Wireless Networks (2019) 25:2751–2768
https://doi.org/10.1007/s11276-019-01991-y

Lightweight solutions to counter DDoS attacks in software defined networking

Mauro Conti[1] · Chhagan Lal[1,3] · Reza Mohammadi[2] · Umashankar Rawat[3]

Published online: 25 April 2019

Abstract
A distributed denial of service (DDoS) attack on any of the major components (e.g., controller, switches, and southbound channel) of software defined networking (SDN) architecture is a critical security threat. For example, the breakdown of controller could disrupt the data communication in the whole SDN network. A possible way to perform DoS is to generate a large number of new, but short length traffic flows. These flows will trigger malicious flooding requests to overload the controller and causes overflow in flow tables at SDN switches. In this paper, we propose two lightweight and practically feasible countermeasures against two different types of DDoS attacks called *Route Spoofing* and *Resource Exhaustion* in SDN networks. For *Route Spoofing* attack, we introduce a technique called "selective blocking", which stops an adversary node from maliciously using other users active communication routes. To countermeasure *Resource Exhaustion* attack, we propose a solution called "periodic monitoring", which detects adversary nodes based on the traffic analysis statistics that are gathered within a time window. We implement and perform result analysis of the attacks and their proposed countermeasures. When using our proposed countermeasures in the target SDN scenarios, the simulation results indicate an adequate reduction in bandwidth consumption and processing delay of new request, and it also depicts substantial gain in packet delivery rate. Additionally, we present the receiver operating characteristic curve, which shows the sensitivity and specificity of our countermeasures along with their detection accuracy.

Keywords Software-defined networking (SDN) · Security · Denial-of-service attack · OpenFlow, Resource exhaustion attack

1 Introduction

Over the years, the continuous growth in enterprise networks and data centers such as Google, Facebook, and Microsoft, regarding size and complexity raises various administrative and security challenges. As a result, the network research community recognizes SDN, an approach to meet these challenges. SDN decouples the control logic from the closed and proprietary implementations of traditional network devices. It enables practitioners and scientists to support the design of fine-grained network management policies and innovative network functions in a much more straightforward, flexible, and powerful way [1]. Therefore, SDN has quickly emerged as a novel technology for various networking architectures. For instance, Internet of Things (IoT) [2], Heterogeneous Ultra-Dense Networks [3], and Cloud Computing Environments [4] are some of the applications that uses the unique SDN features to

✉ Chhagan Lal
chhagan@math.unipd.it

Mauro Conti
conti@math.unipd.it

Reza Mohammadi
mohammadi.rm@gmail.com

Umashankar Rawat
umashankar.rawat@jaipur.manipal.edu

[1] Department of Mathematics, University of Padova, Padua, Italy

[2] Computer Engineering Department, Bu-Ali Sina University, Hamedan, Islamic Republic of Iran

[3] Manipal University Jaipur, Jaipur, India

improve their network security or communication reliability. Unfortunately, the lack of smart network planning, security threats, and situation management in SDN limit its performance and applicability in the real world [5].

The reference implementation of SDN that is widely in-use is called *OpenFlow* [6]. The existing OpenFlow protocol implementation in SDN mainly uses two communication interfaces namely "northbound" and "southbound". The northbound interface facilitates communication between the controller and the application/business logic, while the southbound interface is used for relaying information between the control plane and the data plane components. When a matching rule is not present in the forwarding table of an OpenFlow switch (OF-switch) for an incoming message, the data plane asks the controller for further actions through a southbound protocol. In this way, for each new message[1] that an OF-switch receives, it has to query the controller to install new rules. Hence, each new message at OF-switch consumes resources (e.g., CPU, memory and bandwidth) at both the planes. This handling process for each new packet in SDN leads to issues related to its scalability and security. For instance, an attacker could launch a dedicated DDoS attack [7] by sending a large number of new messages to the control plane. These messages will cause communication bottleneck in southbound communication and throttle the throughput and processing capacities of the controller as well as edge OF-switches.

1.1 Motivation

In order to perform a DDoS attack in SDN, the adversary can exhaust resources of one or more of the following network components.

- Buffer memory at OF-switches,
- Communication channel bandwidth between data and control planes,
- Processing power at controller and OF-switches,
- Reserving unnecessary resources at some target servers, and
- Communication channels bandwidth at data plane.

The existing SDN architectures place the network security applications at control plane to counter various security threats such as DDoS attacks. However, implementing security features at control plane hinders the performance of these security applications. It is due to the presence of inherent communication bottleneck between the data plane and the control plane. The bottleneck not only add communication delay, but it also limits the amount of traffic that these security applications can process. Additionally, such implementations also limit scalability due to the presence of an insufficient number of controllers in the SDN as compared with the number of OF-switches. Therefore, to provision possible and robust security applications in SDN, the OF-switches need to support more advanced functionalities rather than just forwarding the control and data messages.

Over the years, various solutions were proposed to countermeasure DDoS attacks in SDN networks [8–11]. Although these approaches have different implementation details to detect attacks, they share a similar design principle and architecture. A vital issue in the designs is to identify which flow information (such as destination IP, port number, and packet sequence number) is essential to be reported to the controller. Note that, due to its capacity limitation, it is not feasible for the controller to receive and analyze information for all network flows from all switches. A simple solution to this problem that has been proposed in most of the state-of-the-art is to rely on OF-switches to perform pre-processing on received flow information, generate summarized statistics (e.g., changes of flow rates), and report the summary to the controller. However, to accomplish the pre-processing at OF-switches, one has to rely on extra components (e.g., appliances), which results in additional deployment cost. Also, flow pre-processing in switches (e.g., analysis of changes of flow rates) may lead to loss in crucial original information, and thereby the controller could mistakenly omit attack flows. It is especially serious when stealthy DDoS attacks (e.g., Crossfire [12] through route spoofing) or non-link-flooding attacks (e.g., TCP flooding to achieve resource exhaustion attack) take place. Under these attacks, the information reported by the OF-switches always fall in the standard category, and thus, it will be difficult for the controller to know if an attack is happen. Hence, such techniques may neither be able to detect specific attacks, and even if they do, the detection accuracy is questionable.

1.2 Contributions

In this paper, we propose and analyze potentially useful solutions to detect and prevent two DDoS attacks in SDN. The first attack called *Route Spoofing* (*RS*), which targets the data plane resources, i.e., computation resources at OF-switches and communication channels bandwidth. The second attack called *Resource Exhaustion* (*RE*), which exploits the southbound interface's scalability issue that exists due to the limited bandwidth channel between the data and control planes. Both these attacks could work in wired or wireless networks that are attached with SDN edge routers. However, the attacks are more effective in the wireless SDN networks because the IP and MAC spoofing

[1] We use the term "new message" for the packets for which the OF-switch does not find a matching rule in its forwarding table.

is relatively easy to perform in wireless networks when compared with the wired networks. The major reasons behind analyzing these two attacks together in this paper are as follow: (1) both the attacks aim to perform DoS through data or control plane against the benign hosts, (2) the system and attack model for both the attacks are same due to the nature of these attacks, and (3) one attack performs DoS at data plane while the other performs it on control plane, thus, these attacks cover the behaviour of a large set of DoS attacks, and we believe that the same countermeasures, which we proposed in this paper can be used with slight or no modifications to counter various other forms of DoS attacks in SDN.

To handle the *RS* attack, we design and implement a module called "selective blocking" at OF-switches. It checks each incoming message at the OF-switches against a set of security policies that are installed on these switches, and the suspected malicious messages will be forwarded to the controller to take further security actions. Another security module called "periodic monitoring", which is also installed at OF-switches to enforce a set of rules on the incoming traffic and it uses randomness in the received traffic as a measure of anomaly detection. The violation of these rules and low randomness in few selected metric values will be suspected as an indication of an ongoing *RE* attack. This paper is an extended version of our preliminary work appeared in [13]. In this paper, we make the following significant research contributions.

- We present novel DDoS attacks, and we show their impact through implementation. In particular, through simulation results, we analyze the behavior of various data and control plane network entities against two DDoS attacks called *Route Spoofing* and *Resource Exhaustion* in various target SDN scenarios. We show that the RS attack effectively causes DDoS for end-hosts by disrupting the communication system at data plane, while the RE causes DDoS by exhausting the resources of controller, OF-switches, and data servers (such as a remote HTTP server).
- We propose two simple and practically feasible extensions called "selective blocking" and "periodic monitoring" which are mainly implemented on the data plane OF-switches. These solutions provide adequate security from the aforementioned DDoS attacks. Unlike our preliminary work that appeared in [13], in this work the "periodic monitoring" countermeasure consists of a two phase detection process, i.e., entropy-based and new low-traffic flows rate based. The two solutions can work separately or can be combined to complement each others limitation. Additionally, we ensure that the proposed extensions are transparent (i.e., no changes are required at end hosts machines) and they exhibit lower collateral impact (i.e., benign flow's setup time is not adversely affected and keep the false positives and false negatives lower).
- To show the feasibility and effectiveness of the proposed countermeasures, we implement a prototype SDN system in *MiniNet* emulator, which is evaluated in various adversarial scenarios. In this extended version, we show that our proposed countermeasures have the following key advantages: (1) promptness - the detection modules can quickly identify the compromised data flows and OF-switch interfaces after a small number of consecutive observations, which are done using number of mice flows, randomness in the specific parameters of received new messages, and identifying the IP and MAC spoofing, (2) versatility: our proposed detection modules exhibits the ability to detect DDoS attacks of versatile nature, i.e., no matter how the malicious traffic is generated, and (3) accuracy: the proposed countermeasures can differentiate between benign, flash, and malicious traffic, and make more accurate decisions when compared to the state-of-the-art. The evaluation results show that the proposed countermeasures can efficiently tackle the two aforementioned DDoS attacks regarding detection time, network throughput, correctness, and network overheads.

1.3 Organization

The rest of this paper is organized as follows. In Sect. 2, we present the related work. Section 3 describes the design and working methodology of *Route Spoofing* attack and its countermeasure, and Sect. 4 describes the design and working methodology of *Resource Exhaustion* attack and its countermeasure. In Sect. 5, we present the details of target SDN scenario that we set up for emulation, and we also present the performance evaluation using the results obtained. Finally, Sect. 6 concludes our work with possible future directions.

2 Related work

The programmability and flexibility provided by SDN allows the network providers to enable in-network security functions. These functions include firewalls, monitoring applications, access control and middlebox support through OF-switches [14]. For instance, authors in [15] propose *Bohatei*, which is a SDN and NFV based flexible and elastic DDoS defense system for traditional networks. Similarly, other research works use SDN technologies for traffic monitoring and tackling DDoS and amplification

attacks in traditional networks [16, 17]. However, before using SDN to improve the security of other networks, it is imperative to ensure that the SDN functionalities are protected from various attacks [18]. To this end, this section reviews the state-of-the-art countermeasure techniques that have been proposed in SDN to address its various security challenges [10].

SPHINX [19] is a framework which detects SYN flooding attack by investigating the rate of the new SYN requests. If the rate of new SYN request is above the administrator-specific threshold, SPHINX raises the alarm, and the controller can perform the prevention. FlowFence [20] is a defense system for SDN which is based on bandwidth coordination technique. In FlowFence, each OF-switch has a monitor that measures the average occupation of its interfaces. If the monitor detects any congestion on the interfaces, the OF-switch reports to the controller. To prevent users' starvation, the controller limits the rate of ongoing flow along a path.

Authors in [21] presents AVANT-GUARD, which detect and prevent the TCP SYN flooding attack. AVANT-GUARD uses connection migration mechanism in data plane switches as a proxy for incoming TCP SYN packets and limits the effect of the control plane saturation attack. AVANT-GUARD informs the controller and request for flow rules only for the users who have been performed three-way TCP handshaking process. It handles only attacks launched using TCP services, but no support is provided for UDP flooding attacks. While [22] presents TopoGuard, which is a security extension for SDN. It is implemented on the controller, and it can detect network topology poisoning attacks along with the host location hijacking and link fabrication attacks. LineSwitch [23] is another solution which has been proposed to eliminate the weaknesses of AVANT-GUARD. Similar to AVANT-GUARD, LineSwicth acts as a TCP proxy and it also uses connection migration mechanism. But, unlike AVANT-Guard, LineSwitch proxies a minimum number of TCP requests and uses probabilistic blacklisting mechanism.

In [9], authors propose an entropy-based DDoS detection mechanism implemented at the edge OF-switches. The authors extend a copy of the packet number counter of each flow table entry in the OpenFlow table called as "*RP_Local*". The entropy is calculated by each OF edge switch based on the destination IP's which are present in their network. However, the approach requires placing the proposed solution in every OF-edge switch as part of DDoS detection, and it assumes a static threshold and provides no method to discriminate flash crowd from the attack traffic.

FloodGuard [24] is an OpenFlow extension for preventing DOS attacks in SDN. FloodGuard consists of two modules: (1) proactive flow rule analyzer, and (2) packet migration. The former proactively insert the flow rules to the switches to prevent overloading problem during the data-to-control plane saturation attack. In [25], authors propose a collaborative technique for SYN flooding attack detection and containment. Authors develop new components called "monitors" and "correlators" for mitigating DDoS attacks. The monitor continuously listens to the ongoing traffic to detect the SYN packets with different source IP addresses which denote IP spoofing. When the monitor detects an attack, it informs the correlator by sending an alert message that contains the number of source IP addresses found in SYN packets.

In [26], authors introduce a new attack in SDN namely "freeloading" attack in which an attacker bypasses the process of installing the flow rules, and it sends their messages by exploiting the existing flow rules in the forwarding table of an OF-switch. In freeloading, an attacker spoof's IP/MAC address of an existing host in the network, and it gathers the information about the host's flow rules. Then, the attacker compromises the physical access to the local networks and launches a stealthy attack without installing any flow rules. To prevent this type of attacks, the authors proposed watermarking technique. The proposed countermeasure requires changes in end-host software, thus lacks transparency.

Authors in [8] proposes SLICOTS, an extension module in the OpenDayLight controller to countermeasure TCP SYN flooding attack. In SLICOTS, first, all the TCP communications are stored on their nature of set-flag in a *"pending_list"*. The pending list attaches every connection record with either $\langle SYN, SYN\_ACK, RST \rangle$. Any record on "pending_list" is counted as an illegitimate record, and if the number of illegitimate records for a specific host exceeds a predefined threshold (K), it will install a DROP rule on the edge switch of that host. SLICOTS suffers from MAC spoofing, and it statically chooses a threshold of the pending connections associated with a host, and it does not provide any discriminating behavior between flash crowd vs. attack traffic. Similarly, authors in [10] presents a thorough treatment of solutions against DDoS in the SDN environment.

FL-GUARD (Floodlight-based guard system) in [27] is presented as a solution to handle the DDoS attacks in SDN. The proposed system works in three steps, first, it realize an anti-spoofing of source IP address, second, it support vector machine algorithm to detect the DDoS attacks, and third, the controller inserts a new rule in the flow table to block attacks at the source port. Authors in [11] proposes Reinforcing Anti-DDoS Actions in Realtime (RADAR) to detect and throttle DDoS attacks via adaptive correlation analysis built upon unmodified commercial off-the-shelf SDN switches. It is a practical system to defend against a wide range of flooding-based DDoS attacks, e.g., link flooding, SYN flooding, and UDP-based amplification

attacks while requiring neither modifications in SDN switches/protocols nor new appliances. However, the paper does provide a way to differentiate between the flash and malicious traffic, and it suffers from the MAC and IP spoofing attacks. Recently, authors in [28] present SGuard, a security application on top of the NOX controller that mainly contains two modules: Access control module and Classification module. It employs six-tuple as a feature vector to classify traffic flows, meanwhile optimizing classification by feature ranking and selecting algorithms. All the modules will cooperate with each other to complete a series of tasks such as authorization, classification and so on.

3 Route spoofing attack and countermeasure

In this section, we discuss the working methodology of *Route Spoofing* (*RS*) attack, and we present details of our proposed countermeasure for the same.

3.1 Route spoofing attack

In SDN, it is possible for malicious users to send their messages by exploiting the existing active data flow's information from the network. In wireless local area networks, where multiple nodes are connected to an OF-switch, an adversary can learn about the active data flows by first performing eavesdropping attack on these OF-switches. To perform eavesdropping, the adversary can simply use the promiscuous mode feature of the wireless networks. Once the adversary has information about the active data flows, then it can use these spoofed routes for data transmission by using the IP addresses of their neighbor nodes as the source IP address for its own transmitting messages. Thus, performs the *RS* attack by sending traffic on active routes that belong to other users in the network. Additional, other information about network topology and active data flows can also be easily obtained using network tools such as trace-route, ping, and link layer discovery packet (LLDP) [26]. An adversary can use multiple neighbor IP addresses to send a large amount of traffic to different destinations in the network. This behavior can lead to data plane resource exhaustion attack because the adversary could spoof a set of active routes, and it uses these to transmit unwanted high data-rate traffic in the network. The presence of high data-rate unwanted traffic will cause low packet delivery for legitimate traffic flows due to link congestion, it increases end-to-end delay, and OF-switches input-output buffer will overflow.

The possible real-world scenarios in which the *RS* attack affects greatly are as follows:

- In a network, where each user has to pay for their data usage, an adversary can get free data transmission, and the genuine user whose IP address is used by the adversary for data transmission has to pay extra.
- In a network, where the data plane has limited communication bandwidth due to a large number of users in the network, such behavior of adversary will result in a DoS attack. Depending upon the rate of data transmission by an adversary on the spoofed routes or number of active routes spoofed, it can cause a different level of congestion on the data plane.

Figure 1 illustrates an instance of a possible attack scenario for *RS* attack. Table 1 depicts an instance of the forwarding rule table (FR_{table}) at OF-switch (say $S1$). As it can be seen from Table 1, the rules that the controller has installed has the following interpretation; all the packets received from port 1 with source and destination IPs as 192.168.1.1 and 192.168.1.4 will be forwarded to output port 3, and all the packets received from port 2 with source and destination IPs as 192.168.1.2 and 192.168.1.3 will be forwarded to port 4. Let's assume that host 2 is an attacker node, and as per the entries in Table 1, it can only send data to host 3. Both host 1 and host 2 are connected to a switch, which is then connected to port 1 of $S1$. In this scenario, the attacker can easily spoof the IP address of host one as both are in the same local area network, and later it will use the spoofed IP address as source address in all the messages that it will transmit to host 3. In this way, host 2 uses a forged address, and it can lead towards possible data plane resource exhaustion attack.

3.2 Proposed countermeasure

The detection unit of our proposed countermeasure called "selective blocking" for *RS* attack comprises of two separate modules (i.e., IP and MAC spoofing detectors). These modules also run on the OF-switches as it is shown in Fig. 2, hence, an adversary that resides on the host network has no knowledge or has no control over these modules. The adversary can only perform the IP spoofing or MAC spoofing on the hosts that resides in the same network with the aim to launch the *RS* attack. It is seen from Fig. 2 that upon reception of a new message, IP spoofing detector checks for IP spoofing attack because a *RS* attacker uses spoofed IP addresses. If the detector detects an attack, i.e., multiple nodes are using the same source IP, it notifies the controller for further action. However, if no IP spoofing is detected then the possibility that the IP spoofing is being done along with MAC spoofing is checked. It is done by forwarding the message to the MAC spoofing detector. If a MAC spoofing attack is detected, it is informed to the controller. If no spoofing is detected, the packet is

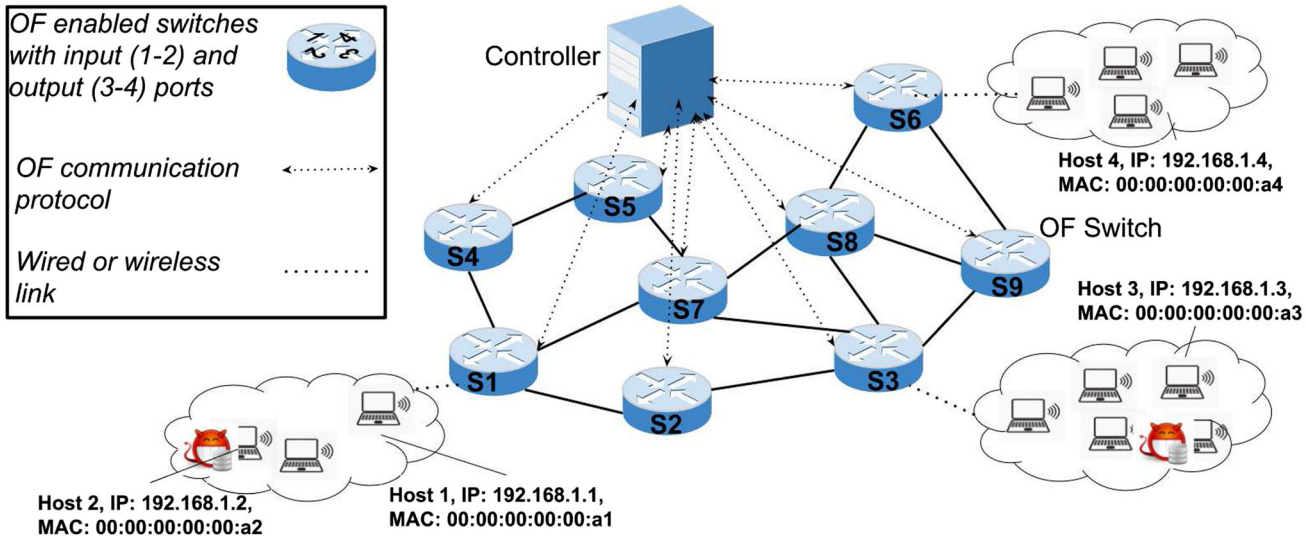


Fig. 1 Example—Target SDN scenario

Table 1 Forwarding table at $S1$

P_in	SRC_MAC	DST_MAC	SRC_IP	DST_IP	...	ACTIONS
1	*	*	192.168.1.1	192.168.1.4	...	P_out 3
2	*	*	192.168.1.2	192.168.1.3	...	P_out 4
...	...	...	...	...	...	...

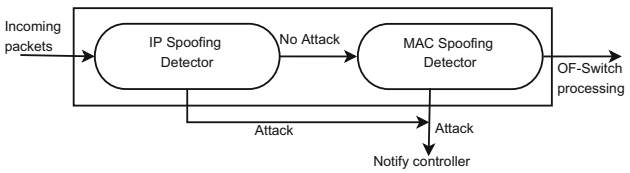


Fig. 2 Detection unit running at OF-Switches

Table 2 network_table instance

IP address	MAC address
92.168.1.1	00:00:00:00:00:a1
192.168.1.2	00:00:00:00:00:a2
192.168.1.3	00:00:00:00:00:a3
192.168.1.4	00:00:00:00:00:a4
...	...

processed normally by OF-switch. The functionalities of these two detection modules are as follows.

The *IP spoofing detector* will use the information gathered from MAC layer header of the received new packet. We implement the indicator on the OF-switch as a module that runs the detection process for the *RS* attack, and upon detection, it sends an *alarm_packet* to the controller. The prevention unit is implemented at the controller. During its processing, the detection unit creates a table named *network_table*, which stores IP and MAC addresses of currently active flows. An instance of *network_table* is shown in Table 2. Each entry in the *network_table* consists of a unique $<IP, MAC>$ pair, which are currently active in data transmission in SDN network. *network_table* is indexed by MAC address. We assume that a single IP address is bind with only one network interface card (i.e., MAC address) at any given time, thus each entry of $<IP, MAC>$ pair in the *network_table* exhibits a relation R, which satisfy the functional dependency $IP_i \rightarrow MAC_i$ (i.e., each IP_i value in R is associated with precisely one MAC_i value in R). When an OF-switch receives a packet, the IP spoofing detector running on it will extract the $<IP, MAC>$ pair from the packet, and checks it in the *network_table* for a relation R. If this pair is not present in the *network_table*, it is added to it. But, if the switch finds an IP address which is associated with two or more different MAC addresses in the *network_table*, it concludes that an adversary might be using an IP address of a neighbor to send traffic in the network anonymously. However, the detection process will fail, if an adversary spoofs the $<IP, MAC>$ pair because by doing so it will be able to satisfy the relation R. Thus, the IP spoofing detector will not able to detect an IP spoofing instance. For this reason, all the packets that

passes the first detector will feed to the second detector (i.e., MAC spoofing detector) as it is shown in Fig. 2.

The *MAC spoofing detector* is implemented based on the approach presented in [29]. The proposed technique keeps track of the sequence number of each traffic flow to detect MAC spoofing attack. Upon reception of a frame, the algorithm calculates the gap G between the sequence number of the current frame and that of the last frame received from the same source address. If $G = 0$, the current frame is considered as a retransmitted frame, while if $G = 1$ or $G = 2$, the current frame is considered the right one. But, if the gap between the current frame and previous frame is in between 3 and 4096, then it is considered an abnormal sequence number. We choose the method given in [29] for our MAC spoofing detector implementation due to its following advantages:

- It does not require any change at end hosts.
- There is no additional overhead incurred in extraction of sequence numbers for MAC spoofing detection because these fields are extracted from each received packet when the packet header is being checked for the *RS*) attack detection.
- The detection approach remains robust, even in cases, where an adversary predicts the next sequence number. In such case, the OF-Switch will receive two frames (one from adversary and one from the benign host) with the same sequence number, and the OF-switch needs to check weather its a retransmitted frame or a spoofed frame. To resolve such situations, the OF-switch keeps a copy of the recently received frame, and it can use the copy to verify if the current frame is retransmitted frame or not.
- It only leverages the sequence number field in the link-layer header of IEEE 802.11 frames.
- The false positive and false negative rate of this method is nearly zero.

Once an anomaly is encountered by any of the detection units running on the edge OF-switch, the switch generates a packet named *alarm_packet* consisting of the detected source host's $< IP, MAC >$ pair. The mitigation unit at controller gets activated as soon as it receives the *alarm_packet*. We implement the mitigation unit as a module on the OpenDayLight controller, which uses *_IListenDataPacket_* interface of OpenDayLight to listen for incoming packets. In particular, inside the OpenDay-Light controller, a module named "Host Tracking" create and maintain a *Host Profile* data structure to track the location of a host, i.e., from which OF-switch a host is connected to the SDN network. The Host Profile contains MAC, IP and Location (i.e., network address) of the host, this information is collected by the controller when a host connects first time with an OF-switch. The prevention module uses this Host Profile to recognize the attacker in the SDN. When the prevention module receives the *alarm_packet* from an OF-switch through the OpenFlow southbound interface, it first extracts the $< IP, MAC >$ pairs from the packet. Then, it checks them with the Host Profile. In the Host Profile structure, the IP addresses will be associated with only one MAC address and not with the rest of the MAC address(es) that are mentioned in the received packet, thus, it identifies the host that is using a neighbor IP address to perform *RS* attack. As a result, the controller sends a forwarding rule to block the malicious host(s).

When an attacker performs the IP and MAC spoofing at the same time, our detection approach is able to identify the attack, however it is not possible to identify which of the one from the two identified nodes is an attacker. For instance, when the MAC spoofing detector founds two frames with the same sequence number and they are not the identical (i.e., not a retransmission event), it has no way to identify which of the frame is spoofed one. Therefore, in our *RS*) attack detection approach, while blocking the attacker flows the legitimate users might also get blocked. However, our approach blocks the detected malicious flows for a short random duration. The blocking period starts with a minimum value (i.e., 5 ms), and this value increases each time the same flow is detected as malicious flow by the controller. The blocked period allows the genuine hosts attached to that flow to use the specific countermeasures at their end to avoid future IP and MAC spoofing. Our blocking approach is aimed to let the benign hosts to work at their full capacity, which we achieve by our method because as soon as the malicious hosts high rate traffic is blocked, the network resources become available for the benign hosts. Figure 3 depicts our proposed attack detection and prevention technique for *Route Spoofing* attack regarding a flow-chart.

4 Resource exhaustion attack and countermeasure

In this section, first we describe the *Resource Exhaustion* (*RE*) attack which leads to data and control plane saturation in SDN. Later, we present our proposed countermeasure to detect and mitigate the *RE* attack. Table 3 provides the list of symbols along with their meaning that has been used to explain the RE attack and its countermeasure.

4.1 Resource exhaustion attack

In *RE* attack, the aim of an adversary is two-folded.

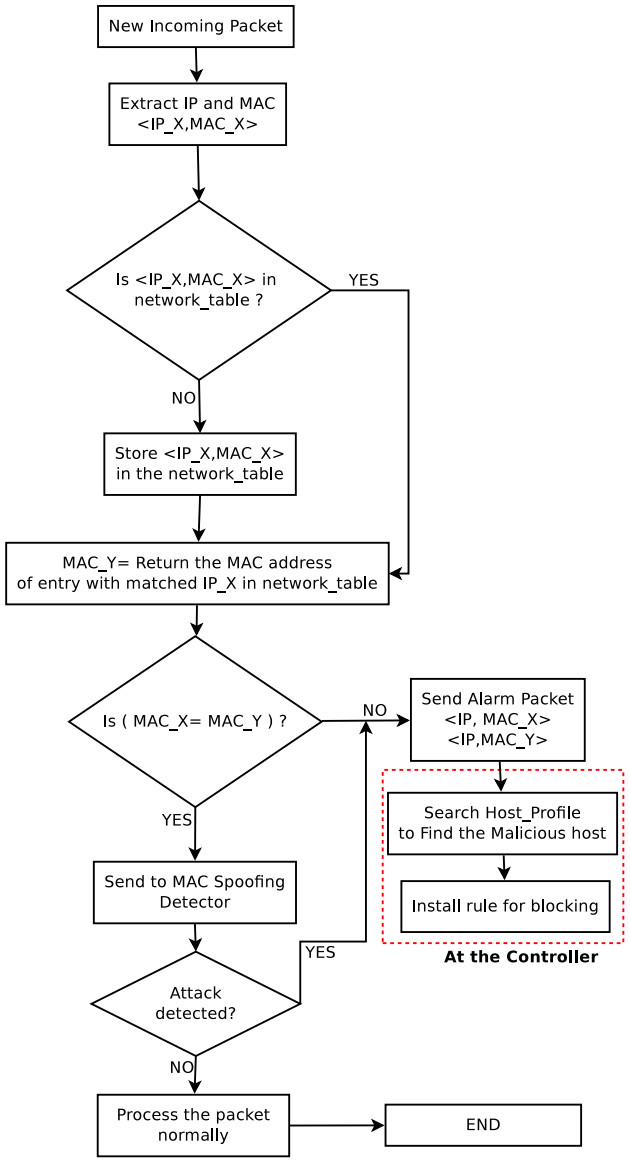


Fig. 3 Flow-chart to countermeasure route spoofing attack

– Perform a TCP SYN flooding attack to target the controller's processing power and destination server's resources. When a new SYN packet is received by an OF-switch, the packet is sent to the controller for further processing. The controller identifies a route between source and target server and install the forwarding rules on the data plan switches. Upon reception of a SYN message the server opens a new connection and allocate a set of resources to this new connection. In this way, an attacker uses several fake SYN packets to waste: (1) resources at the server, (2) processing power at the controller, and TCAM memory at the OF-Switches. The primary detection phase of our proposed countermeasure detects the TCP SYN flooding attack (refer to Sect. 4.2.1).

– An adversary send huge amount of new packets (or new very small duration data flows) with different source IP and port, and/or destination IP and port. For each such packet, the receiving OF-switch will forward it to the controller for further processing. Such packets exhaust the southbound link bandwidth, OF-Switch TCAM memory, and controller resources, thus denies or delays service to benign hosts. We consider that these new packets are UDP packets, but the attack can be combined with the TCP SYN flooding packets as well. To identify such fake data flows, we propose a countermeasure, which we explain in Sect. 4.2.2.

4.2 Proposed countermeasure

To countermeasure the RE attack, we propose a lightweight solution called "periodic monitoring". It detects the adversary nodes based on the traffic analysis statistics gathered over a period using a two-phase (called primary i.e., Sect. 4.2.1, and secondary i.e., Sect. 4.2.2) anomaly detection module. Once an anomaly is detected, the switch notifies about it to the controller for mitigation purposes. The detection module is implemented on the OF-switch as an extension, while the prevention technique is implemented at the controller.

4.2.1 Primary detection phase

In this phase, our proposed approach uses the randomness of data packets in a flow to detect the DDoS attack. It is because an increase in randomness in data flows lead to an increase in entropy and vice verse. The two components that are essential for DDoS detection using entropy are: (a) window size, and (b) entropy threshold value. The window size is either based on a period or on a specified number of packets, we use a period as a measure for window size. Entropy is calculated within this period to measure the randomness in receiving packets. However, to detect an attack, a threshold entropy is required. If the newly calculated entropy passes a threshold entropy or is below it, then depending on the detection technique used, a possible attack can be detected.

Equation 1 shows a window, where r_i is the random variable and f_i represent its frequency of occurrence. To compute the entropy, Eq. 2 is used where $P(r_i)$ denotes the probability of occurrence of each random variable in the set.

$$W = \{(r_1,f_1),(r_2,f_2),(r_3,f_3),\ldots,(r_n,f_n)\}, \quad (1)$$

$$P(r_i) = f_i/N \quad (2)$$

Table 3 Notations

Symbol	Meaning
R	Discrete random variable
r_i	Instance of R
f_i	Frequency of occurrence of r_i
$P(r_i)$	Probability of occurrence r_i in W
W	Monitoring window
E_i	Set of entropy in W
N	Total number of occurrences of all the outcomes
n	Number of unique outcomes
λ	Threshold entropy
Δt	Time interval of W
σ	Standard deviation of normal entropy values
μ	Number of nearest normal entropy values, i.e., mean entropy
M	Minimum times that satisfied the Eq. 6

$$N = f_1 + f_2 + f_3 + \cdots + f_n, \tag{3}$$

Here, **N** represent the total number of occurrences of all the outcomes. The entropy of a discrete random variable **(R)** that is present in a system is defined as follow.

$$E(R) = \sum_{i=1}^{n} -P(r_i) log_2 P(r_i). \tag{4}$$

Here, **n** represent the number of unique outcomes (i.e., unique $dest_{IP}$ or $dest_{port}$). Since entropy falls in the range of [0, $log_2 n$], and it would vary for different window sizes. Therefore, we normalize the entropy to get fall in the same scale of entropy, i.e., in the range of [0, 1], which is independent of the sample size of the window as it can be seen in Eq. 5.

$$E^N(R) = \frac{E(R)}{log_2 n}. \tag{5}$$

Our approach calculates the entropy of incoming new packets in a fixed time window to determine a possible abnormal behavior. In particular, entropy is calculated on a *set*, where the set consists of one or more of the following parameters, namely, source IP address (src_{IP}), destination IP address ($dest_{IP}$), destination port ($dest_{port}$), OF switch input port (OFS_{ip}), and TCP flags (TCP_{flag}) [30]. In general, in non-attack scenarios there are multiple hosts which communicates with each other and their packets identified with a set $< dest_{IP}, dest_{port} >$ or $< src_{IP}, dest_{IP} >$ will be randomly distributed in the network. However, in the presence of an adversary, a particular service irrespective of its location will generate packets with same $dest_{IP}$, $dest_{port}$ or generate a large number of new messages by just changing the $dest_{IP}$. In such scenarios, the randomness is limited due to nature of the attack. Hence entropy will decrease to a noticeable level. Although an adversary could also send new messages with spoofed IP addresses, this will increase the randomness (or entropy) when it is calculated using the set $< src_{IP}, dest_{IP} >$. But, if the entropy calculation is being done based on the *number of new packets received with different source IP in a given window*, then the randomness will be low. However, the low randomness could also occur if the same source is establishing some genuine connections with different destination nodes. For such cases, i.e., where a possibility of an ongoing attack is detected by our primary phase, the secondary phase will do further analysis to ensure that the lower entropy is due to an attack and not due to the increase in the genuine traffic flows in the network.

Key Observation

Let's take an example of TCP SYN flooding attack. Flow entropy is higher for a normal case of traffic scenario because the number of packets for a destination IP will be of uniform nature. However, once message flooding (such as SYN packets) is active, there will be a dominance of attack flows, and consequently, a continuous significance decrease in the entropy would be observed for the duration of the attack period. It is because the concentration of packets for a particular destination IP (victim node) will be very much greater than for the other destination IP. Therefore, it causes a situation of alarm when there is a sudden decrease in entropy.

Formal Proof Assume that a TCP SYN flooding attack is taking place. Let E(R) denotes entropy at the two consecutive time units within an interval, say Δt (t_0 and $t_0 + \Delta t$) respectively. As the difference found in the subsequent entropy values is less than a quantity λ (threshold entropy), hence we can write:

$$E(R)\,|_{t=t_0} - E(R)\,|_{t=t_0+\Delta t} \geq \lambda, \tag{6}$$

where, t represents time, and λ ($\lambda > 0$) represents the threshold entropy at that point of time. Subsequently, we are now in a position to ascertain whether entropy will decrease or increase continuously for the chosen time window. Therefore, we have to test the nature of the deviate of entropy over the assumed time period.

$$E^{'}(R) = \lim_{\Delta t \to 0} \frac{E(R)\,|_{t=t_0+\Delta t} - E(R)\,|_{t=t_0}}{\Delta t}. \tag{7}$$

Combining Eqs. 6 and 7 we get,

$$E^{'}(R) \leq -\lambda, \tag{8}$$

$$E^{'}(R) < 0. \tag{9}$$

Using Eqs. 7–9, it could be explained that if there is a continuous decrements occur in the values of entropy, and it is found to be lower than a threshold, it could be seen as a

sign of either malicious traffic or a surge of a flash crowd. Hence, further work has to be done to discriminate the two flows. To this end, we will propose, in Sect. 4.2.2, the details of our secondary detection phase which executes after the primary phase to improve the true positive rate of our detection process. Please note that both the detection phases will run in parallel and report the controller when an anomaly is found. Therefore, in cases, where the primary phase is fail to differentiate between malicious traffic and flash traffic, the secondary phase might get success or vice verse.

Threshold Entropy Estimation

Threshold estimation process consists of two parts. First is to find an average entropy value, and the second is to perform a normal distribution analysis on the calculated list of entropy values to get an average entropy (*apt*) which is then set to threshold entropy. For calculating the *apt*, we setup the network traffic in the following manner.

- 5% of the overall network traffic directed towards victim host.
- 10% of the overall network traffic directed towards a victim host.
- 15% of the overall network traffic directed towards a victim host.
- 20% of the overall network traffic directed towards a victim host.
- 25% of the overall network traffic directed towards a victim host.

Figure 4 describes our theory (using the simulation results) for getting the normalized entropy values with respect to above mentioned approach. All the five scenarios would provide us a set of points of valid normalized entropy which is subject to a varied intensity of traffic distribution.

Collected samples of normal entropy are subjected to normal distribution analysis of statistical approaches. When scatter-plotting these values, we see that it follows a Bell-like curve. Therefore according to "standard deviation and coverage theorem of normal distribution theory" about 68% of values come from a normal distribution is found to fall in one standard deviation (σ) with the mean (μ), about 95% of the values lie fall two standard deviations, and about 99.7% falls within three standard deviations. This fact is well known as the $68-95-99.7$ (empirical) rule or the *3-sigma rule*. Therefore, mathematically for finding the initial or subsequent points in the threshold. Once a set of points of valid normalized entropy (E_i) is available which is gathered during a fixed window size, our *THRESHOLD* function shown in Algorithm 1 will be used to calculate the threshold entropy.

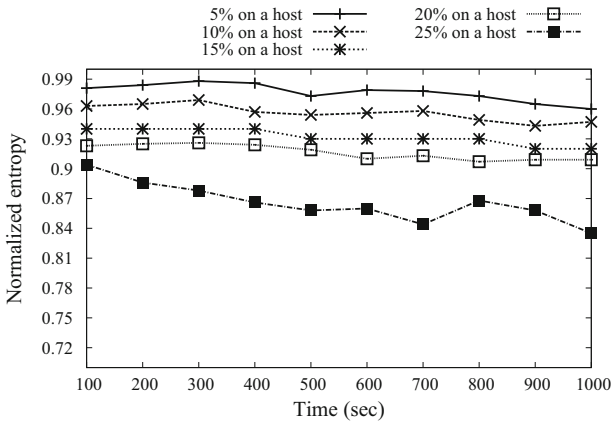


Fig. 4 Distribution of normalized entropy with varying load on victim host

Algorithm 1 Threshold estimation

1: **Input:** $< E_i >$
2: **Output:** Δ

START: THRESHOLD$< E_i >$
3: $\mu \leftarrow \sigma \leftarrow 0$
4: $\mu = \sum_{i=1}^{E_i.size()}(E_i)$
5: $\sigma = \sqrt{\frac{\sum_{i=1}^{E_i.size()}(E_i-\mu)^2}{E_i.size()}}$
6: $\lambda_0 = \mu - 3\sigma$
END

Time and space complexity Procedure threshold estimation requires a traversal of the entire size of $< E_i >$. Generalizing it, the algorithm incurs n steps for the execution, thus contributing a time complexity of $O(n)$. Local variables μ and σ require constant space, but $< E_i >$ being a list of size n contributes to a total space complexity of $O(n)$.

Finally, since the network traffic characteristics cannot remain similar all the time, we believe that the threshold entropy should be adaptive. Therefore, with the dynamism of traffic scenario, the point of threshold should change over the time. Hence, whenever any window's entropy is found to be greater than the threshold, we would regard that entropy as one of the normal entropy and put that entropy value into $< E_i >$, as this one indicates one of the benign entropy. Now, when the E_i is full, mean and standard deviations for the set $< E_i >$ is calculated. Hence, a new threshold will be provided to the system to proceed further. By doing this, we have read the characterization of the network traffic situation and act so.

4.2.2 Secondary detection phase

Algorithm 2 depicts the steps taken during secondary detection phase. As it can be seen from Algorithm 2 that the detection method uses a Packet Monitoring Table (OF_s^{table}) at each OF_s. It consists of $<Npckt^{Sip}, Npckt^{Dip}, Npckt^{MAC}, T, C_T>$ tuple. When a openflow switch (OF_s) receives a new data packet (i.e., N_{pckt}), it extracts the tuple $<Npckt^{Sip}, Npckt^{Dip}, Npckt^{MAC}>$, and the packet is sent to the controller. The $Npckt^{MAC}$ of the extracted tuple is searched in the OF_s^{table}. If a match is found, the value of C_T in the corresponding row is increased by one, if no match is found, then a new tuple is created, and a TIMER (T) of t second is associated with the newly created entry. The above process is done only for the N_{pckt}, thus it minimizes the control overhead of searching and storing new entries in the OF_s^{table}.

Algorithm 2 Second phase of "periodic monitoring" for *Resource Exhaustion* attack

Require: INPUTs at any OF-Switch (OF_s):
New data packet: N_{pckt}
N_{pckt}'s Source and Destination IPs: N_{pckt}^{Sip}, N_{pckt}^{Dip}
N_{pckt}'s MAC Address: N_{pckt}^{MAC}
Packet Monitoring Table at OF_s: OF_s^{table}
TIMER: T, TIMER counter: T_c, $R_{value} \in 10, 100$
Threshold value for parallel flows on a host (T_{flow}) = R_{value}
Counter value associated with each $T = C_T$, Average C_T: C_T^{avg}
Ensure: OUTPUT: Block hosts suspected as *RE* attackers

```
for Each N_pckt received at any OF_s do
    Extract {N_pckt^Sip, N_pckt^Dip, N_pckt^MAC}
    for Each entry in OF_s^table do
        if OF_s^table (N_pckt^MAC) ≡ N_pckt^MAC then
            Increment C_T
        else
            Create a new entry in OF_s^table
            Set T_c = C_T = C_T^avg = 0
            Associate T of t secs with the entry
        end if
    end for
end for
for Each expiry of T do
    if (C_T^avg > T_flow) ∧ (T_c > 2) then
        Send alarm_packet to controller
        Remove entry from OF_s^table
    else
        Increment T_c
        C_T^avg = C_T^avg + C_T
        Reset T, T_flow and C_T
    end if
end for
```

Upon the expiry of each T, the detector module compares the value of average C_T (i.e, C_T^{avg}) with the specified value of T_{flow}. Each time the value of T_{flow} is randomly selected between a set of values. If the C_T^{avg} generated by the host is more than T_{flow} in last n times,[2] each t second duration, the host is suspected as a *RE* attacker. For the suspected host, the second phase of the "periodic monitoring" technique will be performed before sending an *alarm_packet* to the controller.

In the second phase, the detection method uses multiple parameters which include, a time interval (t), number of mice[3] flows (F_{mice}), mice flow threshold (T_{mice}), and a probabilistic parameter which is chosen by the network administrator. We choose this probabilistic parameter in the same way we choose the threshold entropy in our primary phase. Using these mice values, an RF attack detection parameter (RF_{det}) is calculated at the OF-switch using the following Eq. 12.

$$RF_{det} = \frac{F_{mice} - T_{mice}}{F_{mice}}. \tag{10}$$

To avoid blocking the benign users that send more traffics containing the mice flows, we have considered a probabilistic parameter say $P \in \{0, 1\}$, and it is set by the administrator. The detection unit periodically counts the number of mice flows for each host. The edge OF_s detects a host that sends more than F_{mice} per t second, and it calculates RF_{det} value for that host. If RF_{det} is greater than P, then it informs the controller to block the host. To better understand the approach, let's take an example. Suppose that $t = 5$ s, $P = 0.5$ and $T_{mice} = 10$ are the assigned values to these parameters for the detection unit on the edge OF_s. In this scenario, assume that there are two users, one benign and the other is attacker. The benign user generates 15 mice flows, and the attacker generates 25 mice flows per t. For these two users, the RF_{det} values will be as follows:

$$RF_{det}^{benign} = \frac{15 - 10}{15} = 0.3, \tag{11}$$

$$RF_{det}^{attacker} = \frac{25 - 10}{25} = 0.6. \tag{12}$$

Since $RF_{det}^{attacker}$ is greater than $P = 0.5$, the detection unit informs the controller about the attacker. According to Eq. 12, it can be concluded that if the attackers generate a high volume of mice flows, the probability of blocking them is increased. In particular, this phase can suspect a host as an attacker who generates mice flows greater than below the equation:

$$F_{mice} > \frac{T_{mice}}{1 - P}. \tag{13}$$

It is worth to mention that using probabilistic parameter such as above for attack detection is a conventional

[2] Use of lower values for n might increase the number of false positives, and use of higher values will increase the detection time for the attack.

[3] A mice flow is a data flow that contains less than 3 packets.

approach and have used by various literature such as [23, 31, 32] to name a few. Our proposed "periodic monitoring" technique only notifies the controller by sending an *alarm_packet*, if a host is suspected an attacker in both the aforementioned detection phases. Upon reception of *alarm_packet*, the control detects (by using Host Profile data structure) the host machine and install blocking rules on OF_s for the suspected attacker node.

5 Simulation and result analysis

In this section, we present the details of the experimental setup, which we use to analyze the performance of our proposed novel attacks and their respective countermeasures. In Sect. 5.2, we present the impact of Route Spoofing attack on various SDN scenarios along with the effectiveness of our proposed countermeasure by considering various network metrics. Similarly, the impact of Resource Exhaustion attack and its countermeasure is presented in Sect. 5.3.

5.1 Evaluation setup

We use Mininet [33] as an emulator for implementing all the target scenarios along with OpenDaylight [34] as a network controller. As described in the previous section, both the countermeasures consists of two units (i.e., the detection unit, implemented at OF-Switches, and the prevention unit, implemented on the controller) both are installed as a plugin into the software OpenFlow reference switch [35]. All experiments are executed over on a machine with the following configuration: CPU Intel Core i7-4700MQ-2.4 GHz and 6 GB RAM. The impact of both the attacks and the effectiveness of their countermeasures have been investigated over similar topologies. The target SDN topology consists of 9 OF-switches, 12 hosts (8 benign and four attackers), and one controller. The source-destination pairs are selected randomly in the network scenario. The link bandwidth is set to 100 Mbps, and the delay for each link is set to 5 ms. Simulation time is set to 500 s.

5.2 Route spoofing attack

In the scenario that we use to analyze *RS* attack, we implemented two separate Java applications to configure source and destination nodes as UDP client and server. There are four benign senders who send traffic to 4 receivers. Each benign sender runs the UDP client application to send 5–30 packets of size 512 bytes per second to the benign receiver. To perform the attack, we use Hping [36] that generates spoofed IP addresses. Each attacker sends UDP packets of size 1000 bytes with different rates using the Hping tool. To distribute the attack effect over the network, the first attacker starts to send packets after the 40th second, and the remaining attackers start with the interval of 20 s. The spoofed packets go along the existing path between the benign sender and receiver without the intervention of the controller.

We use the following metrics to shows the impact of the attack and the effectiveness of our proposed countermeasure.

- *Packet delivery ratio* (ρ) This metric will indicate the data packets that are dropped in intermediate switches due to a buffer overflow caused by the high-speed attacker traffic over the spoofed routes.
- *Average end-to-end delay* (Δ) This metric will show how the attacker's traffic is introducing various delays in the genuine user traffic, such as delay in forwarding data packets at OF-switches.
- *Network bandwidth consumption* (β) This metric denotes the impact of attacker added traffic on the bandwidth consumption at data plane.

Figure 5 shows the effect on ρ with the increase in attack rate for normal scenario, for *RS* attack (*RS-A*), and for *RS* countermeasure (*RS-C*). As depicted in Fig. 5 that when the rate of malicious packets increases the ρ decreases for benign users. This is because the malicious traffic causes congestion over the network links. But with the use of our proposed countermeasure, the malicious users have been detected and blocked. Thus ρ does not change much for benign users.

Figure 6 shows the effect on Δ with the increase in attack rate for normal, *RS-A* and *RS-C*. It is evident that increasing the rate of malicious packets lead to congestion over network links as well as on the OF_S forwarding queues. Thus, it incurs long delays for data packets. We

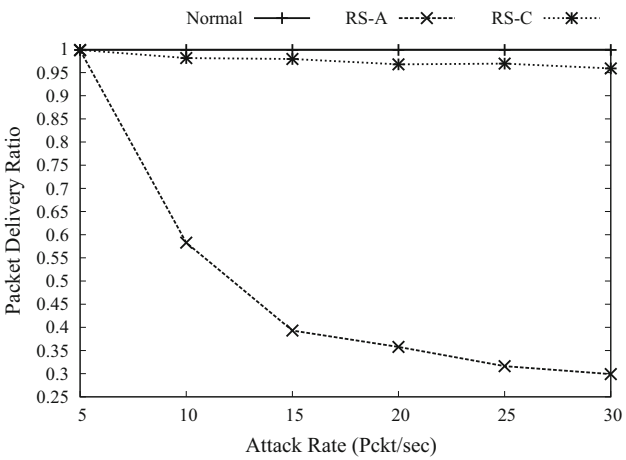


Fig. 5 Packet delivery ratio (ρ) with increased malicious packet rate

can see in Fig. 6 that for RS-A the delay starts decreasing after 15 Pckt/s. It is because the Δ becomes saturated around the point where attacker rate approaches to 15 Pckt/s, and the further increase in attack rate will not affect the Δ in a significant way. It has been observed that at higher attack rates the attack packets were started to cancel the malicious effect of each other, because the controller is already busy with the processing of the previously received attack packets. In the presence of our proposed countermeasure, the malicious packets will no longer be present in the network, and this prevents longer delays in data delivery. Figure 5 shows simulation results for average β in *RS* for the three comparing techniques. As it can be seen from Fig. 7, the presence of a large number of malicious packets in the network due to *RS* attack leads to higher bandwidth consumption. The proposed countermeasure decreases β due to the reasons above.

Figures 8 and 9 shows the simulation results for β with the simulation time in the absence and in the presence of the our proposed countermeasure. These figures helps to properly investigate the behavior of *RS* attack and our countermeasure in real time. Figure 9 shows that using countermeasure technique leads to preventing malicious packets injection to the network, thus results in lower β.

5.3 Resource exhaustion attack

In this scenario, we have configured the application layer for hosts that act as source or destination in the network. We also configured few cases where a host can use multiple IPs associated with the same MAC address. This is done to depict situations where multiple flows are setup between two hosts. We use a scenario where six senders initiate two new requests with different destination IP per second. When the controller receives these requests, it calculates a route and installs forwarding rules at OF_s. To configure the attackers, the same script is used where four attackers generate TCP (for SYN flooding attack) for a target server and UDP packets with different destination IP

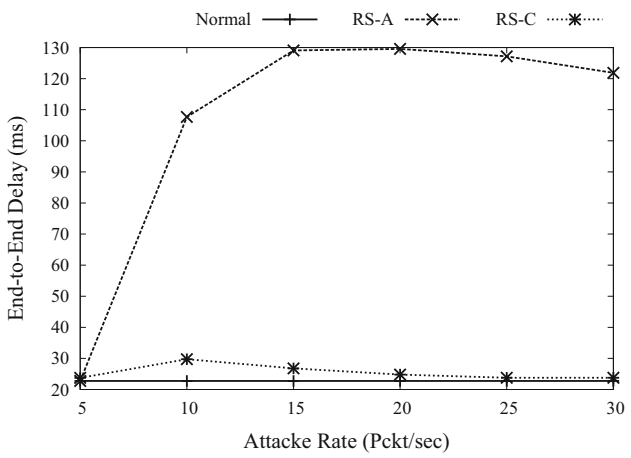


Fig. 6 End to end delay (Δ) with increased malicious packet rate

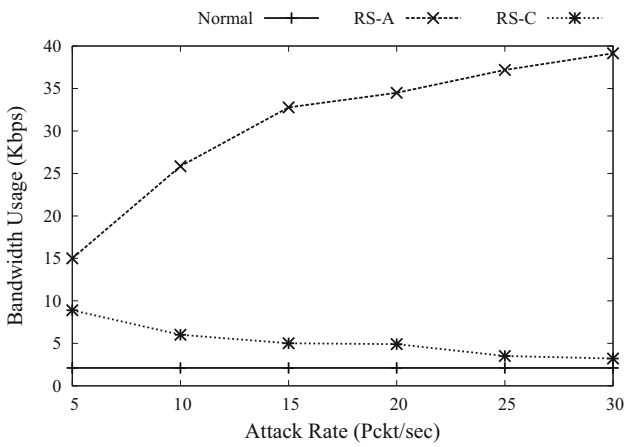


Fig. 7 Bandwidth consumption (β) with increased malicious packet rate

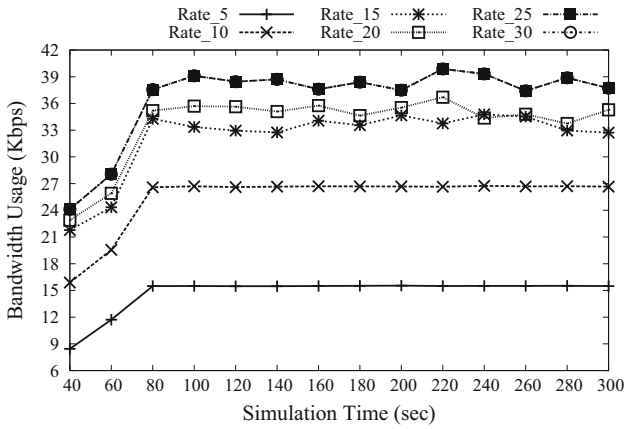


Fig. 8 Bandwidth consumption (β) with simulation time without countermeasure

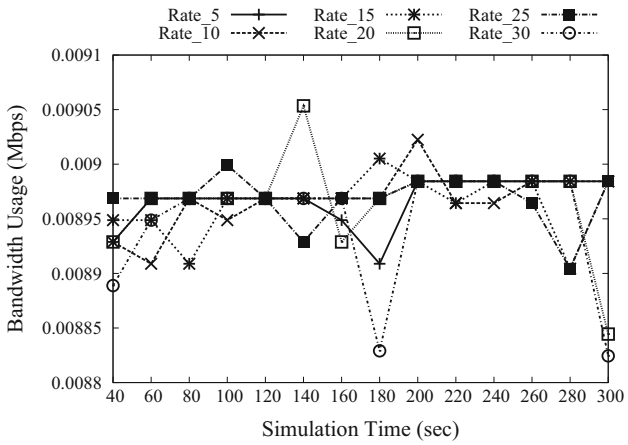


Fig. 9 Bandwidth consumption (β) with simulation time with countermeasure

addresses. DITG (Distributed Internet Traffic Generator) is used to generate the legitimate traffic. The legitimate traffic consists of 70% TCP data and 30% UDP data. The Center for Applied Internet Data Analysis (CAIDA) has released "DDoS Attack 2007" Dataset containing the attack traffic to the victim nodes. We use this dataset for our experiments and use the *Scapy* (Python tool) to generate DDoS flooding attack traffic from bots to the victim node.

We start our analysis using the results gathered for anomaly detection that is performed by the primary phase of our "periodic monitoring" module. We evaluate the performance of our entropy-based primary detection module against the TCP SYN flooding attack and compare it with recent state-of-the-art (i.e., SLICOTS [8]).

To compare our entropy based detection (EBD) technique with SLICOTS, we use the target scenario which is similar to SLICOTS paper. Specifically, we perform the comparison for scalability and sensitivity metrics. In the target scenario ($S1$), the attackers generate malicious SYN packets with a rate ranging from 50 to 350 packets per second (upper bound is due to hardware limitations). The number of malicious and benign hosts are set to 120 and 2, respectively. For EBD technique, the values of $\delta t = 6$ (based on Fig. 10) and $M = 3$, and the threshold entropy λ is calculated dynamically using Algorithm 1, while for SLICOTS, the value of threshold (K) is varied between 10 and 100 [8] (Fig. 11). For further discussion on this trade-off, in Fig. 12, we show the receiver operating characteristic (ROC) curve for both the comparing protocols. The FPR (or 1-specificity) is the percentage of malicious SYN packets which are incorrectly identified as benign SYN packets, and the TPR (or sensitivity) is the percentage of benign packets which are correctly identified as benign packets. These metrics are measured under DDoS TCP SYN flooding attacks to one victim host. As it is seen in Fig. 12, the EBD reaches to 100% TPR at the point when it has approximately 13% FPR. On the other hand, the SLICOTS results in nearly 20% FPR until it reaches to 100% TPR. The EBD has 7% low FPR than SLICOTS because unlike SLICOTS which uses fixed threshold values to identify the fake SYN requests generated by an adversary, the EDB uses adaptive threshold entropy calculation procedure. Moreover, the randomness check performed in EDB uses a carefully selected set of parameters (i.e., source and destination IP addresses, destination port, OF switch input port, and TCP flags), which are absent in SLICOTS.

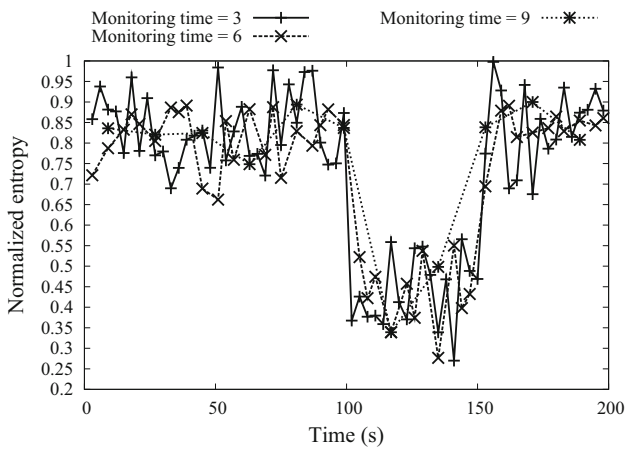


Fig. 10 Normalized entropy values with different Δt

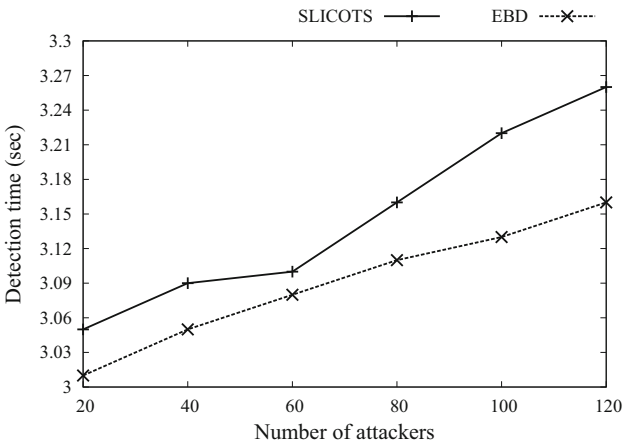


Fig. 11 Attack detection time with increasing number of attackers

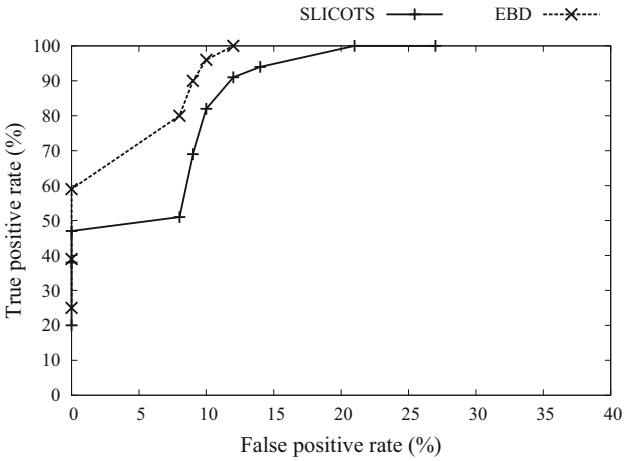


Fig. 12 ROC curve comparison between SLICOTS and EBD

Figure 11 shows the attack detection time with increasing number of attackers for SLICOTS and EBD techniques. As it can be seen in Fig. 11, the detection time remains nearly same for both the methods because both solutions use the similar type of constant value(s) as a threshold for blocking the attackers. However, the detection time for EBD is slightly lower due to its threshold entropy calculation process which is dynamic and adaptive. The detection time for EBD will be more than SLICOTS if

it is followed by the secondary detection phase of our periodic monitoring module, although this will also yield lower false positives and negatives. Hence, a trade-off between detection time and precision could be used depending on the application scenario.

The rate of generating of different request packets for attackers is kept much higher when compared to benign users. As we mentioned in Section III-B, the detection unit sends an alarm to the controller if, it finds an attacker who sends more than T_f new flows in t s. Indeed, choosing a large value for these parameters would result in a delay in the attack detection, and thus scarce efficiency. On the contrary, by choosing a low value for these parameters would cause a too early detection with a higher false positive rate. For all the scenarios, unless specified otherwise, we have set the threshold $T_{mice} = 10$, $t = 1$ (i.e., the detection unit monitors flows after every second) and $P = 0.3$. We consider mice flow as a flow which has lower than three packets. The metrics of interest for this attack are as follows: a) *Average number of entries in the* OF_s (η), and *benign request processing time* (τ).

Figure 13 shows simulation results for η for normal scenario (i.e.,pure), for *RE* attack (*RE-A*) and for *RE* countermeasure (*RS-C*). As it can be seen in Fig. 13 that the number of entries in the forwarding table of an OF_s increases rapidly due to the *RE* attack. This increase in entries not only causes the lookup delay for forwarding packets but, it also makes the OF_s to overwrite the benign user data flow rules in the forwarding tables due to insufficient memory. Moreover, the Fig. 8 also shows that by using our proposed technique, the controller blocks the suspect attackers and as a result, the number of entries stored in OF_s decreases significantly.

Figure 14 shows simulation results for τ with an increase in the rate of malicious packets in the network for pure, *RE-A* and *RS-C*. As shown in Fig. 14, the processing time of new benign requests increases with the presence of malicious packets in the network. This is because in the *RE* attack, the data to control plane link, as well as the controller, becomes saturated due to a large number of requests coming from OF_s and as a result, the controller will not be able to process new requests immediately. By implementing our countermeasure, the attackers will be blocked and processing time for new requests will be decreased.

RE attack detection time and precision with varying values of the probabilistic parameter P, some mice flows per second (F_{mice}), mice flow threshold (T_{mice}), as the second phase of our proposed "periodic monitoring" technique has these three adjustable parameters. The Fig. 15 shows results for the attack detection time with increasing T_{mice} values for various P values. For the results in Fig. 15, we consider a fixed value of 20 flows for F_{mice}. As it can be seen from the Fig. 15, as expected the attack detection time increases with increase in T_{mice} and P. Although, the increase in detection time with lower P comes with lower detection precision as few honest users will also be detected as attacker due to low probability values. In particular, we can conclude that choosing large values for P and T_{mice} leads to longer delays in attack detection procedure, while their too lower values cause an increase in false positives. The above reasons also hold true for the detection precision results shown in Fig. 16.

6 Conclusions

In this paper, we investigated two security vulnerabilities in SDN, and we implemented effective countermeasures to prevent the same. We have shown that an attacker, with minimal effort, can launch attacks such as *Route Spoofing* and *Resource Exhaustion*, which causes significant harm to the data communication process in the SDN. To effectively handle these attacks, we propose lightweight and efficient

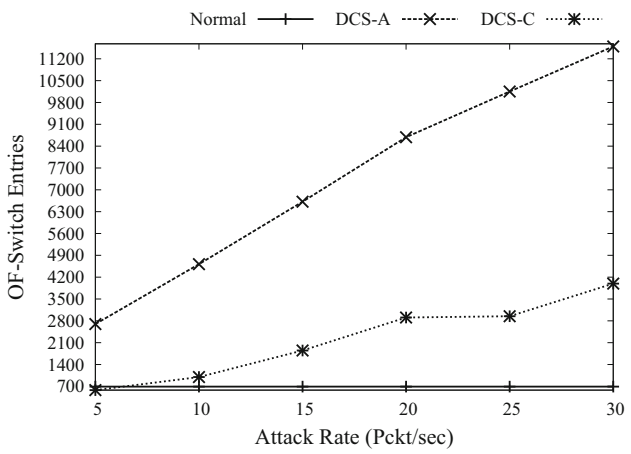


Fig. 13 Number of entries in the OF_s with malicious packet rate

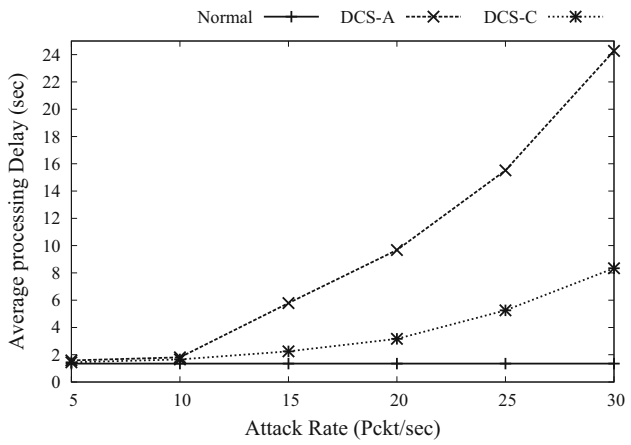


Fig. 14 Benign request processing time with malicious packet rate

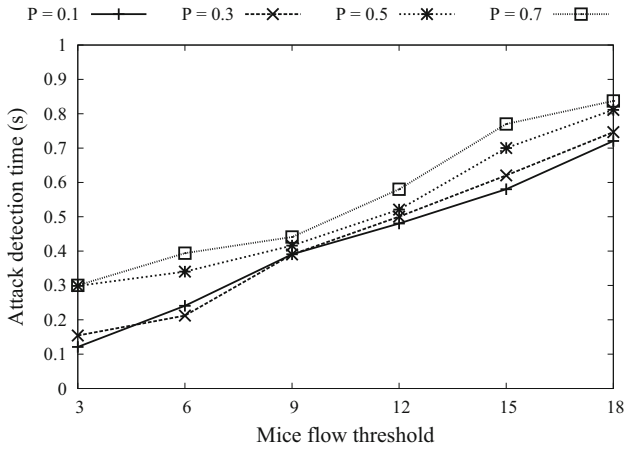


Fig. 15 Attack detection time with increasing mice threshold for various P values

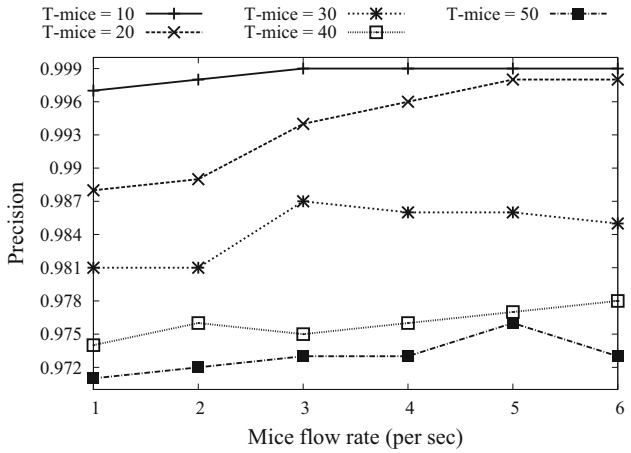


Fig. 16 Detection precision with varying mice rate

solutions that detect and block the attacker in its initial phase of an attack. Simulation results obtained for the relevant metrics shows the feasibility of implementation and correctness of the proposed countermeasures. Our proposed solution monitors ongoing traffics behavior on data plane and informs the controller in case of any security policy violations, and the controller takes adequate action to isolate the detected malicious nodes from the network. We have observed that when both the detection phases in "periodic monitering" works in parallel, the number of attacks detected are higher, but the precision is a bit lower when compared to the scenarios where the secondary phase is executed following a primary detection phase. However, in the later, the detection time increases.

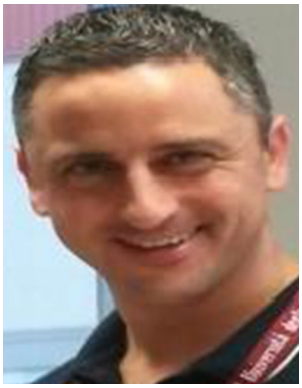
As a future work, we would like to investigate current and improved solutions for cases where a stronger adversary exists in the network. For such cases, we will explore solutions that include probabilist and statistical techniques for detecting the attacks in more effective and accurate manner.

Acknowledgements Chhagan Lal and Mauro Conti are supported in part by EU LOCARD Project under Grant H2020-SU-SEC-2018-832735, and in part by Huawei Project "Secure Remote OTA Updates for In-Vehicle Software Systems" under Grant HIRPO 2018040400359-2018. The work of M. Conti was supported by the Marie Curie Fellowship through European Commission under Agreement PCIG11-GA-2012-321980.

References

1. Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, *103*(1), 14–76.
2. Conti, M., Kaliyar, P., & Lal C. (2018). CENSOR: Cloud-enabled secure IoT architecture over SDN paradigm. *Concurrency and Computation: Practice and Experience*, *31*(8), e4978.
3. Du, J., Gelenbe, E., Jiang, C., Zhang, H., & Ren, Y. (2017). Contract design for traffic offloading and resource allocation in heterogeneous ultra-dense networks. *IEEE Journal on Selected Areas in Communications*, *35*(11), 2457–2467.
4. Yan, Q., Yu, F. R., Gong, Q., & Li, J. (2016). Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE Communications Surveys & Tutorials*, *18*(1), 602–622.
5. Wickboldt, J. A., Jesus, W. P. D., Isolani, P. H., Both, C. B., Rochol, J., & Granville, L. Z. (2015). Software-defined networking: Management requirements and challenges. *IEEE Communications Magazine*, *53*(1), 278–285.
6. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., et al. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Computer Communication Review*, *38*(2), 69–74.
7. Zhang, P., Wang, H., Hu, C., & Lin, C. (2016). On denial of service attacks in software defined networks. *IEEE Network*, *30*(6), 28–33.
8. Mohammadi, R., Javidan, R., & Conti, M. (2017). Slicots: An sdn-based lightweight countermeasure for tcp syn flooding attacks. *IEEE Transactions on Network and Service Management*, *14*, 487–497.
9. Wang, R., Jia, Z., & Ju, L. (2015). An entropy-based distributed DDoS detection mechanism in software-defined networking. In *2015 IEEE Trustcom/BigDataSE/ISPA* (pp. 310–317). Helsinki.
10. Kalkan, K., Gur, G., & Alagoz, F. (2017). Defense mechanisms against ddos attacks in sdn environment. *IEEE Communications Magazine*, *55*(9), 175–179.
11. Zheng, J., Li, Q., Gu, G., Cao, J., Yau, D. K. Y., & Wu, J. (2018). Realtime ddos defense using cots sdn switches via adaptive correlation analysis. *IEEE Transactions on Information Forensics and Security*, *13*(7), 1838–1853.
12. Kang, M. S., Lee, S. B., & Gligor, V. D. (2013). The crossfire attack. In *2013 IEEE symposium on security and privacy* (pp. 127–141).
13. Mohammadi, R., Javidan, R., Keshtgary, M., Conti, M., & Lal, C. (2017). Practical extensions to countermeasure dos attacks in software defined networking. In *2017 IEEE conference on network function virtualization and software defined networks (NFV-SDN)* (pp. 1–6).

14. François, J., Dolberg, L., Festor, O., & Engel, T. (2014). Network security through software defined networking: A survey. In *Proceedings of the conference on principles, systems and applications of IP telecommunications*, ser. IPTComm '14. ACM (pp. 6:1–6:8).
15. Fayaz, S. K., Tobioka, Y., Sekar, V., & Bailey, M. (2015). Bohatei: Flexible and elastic ddos defense. In *24th USENIX security symposium (USENIX Security 15)* (pp. 817–832). Washington, DC: USENIX Association.
16. Rebecchi, F., Boite, J., Nardin, P., Bouet, M., & Conan, V. (2017). Traffic monitoring and ddos detection using stateful sdn. In *2017 IEEE conference on network softwarization (NetSoft)* (pp. 1–2).
17. Chen, C., Chen, Y., Lu, W., Tsai, S., & Yang, M. (2017). Detecting amplification attacks with software defined networking. In *2017 IEEE conference on dependable and secure computing* (pp. 195–201).
18. D'Cruze, H., Wang, P., Sbeit, R . O., & Ray, A. (2018). A software-defined networking (SDN) approach to mitigating DDoS attacks. In S. Latifi (Ed.), *Information technology—New generations*. Cham: Springer.
19. Dhawan, M., Poddar, R., Mahajan, K., & Mann, V. (2015). Sphinx: Detecting security attacks in software-defined networks. In *NDSS*.
20. Felipe, A., Piedrahita, M., Rueda, S., Mattos, D. M. F., Carlos, O., & Duarte, M. B. (2015). Flowfence: A denial of service defense system for software defined networking. In *Global information infrastructure and networking symposium (GIIS)*.
21. Shin, S., Yegneswaran, V., Porras, P., & Gu, G. (2013). Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on computer & communications security*, ser. CCS '13.
22. Sungmin, L. Hong, Xu, H., Wang, G., & Gu (2015). Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *NDSS*.
23. Ambrosin, M., Conti, M., Gaspari, F. D., & Poovendran, R. (2017). Lineswitch: Tackling control plane saturation attacks in software-defined networking. *IEEE/ACM Transactions on Networking*, *25*(2), 1206–1219.
24. Wang, H., Xu, L., & Gu, G. (2015). Floodguard: A dos attack prevention extension in software-defined networks. In *Proceedings of the 2015 45th annual IEEE/IFIP international conference on dependable systems and networks*, ser. DSN '15.
25. Chin, T., Mountrouidou, X., Li, X., & Xiong, K. (2015). Selective packet inspection to detect dos flooding using software defined networking (sdn). In *2015 IEEE 35th international conference on distributed computing systems workshops* (pp. 95–99).
26. Park, Y., Chang, S. Y., & Krishnamurthy, L. M. (2016). Watermarking for detecting freeloader misbehavior in software-defined networks. In *2016 International conference on computing, networking and communications (ICNC)* (pp. 1–6).
27. Liu, J., Lai, Y., & Zhang, S. (2017). Fl-guard: A detection and defense system for DDoS attack in sdn. In *Proceedings of the 2017 international conference on cryptography, security and privacy*, ser. ICCSP '17 (pp. 107–111) ACM. [Online]. Available: http://doi.acm.org/10.1145/3058060.3058074.
28. Wang, T., & Chen, H. (2017). Sguard: A lightweight sdn safeguard architecture for dos attacks. *China Communications*, *14*(6), 113–125.
29. Guo, F., & Chiueh, T-c. (2006). *Sequence number-based MAC address spoof detection* (pp. 309–329). Berlin: Springer.
30. Kumar, P., Tripathi, M., Nehra, A., Conti, M., & Lal, C. (2018). SAFETY: Early detection and mitigation of TCP SYN flood utilizing entropy in SDN. *IEEE Transactions on Network and Service Management*, *15*(4), 1545–1559.
31. Zhu, H., Du, S., Gao, Z., Dong, M., & Cao, Z. (2014). A probabilistic misbehavior detection scheme toward efficient trust establishment in delay-tolerant networks. *IEEE Transactions on Parallel and Distributed Systems*, *25*(1), 22–32.
32. Wang, S., Zhang, Z., & Kadobayashi, Y. (2013). Exploring attack graph for cost-benefit security hardening: A probabilistic approach. *Computer Security*, *32*, 158–169.
33. http://www.mininet.org/.
34. http://www.opendaylight.org.
35. Openflow.org. openflow switching reference system. http://www.openflow.org/wp/downloads/.
36. http://www.hping.org.



Mauro Conti is Full Professor at the University of Padua, Italy, and Affiliate Professor at the University of Washington, Seattle, USA. He obtained his Ph.D. from Sapienza University of Rome, Italy, in 2009. After his Ph.D., he was a Post-Doc Researcher at Vrije Universiteit Amsterdam, The Netherlands. In 2011 he joined as Assistant Professor the University of Padua, where he became Associate Professor in 2015, and Full Professor in 2018. He has been Visiting Researcher at GMU (2008, 2016), UCLA (2010), UCI (2012, 2013, 2014, 2017), TU Darmstadt (2013), UF (2015), and FIU (2015, 2016). He has been awarded with a Marie Curie Fellowship (2012) by the European Commission, and with a Fellowship by the German DAAD (2013). His research is also funded by companies, including Cisco and Intel. His main research interest is in the area of security and privacy. In this area, he published more than 200 papers in topmost international peer-reviewed journals and conference. He is Area Editor-in-Chief for IEEE Communications Surveys and Tutorials, and Associate Editor for several journals, including IEEE Communications Surveys and Tutorials, IEEE Transactions on Information Forensics and Security, and IEEE Transactions on Network and Service Management. He was Program Chair for TRUST 2015, ICISS 2016, WiSec 2017, and General Chair for SecureComm 2012 and ACM SACMAT 2013. He is Senior Member of the IEEE.

Chhagan Lal is Postdoc fellow in Department of Mathematics, University of Padua, Italy and Affiliate Associate Professor at Manipal University Jaipur, India. He obtained his Bachelors in Computer Science and Engineering from MBM Engineering College, Jodhpur, India in 2006. He obtained his Masters degree in Information Technology with specialization in Wireless communication from Indian Institute of Information Technology, Allahabad in 2009, and Ph.D. in Computer Science and Engineering from Malaviya National Institute of Technology, Jaipur, India in 2014. He has been awarded Canadian Commonwealth scholarship in 2012 under Canadian Commonwealth Scholarship Program to work in University of Saskatchewan in Saskatoon, Saskatchewan, Canada. His current research areas include Blockchain Analysis, Security in Wireless networks, Software-

defined networking, Underwater acoustic networks, and context based security solutions for Internet of Things (IoT) networks.

Reza Mohammadi is currently working as an Assistant Professor in Department of Computer Engineering, Bu-Ali Sina University, Hamedan, Iran. He received the M.Sc. degree in computer networks from the Shiraz University of Technology in 2013, his Ph.D. degree in computer networks from Shiraz University of Technology, Shiraz, Iran. He has several publications in international conferences and journals regarding Underwater Wireless Sensor Networks (UWSNs). He currently focused on software defined networks (SDNs) as a new trend in computer networks. His major fields of interest are SDNs, heuristic algorithms, performance evaluation, UWSNs, ad hoc networks, and underground wireless sensor networks.

Umashankar Rawat is currently a Professor with the Department of CSE, Manipal University Jaipur. He received the Ph.D. degree in computer science and engineering from the Jaypee University of Engineering and Technology, Guna, in the area of linux file system security, in 2013, and the M.E. degree in computer engineering from Shri Govindram Seksaria Institute of Technology and Science, Indore, in 2003. He is having 18 years of teaching experience. His current research interests include security issues in Software Defined Networks, information systems security, and distributed systems.