deployed on the OpenFlow switch  |
| [56](2014), [68](2022), [75](2015), [76](2017)             | ✓         | ✓          |                      | Solution involves OpenFlow switches in the implementation   |
| [57](2018)   | ✓         |            |                      | Solution performs extensive analysis on the captured traffic<br>Solution is not lightweight   |
| [60](2016)   | ✓         | ✓          | ✓                    | Extra security layer, additional overhead/extension of the SDN architecture   |
| [65](2017)   | ✓         | ✓          |                      | Implies additional workload on controller<br>Implies communication overhead with switches<br>Mitigation process does not necessarily cease the attack all the time  |
| [13](2020)   | ✓         | ✓          |                      | Additional overhead on the controller due to storage and data collection and manipulation   |
| [69](2015)   |           |            | ✓                    | Attack's impact is reduced without eliminating the attacker<br>Inefficient for high resources controller as low priority queues are processed fast  |
| [70](2016), [71](2015), [72](2020), [73](2016)             |           |            | ✓                    | Attack's impact is reduced without eliminating the attacker<br>Additional overhead is implied on the controller   |
| [74](2013)   | ✓         | ✓          | ✓                    | Requires application-specific hardware and large memory<br>Involves OpenFlow switches in implementation   |
| [77](2018)   | ✓         |            |                      | Approach involves OpenFlow switches in the implementation (partially)   |
| [80](2016)   | ✓         |            |                      | Introduces overhead on the legitimate hosts   |
| [81](2022)   | ✓         |            |                      | Introduces overhead on the legitimate hosts<br>Involves OpenFlow switches in the implementation   |
| [61](2018), [62](2020), [63](2021)                         | ✓         | ✓          |                      | Static threshold for attack detection<br>Involve victim server in detection/mitigation procedure  |
| [22](2021)   | ✓         |            |                      | Additional overhead on the controller due to storage and data collection and manipulation   |
| [34](2018)   | ✓         | ✓          |                      | Implies additional communication costs between the controller and the OpenFlow switches   |
| [37](2023)   | ✓         |            |                      | Requires tuning of the threshold and number of windows considered for attack detection  |
| [40](2022)   | ✓         | ✓          |                      | Requires additional packet analysis at the controller's side<br>Entropy threshold considers CPU utilization factor, inaccurate estimation for high-performance CPUs   |
| [49](2021), [50](2020), [53](2021)                         | ✓         |            |                      | Considers deep learning approach for attack detection<br>Requires additional analysis at the controller side  |
| [54](2022)   | ✓         |            |                      | Considers deep learning approach for attack detection<br>Solution utilizes IG and RF with comprehensive analysis of features for detection  |
| [64](2022)   | ✓         | ✓          |                      | Solution is for low and slow attacks, not for fast attacks  |
| [78](2021)   | ✓         | ✓          |                      | Solution involves OpenFlow switches in the detection process  |
| <b>DeMi</b>  | ✓         | ✓          | ✓                    | Adaptive dynamic threshold for entropy evaluation<br>Lightweight solution does not involve extensive analysis/operations<br>Does not involve the OpenFlow switches in the implementation<br>Does not require additional hardware or an extension of the SDN architecture<br>Eliminates the attack impact in the network |

massive number of new flow requests and then control the activity of such hosts or block their access to the network. Furthermore, these modules aim to maintain operating the forwarding tables of the OpenFlow switches while protecting them from overflow due to the massive number of new flow rules triggered by the attackers. Such protection is achieved by preventing the controller from installing the flow rules of the attacking host while promoting the flow rules that block the traffic of the attacking hosts. This results in reducing the overhead on the communication channel and protecting the controller by limiting the number of requests forwarded to it. The overall architecture of DeMi is shown in Fig. 3 reported here below: The main underlying operating principles of the considered modules are as follows: for the detection module, the entropy concept is considered to measure the randomness in the exchanged network traffic; for the mitigation module, the PoW concept is utilized where the suspicious host needs to spend resources before handling its incoming requests; and, finally, the last underlying principle considered for heavy-load management is the scheduling concept, where the incoming requests are handled based on certain priorities on the controller side. In the following subsections, we discuss the detail of each module and show its role in the proposed solution.

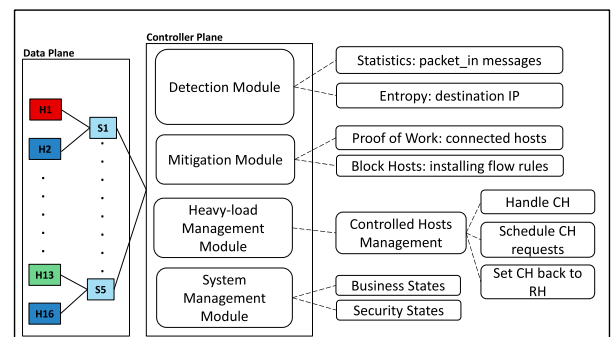


FIGURE 3. DeMi architecture.

### B. SDN CONTROLLER MODULES

DeMi comprises four main modules running in the SDN controller. Each module is responsible for running processes that maintain SDN availability and connectivity. The details of these modules are discussed in the following subsections.

#### 1) DETECTION MODULE

The detection module is responsible for detecting suspicious hosts in the network. It has the primary functionalities of collecting and analyzing statistics about incoming “packet\_in” messages and triggering the mitigation module once needed.

| Considered Fields of Packet Headers in the System |      |         |     |      |         |     |       |         |     |
|---|------|---------|-----|------|---------|-----|-------|---------|-----|
| LAYER 1   |      | LAYER 2 |     |      | LAYER 3 |     |       | LAYER 4 |     |
| IN PORT   | DPID | ETHER   |     |      | IP      |     |       | PORT    |     |
|   |      | src     | dst | type | src     | dst | proto | src     | dst |

FIGURE 4. Extracted packet\_in headers.

The common factor in triggering a DoS attack detection is the abnormalities in the traffic exchanged in the network. For example, in typical situations, a particular pattern in the network traffic is expected to have a certain consumption of network bandwidth or other network resources, such as CPU and memory. A sudden change in such a network pattern or more consumption of resources can be considered abnormal and a possible attack on the network. Based on what is above, in DeMi, the detection procedure considers detecting anomalies in the network pattern via sampling the entropy of occurring events. In detail, such a method measures the randomness of events occurring and the concentration of the network traffic: high entropy values mean high randomness in network traffic and vice-versa. This method is frequently used to detect DoS attacks, and it is considered a valid alternative with respect to signature-based and anomaly-based methods [15]. The destination IP address is the feature related to measuring the randomness in the proposed solution. The detection procedure starts by collecting flow statistics related to the controller's incoming "packet\_in" requests initiated due to the table miss-match processes. The attributes of the "packet\_in" requests are extracted and stored for further processing by the detection module. Such attributes include the incoming packets' source and destination IP addresses, source and destination MAC addresses, port numbers, and the OpenFlow switch ID connected to the controller. Fig.4 below shows the extracted attributes with respect to the different network layers.

Then, within a specific time window determined by receiving a specific number of invalid requests ("packet\_in" messages), the detection module computes the entropy value based on the destination IP address. The entropy value is minimal when the traffic is concentrated around a specific destination in the network. This is because when the incoming packets are directed to the victim host, the number of unique IP addresses considered in that window is reduced. In contrast, the entropy is maximum when the traffic is scattered to several destinations where most destination IP addresses appear at least once in the window in an attack-free situation. As per the entropy, we compute it as shown in the following Eq. 1:

$$H = - \sum_{i=1}^n p_i * \log(p_i) \quad (1)$$

where:

$H$ : entropy value of a particular window containing  $n$  destination addresses with distribution probabilities for the specified window

TABLE 2. Evaluated metrics for violation windows.

| Metric      | Three Windows | Five Windows |
|-------------|---------------|--------------|
| Sensitivity | 90.87%        | 67.46%       |
| Specificity | 92.34%        | 92.28%       |
| Precision   | 97.00%        | 90.01%       |
| Fall_out    | 7.650%        | 7.72%        |
| Accuracy    | 91.27%        | 79.67%       |

- $n$ : the window size. For a window of  $n$  elements, it consists of  $n$  destination IP addresses  
 $p_i$ : distribution of probability for the incoming packet's destination IP address

$p_i$  is evaluated as shown in Eq. 2, where  $x_i$  is the number of incoming packets directed to the IP address destination IP <sub>$i$</sub> :

$$p_i = \frac{x_i}{\sum_{i=1}^n x_i}, \forall i = 1, 2, 3, \dots, n; 0 \leq p_i \leq 1 \quad (2)$$

Once the entropy is evaluated, it is used to monitor flow change in the network. Such monitoring compares the measured entropy value against a predefined initial threshold in an attack-free situation. An entropy value less than the threshold value raises the alarm about a possible DoS attack. This, in turn, triggers other processes of monitoring the source port of the OpenFlow switch from which the packets are coming. The detection of the source port would require the hosts connected through that port of the OpenFlow switch to verify themselves through the mitigation module.

A direct implementation of the above algorithm would yield a relatively high number of false alarms. More specifically, raising the alarm each time the entropy value is below the threshold might increase the number of false positives. Therefore, we decided to raise the alarm only after a specific number of consecutive threshold violations occurred. This means that if the entropy is violated for a successive number of windows, only then the alarm is effectively raised in the network. In order to decide the number of consecutive violations, a set of metrics is evaluated to achieve reliable detection. These metrics are sensitivity, specificity, precision, fall-out, and detection accuracy. We evaluated the considered metrics setting the threshold to either three or five consecutive windows of entropy violations. Table 2 shows the evaluated metrics for the considered windows for an average of 50 runs for a network topology of sixteen hosts, five switches, and one controller.

Additionally, the receiver operating characteristic (ROC) curves are also considered. This is to show the trade-off between the true positive rate (TPR) and false positive rate (FPR) by varying the number of consecutive windows under the attack situation. Based on the experimental measurements, the precision in the case of considering three consecutive windows for entropy violations is 97%, while when considering five windows, it is around 90%.

Additionally, based on Fig. 5 that shows the ROC for both cases, the solution can achieve better performance when considering three consecutive violation windows compared to considering five consecutive windows. Therefore, based

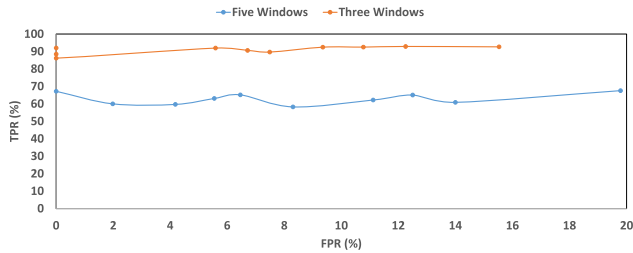


FIGURE 5. Receiver operating characteristic for violation windows.

on the measured metrics and the ROC, considering detection over three consecutive windows leads to better results. Therefore, in the proposed solution, the trigger of the mitigation module is done when the entropy value violates the threshold for three consecutive windows. Each window considered in DeMi is composed of 50 packets. The main reason for choosing 50 packets is to have a reasonable trade-off between a fast computation of the entropy and a large enough sample to be considered representative [36].

Given the variation in traffic volume in a network, setting a static threshold value for entropy violation might cause both false negatives and false positives. This is because the static threshold cannot consider the tendencies and recurring conduct of traffic. Therefore, in the proposed solution, an adaptive threshold adjustment procedure is considered to respond to fluctuations in network traffic. Such a threshold can reflect changes based on traffic volume and aims at minimizing false negatives and positives.

The exponentially weighted moving average (EWMA) [39] is used in the proposed solution to evaluate the adaptive threshold. EWMA is a quantitative measure used to model or describe a time series where the moving average is designed to give older observations lower weights. The weights fall exponentially as the data point gets older. For each sliding window, for an evaluated entropy value that does not violate the threshold, the adaptive threshold estimated at time  $t$  is evaluated as follows:

$$EWMA_t = \alpha * x_t + (1 - \alpha) * EWMA_{(t-1)}, \alpha \in (0, 1) \quad (3)$$

where:

- $\alpha$ : is EWMA factor, higher  $\alpha$  the more closely the EWMA tracks the original time series.
- $x_t$ : is the signal value at time  $t$  (current entropy value).

Therefore, under network traffic changes, the threshold is set adaptively based on the mean value of all experienced traffic while weighing more (exponentially more) recent traffic measurements.

Fig. 6 below shows the operation procedure of the detection module.

## 2) MITIGATION MODULE

The mitigation module is triggered by the detection module once abnormalities in the traffic are detected. The mitigation

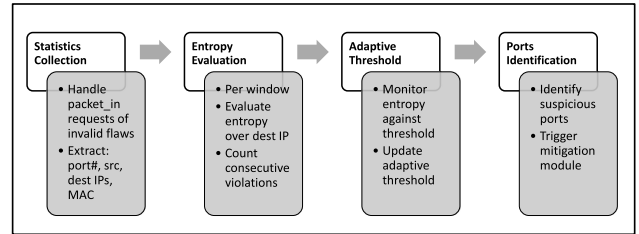


FIGURE 6. Detection module operation procedure.

module is responsible for handling the verification procedure for suspicious hosts through the PoW process [82]. Such a process is based on the concept that, for a client to gain access to a shared resource, it is required to compute some moderately expensive but not intractable pricing function. In contrast, the owner of the shared resource should be able to verify the solution of the pricing function with minimal effort. The pricing function in the proposed solution is shown in Eq. 4 such that, given a challenge  $C$ , find  $X$  such that:

$$LSB_n(SHA_2(C || X)) = 0^n \quad (4)$$

The suspicious host is required to find an input  $X$  to a cryptographic hash function (SHA\_2) that generates a digest with the least significant bits  $LSB_n$  equal to the number of zeros specified by the pricing function. As the cryptographic hash functions are one-way functions, under generally accepted assumptions, the host needs brute force to find the input  $X$ . Such a search process is time-consuming and thus requires a dedicated computation resource from the host. Therefore, to generate massive requests for DoS purposes, an attacker must dedicate a considerable amount of computational resources to find the input  $X$ . On the other hand, the verification process on the controller side is a single hash computed on the provided input to check if it yields a digest with the required number of zeros.

The difficulty of the pricing function is determined by the number of required zeros  $n$ . The initial number of the required zeros is set to 24. Then, the difficulty doubles if the host has been marked as suspicious in the system before or in case the system is in a red security state. Algorithm 1 shows the implementation of the pricing function.

In Algorithm 1, initially, the host receives a challenge  $C$ , a difficulty  $n$ , and a string of length  $l$  as inputs from the controller side. These inputs are fed to the *SOLVE\_CHALLENGE* function at the host. The function needs to generate a random string  $X$  of length  $l$  and is provided together with the challenge  $C$  as inputs to the *GET\_ZEROS* function. Such a function generates a hash digest and returns the count of consecutive zeros in the least significant bits to the function caller (*SOLVE\_CHALLENGE*). The function caller verifies if the returned count matches the required number of zeros ( $n$ ), and if there is no match, the function generates another random  $X$  and repeats the process until a match is found.



**Algorithm 1** Implementation of Pricing Function

**Input:** challenge  $C$ , difficulty  $n$ , string length  $l$   
**Output:**  $X$  with a digest of at least  $n$  zeros in  $LSB$   
**Define:** an empty string  $X$ , an integer variable  $Z$

```

1: function get_Zeros( $C, X$ )
2:   digest  $\leftarrow$  SHA_2( $C||X$ )    ▷ || string concatenation
3:   return count_zeros( $LSB(digest)$ )
4: end function
5: function solve_challenge( $C, n, l$ )
6:    $X \leftarrow NULL, Z \leftarrow 0$ 
7:   while  $Z < n$  do
8:      $X \leftarrow$  random_string( $l$ )    ▷ string of length  $l$ 
9:      $Z \leftarrow$  GET_ZEROS( $C, X$ )
10:  end while
11:  return  $X$ 
12: end function

```

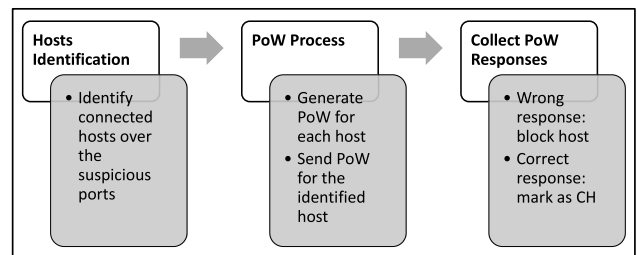
The noteworthy aspects of the considered mitigation procedure are:

- flexibility: the use of the pricing function can be introduced whenever it is desirable to restrain, but not to prohibit, access to a resource (SDN controller),
- selectivity: the legitimate hosts are not affected; only the suspicious hosts are requested to show PoW.
- adjustable difficulty: the level of difficulty can increase based on the host's behavior or the status of the system
- complexity: the use of the hash function guarantees the following:
  - preimage resistance, it is computationally expensive to find the input producing a fixed-length digest;
  - collision resistance, it is hard to find two different inputs that produce the same digest;
  - hash are unbiased functions where the produced digest is unbiased even if the given input is biased,
- ease of verification: the mitigation module can quickly verify the validity of the given solution with minimal overhead; and,
- protection:
  - in case the host is a real attacker, then it needs to spend a considerable amount of computational resources before handling its requests by the controller,
  - even if providing a correct solution to the challenge, a host will be served at a relatively “low” priority compared to other hosts. Additionally, such a host will be suspended if it sends a high number of requests, as will be discussed later in the controlled host section.

Thus, considering such an approach, an attacker must spend a significant amount of computational resources to deliver a large volume of attack traffic that causes the SDN controller to compute routes, eventually making an attack

| Switch ID | Match | Priority | Command | Idle timeout | Hard timeout |
|-----------|-------|----------|---------|--------------|--------------|
|-----------|-------|----------|---------|--------------|--------------|

**FIGURE 7.** Flow Rule configured parameters.



**FIGURE 8.** Mitigation module operation procedure.

prohibitively expensive and, therefore, reducing the overhead on the controller and mitigating the attack.

Providing an invalid solution to the challenge results in blocking the suspicious host and dropping its requests. This is achieved by installing a flow rule by the controller in the OpenFlow switch to block the host from the network. Fig. 7 below shows the configured parameters of the flow rule installed by the controller. Receiving a correct solution to the challenge  $C$  provided to the host redefines the suspicious host in the system as a controlled host. Such hosts are considered genuine but have heavy loads with many incoming “packet\_in” requests compared to the other hosts. Thus, in DeMi, these hosts are controlled by handling and processing their requests based on the level of the workload on the controller, network congestion, security status (red and green—more on this in the following), and the priorities assigned to them.

Hence, DeMi tries to serve different hosts with regular and heavy loads depending on network states. Fig. 8 below shows the overall processes of the mitigation module:

As discussed in the next subsections, controlled hosts have a specific life cycle in the system and specific processes managed by the heavy-load management module, which is described in the following.

### 3) HEAVY-LOAD MANAGEMENT MODULE

The heavy-load management module is triggered when the mitigation module marks a host as a controlled host. Each controlled host has a particular life cycle and associated attributes. The life cycle of such hosts continues until they are promoted as regular hosts again.

The controller has specific activities for the controlled hosts that include: admitting or dropping requests, buffering, scheduling, and serving requests, suspending or unsuspending the controlled hosts, and, finally, setting hosts back to the regular state. The attributes associated with controlled hosts are updated in values during the aforementioned activities.

The main attributes are the time  $t_c$  the host is marked as controlled in the network, the number of initiated “packet\_in” requests of the host  $pkt_{in}$ , and the degree of trust

**TABLE 3.** Attributes of the controlled host.

| Attribute  | Description   |
|------------|---|
| $tc$       | Time the host is marked controlled in the system    |
| $pkt_{in}$ | Number of initiated packet_in requests of the host  |
| $tv$       | Degree of trust of the host in the system           |
| $S$        | This Flag is set to suspend the host                |
| $P$        | This Flag is set to serve the host at high priority |

$tv$  that increases each time the controller admits a request from the host, while it decreases when the controller rejects a request. Another attribute is the suspend/unsuspend flag  $S$ , which is set when the controller suspends a controlled host for a specific amount of time (controlled by a timeout) due to its high value of  $pkt_{in}$ . The last attribute is the priority flag  $P$ , which is set when the host has a higher priority to be served than other controlled hosts. Table 3 below summarizes the main attributes of the controlled host in the proposed solution.

The heavy-load management module handles the requests of controlled hosts with a relatively “lower” priority compared to regular hosts. Such handling depends on the controller’s level of workload and the values assigned to the attributes of the controlled hosts. Initially, the requests of the controlled hosts are served by the controller as long as the controller is not overloaded (in a busy state), meaning that the controller has a regular rate of incoming requests. Once the controller’s state switches to a busy state, the incoming requests are buffered and served at a later stage based on a pre-defined scheduling algorithm.

Each time the host initiates a new request, its  $pkt_{in}$  value is incremented and compared to a dynamic threshold  $thr$ . Such  $thr$  is computed periodically, and it is the average value of initiated requests from all controlled hosts. Having the  $pkt_{in}$  value of the host above  $thr$ , indicates that such a host is more active, with a high number of initiated requests compared to other “controlled” hosts. Thus, when  $pkt_{in} > thr$  results in suspending the host for a given amount of time (time-out). Such a suspension results in dropping the incoming requests for some period and decrementing its  $tv$ . On the other hand, having the host sending requests within the threshold value results in serving or buffering the request and incrementing its  $tv$ .

Once the timeout of the suspended host is reached, that host priority flag is set, indicating a high priority for handling and serving its requests by the controller. Algorithm 2 is used to admit/drop a request of the controlled host. Handling the controlled host’s requests depends on the level of the workload on the controller. Such states are *busy* and *regular*. If in the busy state, the incoming requests of the controlled hosts are buffered, while if in the regular state, the controller serves the incoming or buffered requests. Requests are served in rounds. On a given round, the controller serves some buffered requests. These requests are selected based on an algorithm that considers the attributes of the controlled hosts. Such attributes include the  $P$  and  $S$  flags,  $tv$ , and whether the host is served at the current round. If the algorithm assigns

**Algorithm 2** Handling Requests of Controlled Hosts

**Input:** threshold  $thr$ , host request  $rqst_i$ , trust value  $tv_i$

```

1: function handle_request( $rqst_i$ )
2:   if  $pkt_{in_i} > thr$  then
3:     discard  $rqst_i$ 
4:     set  $S_i = \text{TRUE}$ 
5:   else
6:     if Controller State == “BUSY” then
7:       buffer  $rqst_i$ 
8:     else
9:       serve  $rqst_i$ 
10:    end if
11:   end if
12: end function
13: function update_trust_value( $tv_i$ )
14:   if  $S_i == \text{TRUE}$  then
15:      $tv_i = tv_i - 1$ 
16:   else
17:      $tv_i = 0.5 tv_i + 1$ 
18:   end if
19: end function
20: function update_host_state( $S_i, P_i$ )
21:   if timeout reached for host then
22:     set  $S_i = \text{FALSE}$ 
23:     set  $P_i = \text{TRUE}$ 
24:   end if
25: end function

```

**TABLE 4.** Serving controlled hosts priorities.

| Highest Priority | Flags        | Criteria                                     |
|------------------|--------------|--|
| 1                | P-NS         | priority flag set, not served                |
| 2                | US-TV-NS     | unsuspended, highest trust value, not served |
| 3                | US-TV-SR     | unsuspended, highest trust value, served     |
| 4                | P-SR         | priority flag set, served                    |
| 5                | S-NS or S-SR | suspended-not_served, or suspended-served    |

the hosts a  $P$  flag, the host gains the highest priority, while an  $S$  flag gives the host the lowest priority. In case both flags  $P$  and  $S$  are not set for a host. Then, the host is served according to its  $tv$  value. The higher the  $tv$  value, the higher the priority of the host. Having two hosts with the same value of the  $tv$ , the host with a longer buffer queue is served first.

To have a fair share of the serving queue among the connected hosts in the controller, one request from each host is served per round regardless of the priority assigned. When all cases are covered based on the priority list, some hosts may have more served requests in the same round.

Table 4 shows the priorities considered in selecting the request with respect to the set flags, where “1” is the highest priority.

Each controlled host should eventually be marked as a regular host in the network. This is achieved based on a selection process that employs the weighted sum model (WSM) with beneficial and non-beneficial attributes. Beneficial attributes

**TABLE 5. WSM beneficial and non-beneficial attributes.**

| Attribute                | Type           | Weight |
|--------------------------|----------------|--------|
| Trust Value              | Beneficial     | 0.3    |
| CH Lifetime              | Beneficial     | 0.1    |
| Incoming Requests Rate   | Non-beneficial | 0.4    |
| Current Buffered Packets | Non-beneficial | 0.2    |

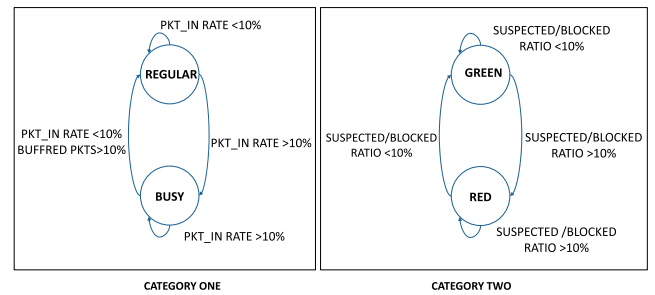
are trust value and time passed since the host was marked as a controlled host (CH lifetime), while non-beneficial attributes are the rate of incoming requests and the number of remaining buffered requests of the controlled host. Additionally, the host should match two primary conditions before being involved in the promotion process. First, it should spend the minimum time required for a host to be controlled in the network. This period of time is variable, based on the controller's level of workload. Second, the trust value of the host should not be zero. On a periodical basis, the controlled host with the highest score based on the WSM is moved by the system to a queue to be released. Based on a round-robin basis, the system passes over the selected hosts in the queue and promotes them to regular hosts when the conditions are met. The conditions are satisfied when the controlled host has zero remaining buffered requests to be served in the network. Table 5 shows the beneficial and non-beneficial attributes considered and their weights in the WSM. The weights assigned to the attributes are set to focus the control on the hosts with high incoming request rates and with high number of buffered packets. Instead, the hosts with high trust values are more eligible to be marked as regular hosts.

#### 4) SYSTEM MANAGEMENT MODULE

The System Management Module (SMM) is the module responsible for setting the controller's states. These states fall under two main categories: the first is related to the level of workload of the controller, while the second is related to the security state of the system. The level of the workload of the controller is represented using two states: regular and busy. Initially, the system and, more specifically, the controller is set to a regular state. During such a state, the controller serves all the incoming requests, including controlled hosts' requests.

As the system management module keeps track of the rate of incoming requests to the controller and the buffer size, the state can be changed to busy. More specifically, such a state is set when the controller's buffer for the incoming requests' handling and processing shows an increase of 10% with respect to the incoming packets rate.

Switching the controller's state to busy results in buffering the incoming "packet\_in" requests of the controlled hosts and, in some cases, suspending them. Hence, higher serviceability to the regular than to the controlled hosts is provided. Another effect of the busy state is related to increasing the minimum time the controlled host needs to spend before being marked again as a regular host. Therefore, controlled

**FIGURE 9. Mealy machine for controller states.**

hosts will stay controlled for a longer time when the controller is in a busy state.

As the SMM tracks the controller activity and the OpenFlow switches, it sets the controller state to regular when needed. The module tracks the rate of the incoming packets to the controller and the number of buffered packets of the controlled hosts. Having the incoming rate of the packets less than 10% and the ratio of the buffered packets of the controlled hosts set to more than 10% with respect to the incoming requests triggers the module to switch the system state to regular. Such switching is needed to balance serving regular and controlled hosts as much as possible.

The second category of the system states is related to security, which can be in two states: "red" or "green", where each state triggers specific activities. The system is considered to be in a red state based on the number of active hosts in the system marked either as *suspected* or *blocked*. The suspected hosts are the hosts that have received a PoW challenge. The blocked hosts are the hosts that are marked as suspicious and that are blocked from sending their requests to the controller. A ratio of suspected and blocked hosts above 10% of the total active hosts in the system results in switching the system to a red state. The effect of such a state requires each newly connected host to satisfy a PoW before obtaining a connection to the network. Another effect is providing more complex PoW (i.e. increasing the number of zeroes as per what is in Eq. 4). Fig. 9 shows the Mealy machine for the controller's states.

## V. SIMULATION AND PERFORMANCE EVALUATION

DeMi is evaluated against multiple performance metrics. This section discusses the evaluation steps, including the setup required for the experiments, simulation environment, topology, and experimental results against different metrics.

### A. SIMULATION ENVIRONMENT AND TOPOLOGY

Mininet [83] is commonly utilized as an SDN network emulator [13], [20], [33]. In particular, it can emulate a complete network of end-hosts, links, and switches on a single Linux kernel using process-based virtualization. It enables the creation of realistic virtual environment scenarios making it convenient for experimenting with SDN solutions.

**TABLE 6.** Simulation parameters.

| Parameter                             | Value |
|---------------------------------------|-------|
| Network Depth                         | 2     |
| Number of OpenFlow Switches           | 5     |
| Number of Hosts                       | 16    |
| Number of Attacking Hosts             | 1     |
| Number of Targeted Victim Hosts       | 1     |
| Average Number of Runs Per Experiment | 50    |

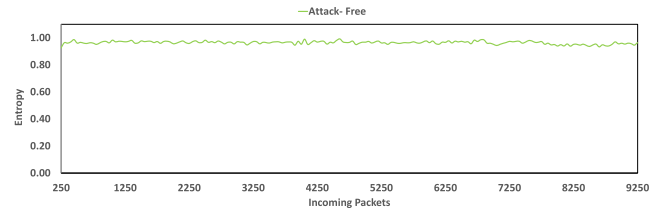
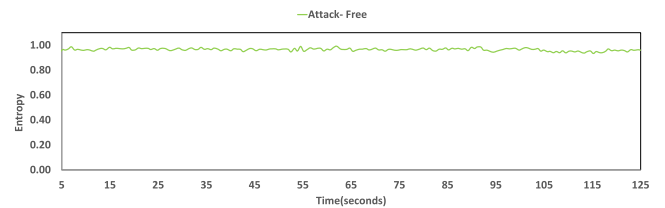
In our simulation scenarios, the Mininet environment is integrated into a test environment running on a PC with 8 GB RAM and Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz processor. For the network switches, Open Virtual Switches (OVS) [84] are used, which are virtual switches that enable network automation through programmatic extension. These switches are connected with POX [85], which is a python-based controller. The L3\_learning module of the POX controller is used for connecting the controller to the network created. Scapy [86], a tool used for computer networks to generate and manipulate packets, is used in simulation scenarios to generate user datagram protocol (UDP) packets for regular and attack traffic. Using the Scapy tool, spoofed IP source packets are generated and addressed to one host. According to [17], the type of traffic, whether UDP, TCP, or ICMP, does not affect the solution as long as we look for invalid header fields in the incoming packet that make the OpenFlow switch generate a “packet\_in” message request due to tablemiss procedure.

Using Mininet, we simulated a medium-sized topology of a tree-type network consisting of depth two with five OpenFlow switches and sixteen hosts is built. The considered topology is the one already described by Fig. 2. The hosts are connected to the controller remotely via the OpenFlow switch through the IP address and the corresponding port of the POX controller in the simulation system. Table 6 shows the setup parameters and their values used to create attack scenarios.

We considered such a small/medium size topology following similar solutions described in other works, such as [20], [24], and [34], to cite a few. Considering more than one attacking/victim host would require modifying the solution to consider protecting a subnet instead of a single victim, as discussed later in our future work.

### 1) ATTACK SCENARIO

The attacking scenario is the following: one of the connected hosts is controlled by the attacker and then is used to generate a DoS attack targeting another host (victim). The legitimate traffic is generated by another host, neither the victim nor the attacker. Moreover, the legitimate host communicates with the controller benignly. Although the created topology is not a large-scale topology, the attacker can generate malicious packet streams that seem to come from different IP addresses. Though the topology is not a large one, the insights that will be gathered will be useful for understanding how DoS works

**FIGURE 10.** Evaluated entropy versus number of packets in an attack-free scenario.**FIGURE 11.** Evaluated entropy over time in attack-free scenario.

on small/medium size networks, and they will be the basis for future work in this direction.

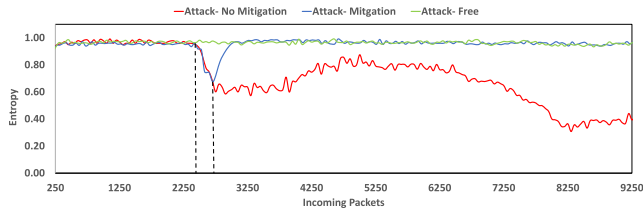
### B. INITIAL ENTROPY EVALUATION UNDER ATTACK-FREE SITUATION

We will start by evaluating the entropy value, the primary metric for the initial attack detection process. Initially, the entropy is evaluated under an attack-free situation where the network generates regular traffic using the Scapy tool. In such a situation, hosts exchange packets with the same probability. This means that there is no concentration in the traffic around a specific destination. Fig. 10 shows the relation between the evaluated entropy and the corresponding number of incoming packets to the controller with the traffic generation interval set to 0.02 seconds. Such an interval means that during each 0.02 seconds time interval, a set of packets are generated and sent to the controller from the legitimate host.

The entropy shown in the figure is after the system is set and the exchange of the address resolution protocol (ARP) requests and replies is over—these activities take place at network setup. As the regular traffic is generated and exchanged over the network with a uniform distribution, the entropy in destination IP addresses increases until it reaches its maximum value (one). The fluctuation of the entropy value around the value one is due to the small number of hosts (sixteen hosts). With a larger network size, more diversity is expected in the destination IP addresses and, therefore, a higher entropy value. The same pattern of the entropy behavior can be seen in Fig. 11, showing the evaluated entropy value and the corresponding time.

The evaluation of the entropy value during the attack-free situation is considered in the proposed approach to set the initial threshold value of the attack detection. This is similar to many IDS approaches in the literature [20], [24], and [34]. Considering an error value of the evaluated entropy in the attack-free situation set to 10%, the first threshold set for the





**FIGURE 12.** Evaluated entropy versus number of packets under attack situation.

attack detection is equal to 0.9. Then, that threshold value is updated based on the adaptive approach using the EWMA approach discussed earlier to reflect the different changes in the network traffic.

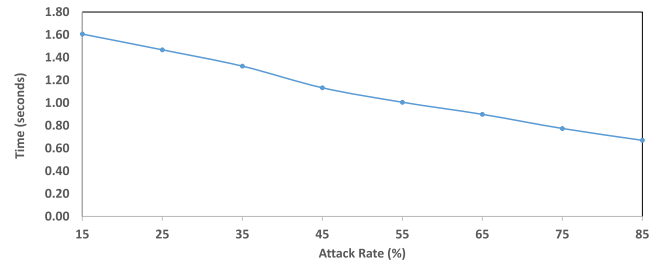
### C. ENTROPY EVALUATION IN ATTACK SCENARIO

Fig. 12 shows the entropy evaluation versus the number of incoming “packet\_in” requests with and without DeMi under the attack scenario. The cited figure shows the measurements after the system is set, where the exchange of ARP requests and replies between the hosts during the initial network setup is over.

In this experiment, the regular traffic generation interval is set to 0.02 seconds, while the attacking traffic generation interval is set to 0.005 seconds. This means 25% of the traffic is malicious. Using the Scapy tool, the source IP of the generated attacking traffic is spoofed and addressed to the victim host.

When a large packet volume arrives at a specific host, the system’s randomness decreases rapidly. Therefore, the value of the entropy goes below the threshold, reaching the lowest value. This is because all the spoofed source IP packets have the victim’s IP address in their destination IP—hence, there is less variability in the range of destination addresses. This inverse relationship between the entropy and the IP addresses variety, and the associated entropy is captured by the fact that, when the attack traffic rate increases, the entropy value decreases. In the shown figure, the entropy value drops when around 2600 packets have been received by the controller. When DeMi is applied, the entropy value starts to enhance (increase) within around 240 packets after the initial drop (at around 2840 packets, as in Fig. 12). Such enhancement increases gradually until it reaches the maximum value, reinstating the system to a healthy state. More specifically, getting the maximum entropy value shows that the traffic is distributed and not directed to a specific host. Hence, demonstrating the system’s ability to eliminate malicious traffic.

It is worth noting what would happen to the entropy value without considering DeMi. After an initial drop in the entropy value, some slight enhancements can be seen. Such enhancements are due to a partial failure of the controller because of the high traffic volume. The controller fails because of the high overload, where such failure results in dropping some of the incoming malicious requests or disconnecting the OpenFlow switch to which the attacker is connected to. Other



**FIGURE 13.** Attack detection time versus attack rate.

**TABLE 7.** Attack generation interval and corresponding attacking percentage.

| Generation Interval | Attacking Traffic Percentage |
|---------------------|------------------------------|
| 0.0033              | 15%                          |
| 0.0050              | 25%                          |
| 0.0070              | 35%                          |
| 0.0090              | 45%                          |
| 0.0110              | 55%                          |
| 0.0130              | 65%                          |
| 0.0150              | 75%                          |
| 0.0170              | 85%                          |

reasons may include the failure of the OpenFlow switch itself due to the overload on its tables; and the installed flow rules by the controller on the OpenFlow switches, which could reduce the number of incoming “packet\_in” requests of the attacker to the controller. Then, as the attacking traffic is still directed to the controller, the entropy value drops gradually until the attack period is over. Eventually, the system improves as the system receives genuine packets when the attack is over.

### D. ATTACK DETECTION AGAINST INTENSITY OF THE ATTACK

To measure the DeMi’s ability to detect attacks when these ones are carried out at different attack intensities, we have developed a set of experiments. Fig. 13 shows the relation between the attack intensity versus the time needed to detect the attack. In this experiment, different packet generation intervals for the attacking traffic are set, and the detection time is measured. The regular traffic generation interval is fixed at 0.02 seconds. Table 7 shows the attacking intervals versus the percentages of the attacking traffic with respect to the total traffic. Setting larger intervals results in a higher number of attacking packet generation.

As it can be seen from the figure, higher attacking rates result in shorter detection times. This is due to the fast drop in the entropy value within a short period of time.

### E. VERIFICATION OF RELIABLE CONNECTION UNDER ATTACK SCENARIO

In another set of experiments, we have measured the solution’s ability to provide reliable connections when the network is under attack. For this purpose, Wireshark, a packet analyzer application that examines the exchanged packets over the network, is used in the following analysis to

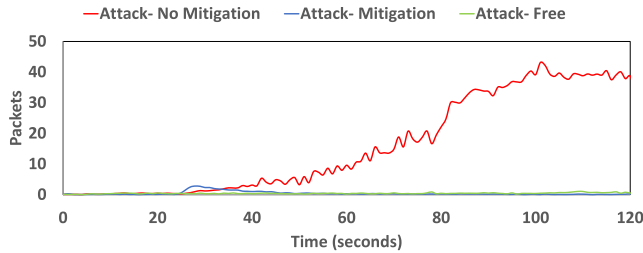


FIGURE 14. Exchanged TCP packets over time.

look closely at the network's behavior. In this analysis, the exchanged TCP packets marked for re-transmissions are filtered-out using Wireshark. Such a re-transmission process is applied to resend the packets lost or damaged due to network congestion or partial failure.

Fig. 14 shows the exchanged TCP packets marked for re-transmission in attack-free and under-attack scenarios for some time period. Initially, the pattern of the exchanged packets is similar for the case of attack-free and attack scenarios. Then, as the attacker starts to generate massive "packet\_in" requests, at around  $t = 25$  seconds, there is an increase in the number of packets marked for re-transmission. Such a re-transmission process indicates packet loss or damage in the network, increasing latency and lowering the speed of requests.

With DeMi in place, the number of the re-transmitted packets drops after an initial increase until the network load is close to the attack-free scenario. More specifically, when DeMi is activated, the total number of re-transmitted packets is roughly the same as the attack-free scenario. These results show the quality of DeMi in thwarting attacks played at different packet injection rates. On the contrary, without considering DeMi, the TCP re-transmissions increase gradually as the attacking traffic is exchanged over the network. Such an increase is about 3,7 times higher with reference to the attack-free scenario. Such a high number of control messages shows the network's inability to successfully exchange packets among the network hosts, OpenFlow switches, and the controller.

#### F. EXCHANGED CONTROL PACKETS UNDER ATTACK SCENARIO

Another set of experiments was designed and carried out to understand the flow of the exchanged network control packets. Some of these packets are exchanged during the initial network setup between the OpenFlow switches and the controller, while other packets are exchanged periodically or to raise an alarm about some events during the network operation. Examples of the control packets exchanged at the network-setup are hello requests and replies, OpenFlow features requests and replies, and configuration packets. These packets are exchanged between the OpenFlow switches and controller to set up the communication channels, negotiate over the protocol version to be used, and communicate the

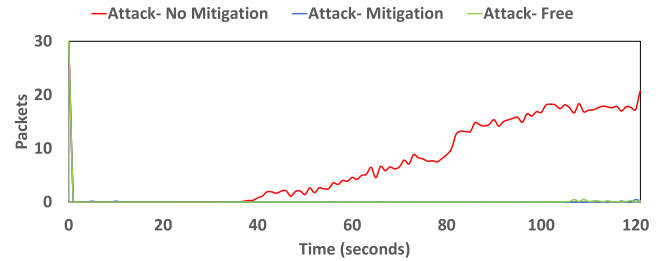


FIGURE 15. Exchanged control packets over time.

features of the connected OpenFlow switches. The other category of the exchanged control packets includes OpenFlow error messages and echo requests and replies. The OpenFlow error messages are exchanged to notify the OpenFlow switches or the controller about communication problems in the network. In contrast, the echo requests and replies are used to test the OpenFlow connection when a connection is hung. These packets are exchanged periodically to verify proper connectivity over the OpenFlow protocol. Fig. 15 shows the exchanged control packets under attack-free and attack scenarios. As it can be seen, the trend is similar for both attack-free and attack scenarios when DeMi is adopted. The two lines (green and blue) overlap in Fig. 15, showing the effectiveness of deploying DeMi. However, it is worth noting that, in an attack scenario where DeMi is not deployed, the number of exchanged packets increases gradually and it is roughly 2,7 times more compared to the case when DeMi is not deployed. This shows the network resources' exhaustion, affecting the network's availability. Indeed, as it can be expected, having a higher attacking rate may result in a higher number of exchanged control packets over the network due to link congestion and the inability of the controller to handle the requests.

Two sample runs were considered to show the distribution of control packets in an attack scenario. One where DeMi is deployed and the other one where it is not. The packets' distribution and corresponding rates are shown in Table 8. In the sample runs, when DeMi is not in place, a total of 1452 packets are exchanged,  $\approx 6.5\%$  of the total OpenFlow packets captured by Wireshark. When DeMi is deployed, just 30 control packets are exchanged, that is  $\approx 0.14\%$  of the total exchanged OpenFlow packets.

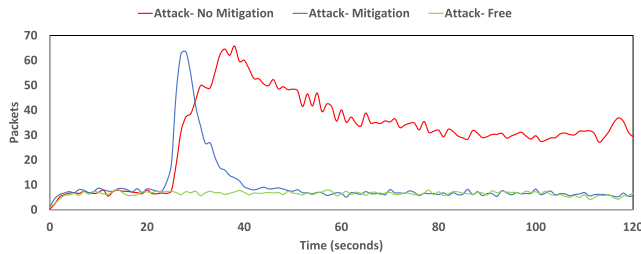
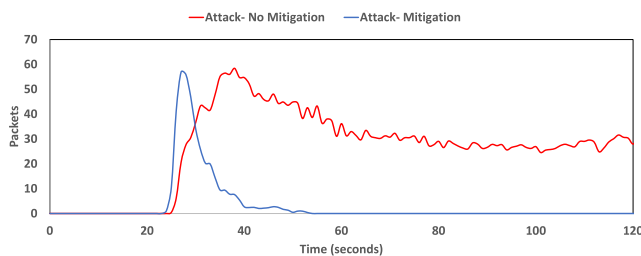
#### G. SECURING THE VICTIM IN ATTACK SCENARIO

We also analyzed the ability of DeMi to secure the victim host from the attacking packets. For this analysis, all the victim's successfully received packets were considered. The measures in Fig. 16 consider all the sources targeting the victim host, including the legitimate and attacker hosts. Instead, Fig. 17 shows only the packets generated by the attacker host and addressed to the victim host. These packets were filtered out using the Wireshark analyzer while considering the destination IP address of the victim host.

In Fig. 16, all the packets addressed to the victim for both attack-free and attack scenarios are reported from both legit

**TABLE 8.** Distribution of exchanged control packets for sample runs.

| Packet Type           | Without DeMi  |            | With DeMi     |            |
|-----------------------|---------------|------------|---------------|------------|
|                       | Total Packets | Percentage | Total Packets | Percentage |
| OFPT_HELLO            | 14 packets    | ~0.92%     | 10 packets    | ~0.33%     |
| OFPT_ERROR            | 1452 packets  | ~95.7%     | 0 packets     | 0%         |
| OFPT_ECHO_REQUEST     | 18 packets    | ~1.18%     | 0 packets     | 0%         |
| OFPT_ECHO_REPLY       | 13 packets    | ~0.85%     | 0 packets     | 0%         |
| OFPT_FEATURES_REQUEST | 5 packets     | ~0.33%     | 5 packets     | ~16.6%     |
| OFPT_FEATURES_REPLY   | 5 packets     | ~0.33%     | 5 packets     | ~16.6%     |
| OFPT_SET_CONFIG       | 5 packets     | ~0.33%     | 5 packets     | ~16.6%     |
| OFPT_BARRIER_REPLY    | 5 packets     | ~0.33%     | 5 packets     | ~16.6%     |

**FIGURE 16.** Incoming packets to victim from all sources over time.**FIGURE 17.** Incoming packets to victim from attacker over time.

and illegit sources. Initially, the trends of the received traffic on the victim host are similar for the two considered scenarios—the attacker has not started sending malicious packets yet. Then, as the attacker starts sending spoofed packets at around  $t = 25$  seconds, the number of packets received by the victim increases.

The cited figure shows how DeMi can protect the victim host by reducing the traffic directed at it. Indeed, the victim continues receiving traffic from legitimate hosts generating regular traffic or from the controller. This indicates that DeMi does not entirely stop the incoming traffic to the victim host but eliminates the attacking traffic, resulting in a similar trend of an attack-free scenario.

However, when DeMi is not deployed, the incoming traffic to the victim increases gradually until it starts to drop and stabilize. That drop is due to the overload in the network that results in partial failure of the controller or the OpenFlow switch to which the attacker is connected—hence limiting its spoofed packets injection capability. Fig. 17 sheds more light on how the victim host is protected from the attacker's traffic. The cited figure shows the transmitted packets from the attacker to the victim host over time. Using Wireshark,

we selected only the packets directed from the source to the destination, discarding the control packets associated with the transmitting process. As seen in the figure, DeMi was able to thwart the attack on the victim host within a relatively short time (at  $t = 27$ ), showing the ability to protect the victim host.

It should be mentioned that deploying DeMi introduces a transient start-up phase that requires more communications due to flow rules installation and attack mitigation processes. Therefore, at the initial stages of the attack mitigation, it can be seen that there is an increase in the incoming packets to the victim host at the beginning of the attack detection—before eliminating the attacking packets later on.

#### H. SDN CONTROLLER AVAILABILITY IN ATTACK SCENARIOS

In order to measure the availability of the POX controller in an attack scenario, we introduce some specialized metrics. These metrics are related to the hello requests, the number of disconnections and the number of OpenFlow errors exchanged over the network operating time. The exchange of the hello requests and replies is used when the OpenFlow switch joins or rejoins the network and connects to the controller.

As shown in Table 9, in an attack scenario where DeMi is not deployed, the rate of exchanged hello requests is higher when compared to a scenario where DeMi is running. Such an increase is due to the inability of the OpenFlow switches to connect to the controller under a heavy attacking load. Once these switches cannot connect to the controller, they initiate hello requests directed to the controller and wait for replays.

Depending on the controller's processing and load handling, it responds to the hello requests to keep the connection with the OpenFlow switches. However, not receiving the response from the controller within a specific time results in losing the connection. In turn, the loss of connection between the controller and the OpenFlow switch results in a socket disconnection issue. In such an issue, all the flow rules previously installed by the controller into the tables of the OpenFlow switches are cleared, as well as the buffered requests in the switches waiting for routing rules from the controller. Such a process results in an additional overhead on the controller to re-install the flow rules again and could cause OpenFlow errors on the controller side.

TABLE 9. Measures of network availability in attack and attack-free situations.

| Type                  | Hello Requests/sec | Socket Disconnections/sec | OpenFlow Errors/sec |
|-----------------------|--------------------|---------------------------|---------------------|
| Attack-free           | 0.071              | 0                         | 0                   |
| Under Attack-DeMi     | 0.078              | 0.01                      | 0                   |
| Under Attack- No DeMi | 0.086              | 0.07                      | 13.88               |

TABLE 10. Comparison of enhancements between selected works and DeMi.

| Work | Entropy Enhancement | Victim Protection | Response Time Reduction | Control Packets Reduction |
|------|---------------------|-------------------|-------------------------|---------------------------|
| [13] |                     |                   | ✓                       | ✓                         |
| [20] | ✓                   |                   |                         |                           |
| [21] | ✓                   | ✓                 |                         |                           |
| [23] | ✓                   |                   |                         |                           |
| [24] | ✓                   |                   |                         |                           |
| [25] | ✓                   |                   |                         |                           |
| [29] | ✓                   |                   |                         |                           |
| [32] | ✓                   |                   |                         |                           |
| [34] | ✓                   |                   | ✓                       |                           |
| [38] | ✓                   | ✓                 |                         |                           |
| [40] | ✓                   |                   | ✓                       |                           |
| [51] |                     | ✓                 | ✓                       |                           |
| [61] |                     | ✓                 | ✓                       |                           |
| [62] |                     | ✓                 | ✓                       |                           |
| [63] |                     | ✓                 | ✓                       |                           |
| [64] |                     | ✓                 | ✓                       |                           |
| [68] |                     | ✓                 | ✓                       |                           |
| [74] |                     | ✓                 | ✓                       |                           |
| [76] | ✓                   | ✓                 |                         | ✓                         |
| DeMi | ✓                   | ✓                 | ✓                       | ✓                         |

The OpenFlow errors occur mainly when the controller attempts to install a flow rule into the OpenFlow switch for an already received buffered packet. Such a buffered packet is a copy of the packet sent to the controller through the “packet\_in” request and referenced by a specific buffer ID. Such buffering reduces the overhead by eliminating the need to send another copy of the same packet from the controller to the OpenFlow switch during the flow rule installation process to route that packet—thus, improving the latency and reducing the overhead over the communication channel. However, due to the heavy load on the controller side and the OpenFlow switch, and adding to that the loss of the connection between the OpenFlow switch and the controller, such a packet (referenced by the buffer ID) may be expired or no longer exists in the OpenFlow switch. Therefore, when the controller tries to install the flow rule, it obtains an error since there is no valid buffer ID at the OpenFlow switch side.

Table 9 summarizes the average number of hello requests, socket disconnections, and OpenFlow errors under attack and attack-free scenarios with and without DeMi. The shown data are for an attacking rate of 25%—a higher attacking rate would result in more disturbance of the network availability.

Finally, to show some of the enhancements that DeMi introduces as a solution for DoS attack detection and mitigation, we highlight some of the related works in the literature that have performance metrics and methodologies

similar to DeMi. Table 10 shows that DeMi is able to provide most of the enhancements that other works introduced.

## VI. CONCLUSION

In this paper, we proposed DeMi, a solution that addresses DoS attacks on SDNs. DeMi. The main stages implemented by DeMi are attack detection, mitigation, and heavy-load management. *Detection* is achieved using a sample entropy approach combined with an adaptive threshold mechanism based on the EWMA. *Mitigation* is enforced using the PoW approach and installing flow rules from the controller into the OpenFlow switches. *Heavy-load management* is implemented through prioritized scheduling. The simulation results of DeMi show an enhanced performance of the POX controller under the DoS attack. Additionally, DeMi effectively reduces both the number of packets that need to be re-transmitted (via TCP protocol) and the exchanged control packets in the network within a relatively short time since the start of the attack—basically restoring the network condition of an attack free scenario. Results are staggering: the number of TCP packets marked for re-transmissions when DeMi is not in place is 3,7 times higher with respect to when DeMi is deployed. As per exchanged control packets, an attack scenario where DeMi is not deployed requires handling about 2,7 more control packets. Finally, DeMi can protect the victim from malicious traffic while not blocking legitimate traffic.



Indeed, as shown, DeMi performance is roughly equal to the ones of the attack-free scenario.

As future work, we would like to investigate how we can enhance our attack detection approach by considering the joint-entropy approach, where more than one class of entropy class can be considered. Additionally, we would like to extend the adversary model to consider an attack targeting a whole subnet instead of a victim host. Finally, we would like to extend the model to tackle DDoS attacks in SDNs where more than one attacker can perform the attack simultaneously on the network.

The clear architectural design, the detailed detection and mitigation solutions, and the shown experimental results, other than being interesting on their own, also pave the way for further research in the domain.

## ACKNOWLEDGMENT

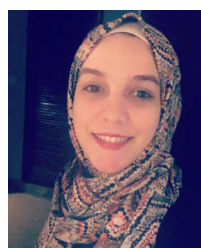
Dr. Roberto Di Pietro produced part of his contribution while at HBKU-CSE. The information and views set out in this publication are those of the authors and do not necessarily reflect the official opinion of HBKU.

## REFERENCES

- [1] M. M. Isa and L. Mhamdi, "An adaptive framework for attack mitigation in SDN environment," in *Proc. IEEE Int. Medit. Conf. Commun. Netw. (MeditCom)*, Sep. 2022, pp. 130–135.
- [2] W. Rafique, L. Qi, I. Yaqoob, M. Imran, R. U. Rasool, and W. Dou, "Complementing IoT services through software defined networking and edge computing: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1761–1804, 3rd Quart., 2020.
- [3] D. Gao, Z. Liu, Y. Liu, C. H. Foh, T. Zhi, and H.-C. Chao, "Defending against packet-in messages flooding attack under SDN context," *Soft Comput.*, vol. 22, no. 20, pp. 6797–6809, Oct. 2018.
- [4] M. Yue, H. Wang, L. Liu, and Z. Wu, "Detecting DoS attacks based on multi-features in SDN," *IEEE Access*, vol. 8, pp. 104688–104700, 2020.
- [5] S. Siddiqui, S. Hameed, S. A. Shah, I. Ahmad, A. Aneiba, D. Draheim, and S. Dustdar, "Toward software-defined networking-based IoT frameworks: A systematic literature review, taxonomy, open challenges and prospects," *IEEE Access*, vol. 10, pp. 70850–70901, 2022.
- [6] M. Khalid, S. Hameed, A. Qadir, S. A. Shah, and D. Draheim, "Towards SDN-based smart contract solution for IoT access control," *Comput. Commun.*, vol. 198, pp. 1–31, Jan. 2023.
- [7] A. Sallam, A. Refaey, and A. Shami, "On the security of SDN: A completed secure and scalable framework using the software-defined perimeter," *IEEE Access*, vol. 7, pp. 146577–146587, 2019.
- [8] R. Xie, M. Xu, J. Cao, and Q. Li, "SoftGuard: Defend against the low-rate TCP attack in SDN," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–6.
- [9] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, Jul. 2013.
- [10] L. F. Eliyan and R. Di Pietro, "DoS and DDoS attacks in software defined networks: A survey of existing solutions and research challenges," *Future Gener. Comput. Syst.*, vol. 122, pp. 149–171, Sep. 2021.
- [11] S. Gao, Z. Peng, B. Xiao, A. Hu, Y. Song, and K. Ren, "Detection and mitigation of DoS attacks in software defined networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1419–1433, Jun. 2020.
- [12] R. N. Carvalho, J. L. Bordim, and E. A. P. Alchieri, "Entropy-based DoS attack identification in SDN," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2019, pp. 627–634.
- [13] M. Imran, M. H. Durad, F. A. Khan, and H. Abbas, "DAISY: A detection and mitigation system against denial-of-service attacks in software-defined networks," *IEEE Syst. J.*, vol. 14, no. 2, pp. 1933–1944, Jun. 2020.
- [14] Z. Liu, Y. He, W. Wang, and B. Zhang, "DDoS attack detection scheme based on entropy and PSO-BP neural network in SDN," *China Commun.*, vol. 16, no. 7, pp. 144–155, Jul. 2019.
- [15] J. David and C. Thomas, "Detection of distributed denial of service attacks based on information theoretic approach in time series models," *J. Inf. Secur. Appl.*, vol. 55, Dec. 2020, Art. no. 102621.
- [16] M. A. Aladaileh, M. Anbar, I. H. Hasbullah, Y.-W. Chong, and Y. K. Sanjalawe, "Detection techniques of distributed denial of service attacks on software-defined networking controller—A review," *IEEE Access*, vol. 8, pp. 143985–143995, 2020.
- [17] S. M. Mousavi and M. St-Hilaire, "Early detection of DDoS attacks against SDN controllers," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Feb. 2015, pp. 77–81.
- [18] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Jul. 1948.
- [19] R. Wang, Z. Jia, and L. Ju, "An entropy-based distributed DDoS detection mechanism in software-defined networking," in *Proc. 14th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Aug. 2015, pp. 310–317.
- [20] R. Swami, M. Dave, and V. Ranga, "Defending DDoS against software defined networks using entropy," in *Proc. 4th Int. Conf. Internet Things, Smart Innov. Usages (IoT-SIU)*, Apr. 2019, pp. 1–5.
- [21] J. Galeano-Brajones, D. Cortés-Polo, J. F. Valenzuela-Valdés, A. M. Mora, and J. Carmona-Murillo, "Detection and mitigation of DoS attacks in SDN. An experimental approach," in *Proc. 6th Int. Conf. Internet Things, Syst., Manag. Secur. (IOTSMS)*, Oct. 2019, pp. 575–580.
- [22] J. Singh and S. Behal, "A novel approach for the detection of DDoS attacks in SDN using information theory metric," in *Proc. 8th Int. Conf. Comput. Sustain. Global Develop. (INDIACom)*, Mar. 2021, pp. 512–516.
- [23] L. Zhou, K. Sood, and Y. Xiang, "ERM: An accurate approach to detect DDoS attacks using entropy rate measurement," *IEEE Commun. Lett.*, vol. 23, no. 10, pp. 1700–1703, Oct. 2019.
- [24] A. Ahalawat, S. S. Dash, A. Panda, and K. S. Babu, "Entropy based DDoS detection and mitigation in OpenFlow enabled SDN," in *Proc. Int. Conf. Vis. Towards Emerg. Trends Commun. Netw. (ViTECoN)*, Mar. 2019, pp. 1–5.
- [25] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Comput. Netw.*, vol. 62, pp. 122–136, Apr. 2014.
- [26] K. S. Sahoo, M. Tiwary, and B. Sahoo, "Detection of high rate DDoS attack from flash events using information metrics in software defined networks," in *Proc. 10th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2018, pp. 421–424.
- [27] Y. Xiang, K. Li, and W. Zhou, "Low-rate DDoS attacks detection and traceback by using new information metrics," *IEEE Trans. Inf. Forensics Security*, vol. 6, no. 2, pp. 426–437, Jun. 2011.
- [28] S. Behal and K. Kumar, "Detection of DDoS attacks and flash events using novel information theory metrics," *Comput. Netw.*, vol. 116, pp. 96–110, Apr. 2017.
- [29] R. Li and B. Wu, "Early detection of DDoS based on  $\phi$ -entropy in SDN networks," in *Proc. IEEE 4th Inf. Technol., Netw., Electron. Autom. Control Conf. (ITNEC)*, vol. 1, Jun. 2020, pp. 731–735.
- [30] A. Rényi, "On measures of information and entropy," in *Proc. 4th Berkeley Symp. Math., Statist. Probab.*, vol. 1, 1960, pp. 547–561.
- [31] P. K. Bhatia and S. Singh, "On a new Csiszar's f-divergence measure," *Cybern. Inf. Technol.*, vol. 13, no. 2, pp. 43–57, 2013.
- [32] J. Mao, W. Deng, and F. Shen, "DDoS flooding attack detection based on joint-entropy with multiple traffic features," in *Proc. 17th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./12th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2018, pp. 237–243.
- [33] K. Kalkan, L. Altay, G. Gür, and F. Alagöz, "JESS: Joint entropy-based DDoS defense scheme in SDN," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2358–2372, Oct. 2018.
- [34] P. Kumar, M. Tripathi, A. Nehra, M. Conti, and C. Lal, "SAFETY: Early detection and mitigation of TCP SYN flood utilizing entropy in SDN," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 4, pp. 1545–1559, Dec. 2018.
- [35] J. David and C. Thomas, "DDoS attack detection using fast entropy approach on flow-based network traffic," *Proc. Comput. Sci.*, vol. 50, pp. 30–36, Jan. 2015.

- [36] S. B. I. Shah, M. Anbar, A. Al-Ani, and A. K. Al-Ani, "Hybridizing entropy based mechanism with adaptive threshold algorithm to detect RA flooding attack in IPv6 networks," in *Computational Science and Technology*, vol. 481. Singapore: Springer, 2019.
- [37] G. Fioravanti, M. G. Spina, and F. De Rango, "Entropy based DDoS detection in software defined networks," in *Proc. IEEE 20th Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2023, pp. 636–639.
- [38] P. Zhai, Y. Song, X. Zhu, L. Cao, J. Zhang, and C. Yang, "Distributed denial of service defense in software defined network using OpenFlow," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, Aug. 2020, pp. 1274–1279.
- [39] P. Cisar and S. M. Cisar, "EWMA statistic in adaptive threshold algorithm," in *Proc. 11th Int. Conf. Intell. Eng. Syst. (INES)*, no. 4, 2007, pp. 51–54.
- [40] N. Saritakumar and K. V. Anusuya, "Early detection and mitigation of DoS attacks in SDN controller," in *Proc. Int. Conf. Intell. Innov. Eng. Technol. (ICIET)*, Sep. 2022, pp. 315–322.
- [41] A. B. Dehkordi, M. R. Soltanaghaei, and F. Z. Boroujeni, "The DDoS attacks detection through machine learning and statistical methods in SDN," *J. Supercomputing*, vol. 77, no. 3, pp. 2383–2415, 2021.
- [42] N. Niknami and J. Wu, "Entropy-KL-ML: Enhancing the entropy-KL-based anomaly detection on software-defined networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 6, pp. 4458–4467, Nov. 2022.
- [43] A. Yadav, A. S. Kori, D. G. Narayn, P. Shettar, and M. M. Moin, "A hybrid approach for detection of DDoS attacks using entropy and machine learning in software defined networks," in *Proc. 12th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT)*, Jul. 2021, pp. 1–7.
- [44] K. S. Sahoo and D. Puthal, "SDN-assisted DDoS defense framework for the Internet of Multimedia Things," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 16, no. 3s, pp. 1–18, Oct. 2020, doi: 10.1145/3394956.
- [45] S. Y. Khamaiseh, A. Al-Alaj, and A. Warner, "FloodDetector: Detecting unknown DoS flooding attacks in SDN," in *Proc. Int. Conf. Internet Things Intell. Appl. (ITIA)*, Nov. 2020, pp. 1–5.
- [46] J. Bhayo, S. A. Shah, S. Hameed, A. Ahmed, J. Nasir, and D. Draheim, "Towards a machine learning-based framework for DDoS attack detection in software-defined IoT (SD-IoT) networks," *Eng. Appl. Artif. Intell.*, vol. 123, Aug. 2023, Art. no. 106432.
- [47] W. Sun, Y. Li, and S. Guan, "An improved method of DDoS attack detection for controller of SDN," in *Proc. IEEE 2nd Int. Conf. Comput. Commun. Eng. Technol. (CCET)*, Aug. 2019, pp. 249–253.
- [48] Y. Liu, T. Zhi, M. Shen, L. Wang, Y. Li, and M. Wan, "Software-defined DDoS detection with information entropy analysis and optimized deep learning," *Future Gener. Comput. Syst.*, vol. 129, pp. 99–114, Apr. 2022.
- [49] R. M. A. Ujjan, Z. Pervez, K. Dahal, W. A. Khan, A. M. Khattak, and B. Hayat, "Entropy based features distribution for anti-DDoS model in SDN," *Sustainability*, vol. 13, no. 3, p. 1522, Feb. 2021.
- [50] L. Wang and Y. Liu, "A DDoS attack detection method based on information entropy and deep learning in SDN," in *Proc. IEEE 4th Inf. Technol., Netw., Electron. Autom. Control Conf. (ITNEC)*, vol. 1, Jun. 2020, pp. 1084–1088.
- [51] A. El Kamel, H. Eltaief, and H. Youssef, "On-the-fly (D)DoS attack mitigation in SDN using deep neural network-based rate limiting," *Comput. Commun.*, vol. 182, pp. 153–169, Jan. 2022.
- [52] M. P. Novaes, L. F. Carvalho, J. Lloret, and M. L. Proença, "Adversarial deep learning approach detection and defense against DDoS attacks in SDN environments," *Future Gener. Comput. Syst.*, vol. 125, pp. 156–167, Dec. 2021.
- [53] J. Hussain and V. Hnamte, "A novel deep learning based intrusion detection system: Software defined network," in *Proc. Int. Conf. Innov. Intell. Informat., Comput., Technol. (ICT)*, Sep. 2021, pp. 506–511.
- [54] M. S. E. Sayed, N.-A. Le-Khac, M. A. Azer, and A. D. Jurcut, "A flow-based anomaly detection approach with feature selection method against DDoS attacks in SDNs," *IEEE Trans. Cognit. Commun. Netw.*, vol. 8, no. 4, pp. 1862–1880, Dec. 2022.
- [55] R. P. Nayak, S. Sethi, S. K. Bhoi, K. S. Sahoo, and A. Nayyar, "ML-MDS: Machine learning based misbehavior detection system for cognitive software-defined multimedia VANETs (CSDMV) in smart cities," *Multimedia Tools Appl.*, vol. 82, no. 3, pp. 3931–3951, Jan. 2023.
- [56] M. Nugraha, I. Paramita, A. Musa, D. Choi, and B. Cho, "Utilizing OpenFlow and sFlow to detect and mitigate SYN flooding attack," *J. Korea Multimedia Soc.*, vol. 17, no. 8, pp. 988–994, Aug. 2014.
- [57] M. A. Al-Adaileh, M. Anbar, Y.-W. Chong, and A. Al-Ani, "Proposed statistical-based approach for detecting distribute denial of service against the controller of software defined network (SADDCS)," in *Proc. MATEC Web Conf.*, vol. 218, 2018, p. 02012.
- [58] N. I. G. Dharma, M. F. Muthohar, J. D. A. Prayuda, K. Priagung, and D. Choi, "Time-based DDoS detection and mitigation for SDN controller," in *Proc. 17th Asia-Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Aug. 2015, pp. 550–553.
- [59] C. YuHunag, T. MinChi, C. YaoTing, C. YuChieh, and C. YanRen, "A novel design for future on-demand service and security," in *Proc. IEEE 12th Int. Conf. Commun. Technol. (ICCT)*, Nov. 2010, pp. 385–388.
- [60] A. Hussein, I. H. Elhadj, A. Chehab, and A. Kayssi, "SDN security plane: An architecture for resilient security services," in *Proc. IEEE Int. Conf. Cloud Eng. Workshop (ICEW)*, Apr. 2016, pp. 54–59.
- [61] K. Hong, Y. Kim, H. Choi, and J. Park, "SDN-assisted slow HTTP DDoS attack defense method," *IEEE Commun. Lett.*, vol. 22, no. 4, pp. 688–691, Apr. 2018.
- [62] R. Sanjeetha, K. N. A. Shastry, H. R. Chetan, and A. Kanavalli, "Mitigating HTTP GET FLOOD DDoS attack using an SDN controller," in *Proc. Int. Conf. Recent Trends Electron., Inf., Commun. Technol. (RTEICT)*, Nov. 2020, pp. 6–10.
- [63] S. Park, Y. Kim, H. Choi, Y. Kyung, and J. Park, "HTTP DDoS flooding attack mitigation in software-defined networking," *IEICE Trans. Inf. Syst.*, vol. E104.D, no. 9, pp. 1496–1499, 2021.
- [64] A. N. H. D. Sai, B. H. Tilak, N. S. Sanjith, P. Suhas, and R. Sanjeetha, "Detection and mitigation of low and slow DDoS attack in an SDN environment," in *Proc. Int. Conf. Distrib. Comput., VLSI, Electr. Circuits Robot. (DISCOVER)*, Oct. 2022, pp. 106–111.
- [65] C. Gkoutis, M. Taha, J. Lloret, and G. Kambourakis, "Lightweight algorithm for protecting SDN controller against DDoS attacks," in *Proc. 10th IFIP Wireless Mobile Netw. Conf. (WMNC)*, Sep. 2017, pp. 1–6.
- [66] Y. E. Oktian, S. Lee, and H. Lee, "Mitigating denial of service (DoS) attacks in OpenFlow networks," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2014, pp. 325–330.
- [67] R. Wang, Z. Jia, and L. Ju, "An entropy-based distributed DDoS detection mechanism in software-defined networking," in *Proc. IEEE Trust-com/BigDataSE/ISPA*, Aug. 2015, pp. 310–317.
- [68] V. Pashkov and A. Antipina, "Protection of the control plane from DDoS attacks in software-defined networks," in *Proc. Int. Conf. Mod. Netw. Technol. (MoNeTec)*, Oct. 2022, pp. 1–7.
- [69] L. Wei and C. Fung, "FlowRanger: A request prioritizing algorithm for controller DoS attacks in software defined networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 5254–5259.
- [70] A. Shoeb and T. Chithralekha, "Resource management of switches and controller during saturation time to avoid DDoS in SDN," in *Proc. IEEE Int. Conf. Eng. Technol. (ICETECH)*, Mar. 2016, pp. 152–157.
- [71] S. Lim, S. Yang, Y. Kim, S. Yang, and H. Kim, "Controller scheduling for continued SDN operation under DDoS attacks," *Electron. Lett.*, vol. 51, no. 16, pp. 1259–1261, Aug. 2015.
- [72] Y. Cui and Q. Qian, "MIND: Message classification based controller scheduling method for resisting DDoS attack in software-defined networking," in *Proc. 5th Int. Conf. Comput. Commun. Syst. (ICCCS)*, May 2020, pp. 486–490.
- [73] P. Zhang, H. Wang, C. Hu, and C. Lin, "On denial of service attacks in software defined networks," *IEEE Netw.*, vol. 30, no. 6, pp. 28–33, Nov. 2016.
- [74] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 413–424.
- [75] H. Wang, L. Xu, and G. Gu, "FloodGuard: A DoS attack prevention extension in software-defined networks," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2015, pp. 239–250.
- [76] J. Boite, P.-A. Nardin, F. Rebecchi, M. Bouet, and V. Conan, "Statesec: Stateful monitoring for DDoS protection in software defined networks," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Jul. 2017, pp. 1–9.
- [77] K. Kalkan, G. Gür, and F. Alagöz, "SDNScore: A statistical defense mechanism against DDoS attacks in SDN environment," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2017, pp. 669–675.
- [78] J. E. Varghese and B. Muniyal, "An efficient IDS framework for DDoS attacks in SDN environment," *IEEE Access*, vol. 9, pp. 69680–69699, 2021.
- [79] H. D. Zubaydi, M. Anbar, and C. Y. Wey, "Review on detection techniques against DDoS attacks on a software-defined networking controller," in *Proc. Palestinian Int. Conf. Inf. Commun. Technol. (PICICT)*, May 2017, pp. 10–16.
- [80] T. Wolf and J. Li, "Denial-of-service prevention for software-defined network controllers," in *Proc. 25th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2016, pp. 1–10.

- [81] C. B. Serna and C. Mas-Machuca, "Preventing control plane overload in SDN networks with programmable data planes," in *Proc. 18th Int. Conf. Netw. Service Manage. (CNSM)*, Oct. 2022, pp. 37–45.
- [82] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Advances in Cryptology—CRYPTO*, E. F. Brickell, Ed. Berlin, Germany: Springer, 1993, pp. 139–147.
- [83] Mininet. (2018). *Mininet Overview—Mininet*. Accessed: May 6, 2023. [Online]. Available: <http://mininet.org/overview/> and <http://mininet.org/overview/>
- [84] (2010). *Open vSwitch*. Accessed: May 6, 2023. [Online]. Available: <http://www.openvswitch.org/> and <http://www.openvswitch.org/>
- [85] L. R. Prete, A. A. Shinoda, C. M. Schweitzer, and R. L. S. de Oliveira, "Simulation in an SDN network scenario using the POX controller," in *Proc. IEEE Colombian Conf. Commun. Comput. (COLCOM)*, Jun. 2014, pp. 1–6.
- [86] (2022). *Scapy*. Accessed: May 6, 2023. [Online]. Available: <https://scapy.net/> and <https://scapy.net/>



**LUBNA FAYEZ ELIYAN** received the B.Sc. degree in computer engineering and the M.Sc. degree in computer networks from Qatar University. She is currently pursuing the Ph.D. degree in computer science and engineering program with CSE, HBKU, Doha, Qatar, with a major in cybersecurity. Her research interests include computer networks and security and privacy of networks, the IoT, and crowd simulation for studying human behavior.



**ROBERTO DI PIETRO** (Fellow, IEEE) received the M.Sc. degree in computer science and the M.Sc. degree in informatics from the University of Pisa, in 1994 and 2003, respectively, and the PMC degree in operations research and strategic decisions and the Ph.D. degree in computer science from the University of Rome "La sapienza," in 2003 and 2004, respectively. He was a Full Professor of cybersecurity with CSE, HBKU; the Global Head of the Security Research, Nokia Bell

Labs, France; an Associate Professor (with tenure) of computer science with the University of Padua; and a Senior Military Officer with MoD, Italy. He has been working in the security field for more than 25 years, leading both technology-oriented and research-focused teams in the private sector, government, and academia (MoD, United Nations HQ, EUROJUST, IAEA, and WIPO). He is currently a Full Professor of cybersecurity with CEMSE, RC3 Center, KAUST. His research interests include AI driven cybersecurity, security and privacy for wired and wireless distributed systems (e.g., blockchain technology, cloud, the IoT, and OSNs), virtualization security, applied cryptography, computer forensics, and data science. Other than being involved in M&A and strategic advising of start-up—and having founded one (exited)—he has been producing more than 280 scientific papers and patents over the cited topics, has coauthored three books, edited one, and contributed to a few others. He is the Scientific Advisory Board for a few knowledgeable Universities. In 2011 to 2012, he was a recipient of the Chair of Excellence from the University Carlos III, Madrid. In 2020, he was a recipient of the Jean-Claude Laprie Award for significantly influencing the theory and practice of Dependable Computing. In 2022, he was awarded the Individual Innovation Award from HBKU. He is a ACM Distinguished Scientist.

• • •