Defence Technology 25 (2023) 76–87

Contents lists available at ScienceDirect

Defence Technology

journal homepage: www.keaipublishing.com/en/journals/defence-technology

IQR-based approach for DDoS detection and mitigation in SDN

Rochak Swami [a, *], Mayank Dave [a], Virender Ranga [b]

[a] Department of Computer Engineering, National Institute of Technology, Kurukshetra, India
[b] Department of Information Technology, Delhi Technological University, Delhi, India

ARTICLE INFO

Article history:
Received 22 July 2022
Received in revised form
26 September 2022
Accepted 11 October 2022
Available online 22 October 2022

Keywords:
SDN
DdoS
IQR
Controller
CPU utilization
Packet_in

ABSTRACT

Software-defined networking (SDN) is a trending networking paradigm that focuses on decoupling of the control logic from the data plane. This decoupling brings programmability and flexibility for the network management by introducing centralized infrastructure. The complete control logic resides in the controller, and thus it becomes the intellectual and most important entity of the SDN infrastructure. With these advantages, SDN faces several security issues in various SDN layers that may prevent the growth and global adoption of this groundbreaking technology. Control plane exhaustion and switch buffer overflow are examples of such security issues. Distributed denial-of-service (DDoS) attacks are one of the most severe attacks that aim to exhaust the controller's CPU to discontinue the whole functioning of the SDN network. Hence, it is necessary to design a quick as well as accurate detection scheme to detect the attack traffic at an early stage. In this paper, we present a defense solution to detect and mitigate spoofed flooding DDoS attacks. The proposed defense solution is implemented in the SDN controller. The detection method is based on the idea of an statistical measure — Interquartile Range (IQR). For the mitigation purpose, the existing SDN-in-built capabilities are utilized. In this work, the experiments are performed considering the spoofed SYN flooding attack. The proposed solution is evaluated using different performance parameters, i.e., detection time, detection accuracy, packet_in messages, and CPU utilization. The experimental results reveal that the proposed defense solution detects and mitigates the attack effectively in different attack scenarios.

1. Introduction

In the generation of rapid deployment of cloud infrastructure for demanding applications, traditional network switching architectures are inefficient. This is because these architectures are difficult to reconfigure and program for changing requirements. The traditional switching networks are based on a strong coupling of data and control planes as these planes are tightly integrated with each other. Data plane include different forwarding devices such as routers, switches, etc. The control plane is responsible for implementing control logic for the forwarding decisions taken by the data plane devices. The network management tasks in these networks are complex, time-consuming, and costly. For example, in case of requiring an update or adding a new policy in the existing mechanisms, the essential modifications are to be made on all the forwarding devices in the entire network. Software-defined networking (SDN) technology has been developed recently to address such challenges. SDN enables the flexibility and automation in network configuration for current and future networks in order to improve the performance of the networks. SDN offers a centralized network intelligence in one network component (i.e., control plane) by separating the control logic from forwarding devices [1,2]. In SDN, the complete control logic is employed in the control plane that may consist of one or more controllers [3]. The functions of the controller in SDN are to manage all the forwarding devices and to take the routing decisions. The communication between controller and forwarding devices is offered by an standard protocol such as OpenFlow [2,4,5]. OpenFlow was developed by Open Net-working Foundation (ONF), and came into existence in 2008 [6]. These properties of SDN provide a programmable, cost effective, and less time-consuming mechanism over traditional networks [7]. However, this effective and intelligent technology also introduces several security vulnerabilities that already exist in

* Corresponding author.
E-mail addresses: rochakswami123@gmail.com, rochak_6170016@nitkkr.ac.in (R. Swami).

Peer review under responsibility of China Ordnance Society

https://doi.org/10.1016/j.dt.2022.10.006

traditional networks [8]. The adversaries exploit vulnerabilities of traditional networks to launch various attacks. One of such popular attacks is known as Distributed denial-of-service (DDoS) attack [9].

DDoS is considered as one of the most disastrous attacks that target government and business organizations. It aims to disrupt the normal functioning of the network by sending a massive amount of network traffic towards the victim and makes the normal users suffer from the unavailability of the service or network's resources [10–14]. DDoS attacks are growing rapidly with time. The prime goal of DDoS attacks is to exhaust or overload the resources of the victim machine. DDoS attacks attempt to violate the availability of the system. In SDN, since controller is the most sensitive and intelligent entity of the complete network, it is, therefore, the most favorite target point of the attackers [15,16]. Although, DDoS attack also affects switch and switch-controller channel, which creates switch's flow table overflow and switch-controller bandwidth consumption. The controller exhaustion can collapse the entire SDN network. By default, if the switch receives a new packet and does not find any matching flow rule in its flow tables, a message is forwarded to the controller in the form of packet_in message for taking an appropriate routing decision. The controller then sends a flow modification rule (Flow_Mod) to the switch to update its flow table as per the latest received flow rules. This Flow_Mod rule avoids the transferring of packet_in messages coming from the same source to the controller. This is because switch can take decisions by following the rules placed in flow table. In case of spoofed flooding DDoS attack, the attacker sends a large number of packets from various random IP addresses to the switch. When the switch does not find any matching rule in the flow tables for the incoming packet, a large number of packet_in messages is sent to controller for handling. This flooding of packets intends to exhaust the controller's resources, which creates a disruption to use the controller's services for normal users. Since the controller manages the complete network, DDoS attacks on the controller can jeopardize the working of the entire SDN network. Therefore, a defense solution against spoofed flooding DDoS attacks for SDN is necessary to be designed, which can provide timely detection and mitigation of the attacks. Various defense solutions to defend against such DDoS attacks have been proposed for SDN in the literature [17–28]. Most of the presented solutions are classified into two categories, i.e., statistical-based and machine learningbased. Statistical-based solutions are more preferable in the aspect of performance over machine learning-based solutions [29]. This is because machine learning methods create a heavy computational burden and have challenging data requirements. Thus, statistical-based solutions are more favorable for real-time scenarios and motivate us to work on designing an statistical-based defense solution. Therefore, in this research work, an statistical measure —Interquartile Range (IQR) based defense solution is proposed to detect the spoofed flooding DDoS attacks. IQR is based on the idea of placing the data values in different quartiles. Data values that violate the IQR-rule are considered as outliers. These outliers indicate a situation of suspicious network traffic and the respective source hosts of suspicious traffic are marked as attackers. SDN is capable enough to take appropriate defensive action if any malicious behaviour is detected. Thus, SDN-in-built functionalities are utilized to mitigate the attack after the detection of attackers. SYN flooding attack is one of the most used DDoS attacks as it can target the bandwidth, memory, and CPU of the controller in case of a large number of requests. This is the reason that we selected this attack to evaluate the proposed solution in this work. In the proposed work, SYN flooding attack is launched with a number of random spoofed IP addresses as illustrated in Fig. 1. This type of DDoS attack affects the controller with more dangerous effects than a non-spoofed attack due to flooding from a large number of sources. An analysis related to this study has been presented in Ref. [30].

The key contributions of the paper are summarized as follow:

- We propose and implement a defense solution to detect and mitigate spoofed SYN flooding attack in SDN. The detection method is based on IQR measure, which is applied on the received *packet_in* messages at a time interval followed by attacker detection. After attack detection, the mitigation is performed by blocking the identified attackers.
- We perform the experimental assessment considering different attack scenarios. The performance of the detection method

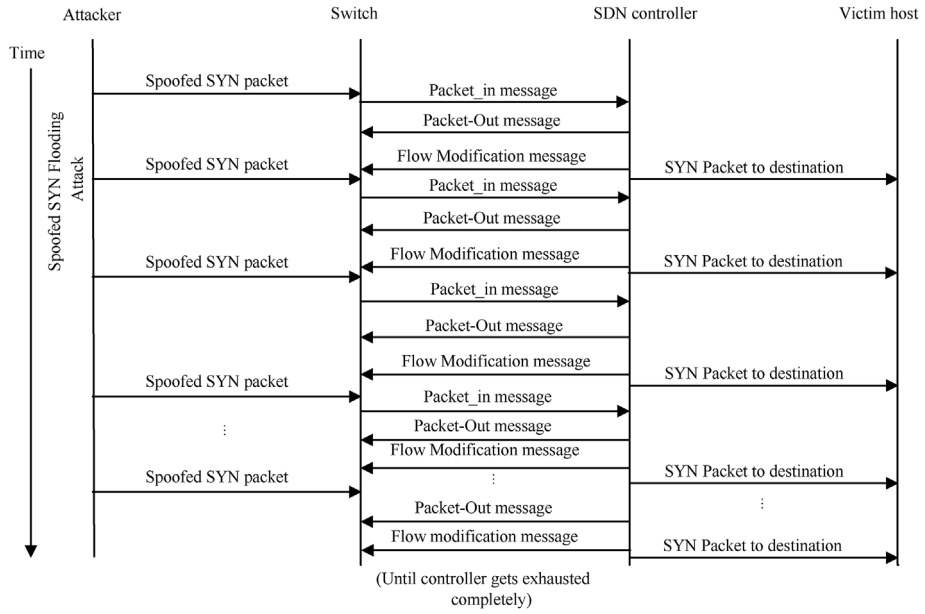

Fig. 1. Spoofed SYN flooding attack in SDN.

(IQR) is evaluated in terms of detection time, detection accuracy in respect to the attacker's identification.
- The performance evaluation of the mitigation module is analyzed in terms of CPU utilization of controller and *packet_in* messages in three cases, i.e., with no attack, under attack without defense, and with defense.

The remaining paper is organized in different sections as follows. Section 2 provides an overview of the existing SDN-based DDoS detection and mitigation mechanisms. Section 3 discusses our proposed defense solution, including its important modules. Section 4 introduces the simulation setup used for the experiments and presents an explanation of the evaluation results. The paper is concluded in Section 5.

2. Related work

In this section, relevant existing research works to defend against flooding DDoS attacks present in the literature are discussed. Flooding attacks are the most frequent DDoS attacks that slow down or crash the services by exhausting the resources of a system. SYN flooding attack is the most attempted DDoS attack among all the flooding attacks. Various defense solutions have been proposed and suggested in recent years to analyze the impact of DDoS attacks by the researchers. A few defense solutions are based on SDN specific network traffic and implemented in the SDN components. Mohammadi et al. [36] proposed a defense mechanism “SLICOTS" to defend the SDN network against SYN flooding attack. The detection logic is deployed in the controller. The proposed mechanism is implemented in OpenDaylight controller as an extension module. As per the proposed mechanism, a record is maintained with the counts of different types of TCP flags for a particular host. When the count of the records of respective host reaches threshold value, the malicious packet is dropped, and a flow rule is placed in the flow table to make the dropping of malicious packets. Kumar et al. [17] suggested a defense system named “Safety" that provides mitigation of SYN flooding attack. The proposed system is based on entropy variation for attack detection. The entropy is computed on destination IP and TCP flags. Performance is evaluated using CPU usage of the victim and controller, and attack detection time. One more entropy based defense mechanism was introduced by Kalkan et al. [18]. The proposed mechanism named joint entropy-based security scheme (JESS) detects and mitigates the DDoS attacks. For attack detection, entropy is computed on the destination IP and different attributes of TCP layer. This is the first defense mechanism with the integration of joint entropy metric for SDN. It has the capability to mitigate unknown attacks also. The performance assessment of JESS is measured in terms of accuracy and false positive rate. Results indicate that JESS performs effectively with good performance results. Piedrahita et al. [34] presented a lightweight and fast DDoS detection mechanism named as FlowFence for SDN. The proposed detection approach is based on a congestion condition used to check the bandwidth utilization. The switches in the SDN network monitor the network traffic, and if the congestion happens, the controller is notified about the situation. The controller takes appropriate action against the congestion. The results indicate that FlowFence reduces the congestion in the network but does not prevent the attack completely. In Ref. [37], Buragohain et al. proposed a defense system, namely, FlowTrApp for data centers in SDN. The defense logic works on metrics, i.e., flow rate and flow duration. These metrics decide the sending rate and flow duration used by normal hosts. The defense mechanism decreases the controller's burden. In Ref. [22], an entropy based flooding DDoS detection technique is proposed for SDN. The proposed detection method is implemented on POX controller. The performance of solution is not measured using any standard metric. Also, it does not provide any mitigation scheme to reduce the CPU usage of the controller. Niyaz et al. [19] suggested a defense mechanism based on deep learning against flooding attacks (TCP, UDP, and ICMP) for SDN. The detection module is deployed on the POX controller. The performance assessment is done considering the parameters, i.e., accuracy, precision, recall, and F-measure. Conti et al. [33] presented a detection approach for the detection of DDoS attacks in SDN. An statistical method, i.e., cumulative sum and adaptive threshold is used as a detection method in the presented approach. Performance of the detection method is analyzed using detection time and detection accuracy. However, it does not provide any mitigation for the real-time attack situation. In Ref. [32], a time-series based detection method is studied. Dehkordi et al. [25] presented an entropy and classification based detection mechanism against both the low and higher rate DDoS attacks. Results indicate that the detection mechanism achieves high accuracy but does not provide any mitigation solution, i.e., it does not improve the performance of the controller.

Most of the existing statistical-based DDoS detection techniques utilized the concept of entropy in SDN. In this paper, we use IQR measure as a detection method. To the best of our knowledge, the IQR-based defense solution has been utilized for the first time in the SDN environment. More-over, few defense solutions provide the mitigation of the attacks in SDN. Our proposed defense solution detects the attack as well as reduces the impact of the attack on the controller. Details of some existing DDoS defense solutions are presented in Table 1. It also provides a theoretical comparison of existing defense solutions with the proposed solution. As per the comparison provided in Table 1, the research works [19, 22, 25, 31–33] provide the detection of the DDoS attack. These works do not mitigate the DDoS attack, i.e., do not reduce the impact of attack on the controller. But the proposed defense solution detects the attack as well mitigates the attack on the controller. Therefore, it can be said that the proposed work provides better performance as compared to other research works mentioned in Table 1.

3. Proposed defense solution

This section introduces the proposed defense solution that offers timely detection and mitigation of spoofed SYN flooding attack in SDN networks. Firstly, the base component of the proposed solution, i.e., statistical measure — “Interquartile Range (IQR)" is discussed. Secondly, we present the designed architecture of the proposed solution and different modules in the form of algorithms.

3.1. Interquartile Range (IQR)

IQR is a popular outlier detection method. An outlier rep resents a value, which deviates from the overall pattern of data [38]. Outlier detection has been widely used to detect anomalies. It can be defined in simple words as “something that is significantly different from others". There are various existing methods to identify the outliers, i.e., Z-score, IQR (univariate) and Density-based clustering, distance-based outlier detection, isolation-forest (multi-variate). However, IQR is one of the most commonly used outlier detection methods. IQR can be defined as a measure of statistical dispersion. IQR is a univariate method in statistical modeling, which utilizes a measure “median" to locate the different data points. The use of measures like median, mean, and mode depends on two factors, i.e., type of variability and presence of outliers. In the case of less variability and no outliers, mean and standard deviation are used as statistical measures to classify the data. But if there is a chance of much variability and outliers, median and IQR are preferred to use.

Table 1
Comparison of some existing defense mechanisms against DDoS attacks in SDN with the proposed defense solution.

Authors/Ref.	Defense Mechanism	Mitigation	Performance parameters	Limitations
Kubra et al. [31]	SDNScore statistical threshold based detection method	No	Accuracy, precision, recall, F-measure	Does not reduce the controller's burden
Dehkordi et al. [25]	Statistical entropy based detection	No	Detection accuracy, false positive rate	Does not reduce the controller's burden
Fouladi et al. [32]	Time series based detection	No	Accuracy, false positive rate, F-measure	Does not reduce the controller's burden
Conti el at [33].	Statistical cumulative sum and adaptive threshold based detection	No	Detection time and accuracy	Does not reduce the controller's burden
Niyaz et al. [19]	Deep learning based detection	No	Accuracy, precision, recall, F-measure	Does not show the performance of detection method at real-time generated network traffic and impact of attack on controller
Mousavi et al. [22]	Statistical entropy based detection	No	No standard parameter	Does not reduce the controller's burden
Piedrahita et al. [34]	FlowFence based on a congestion condition	Yes	Bandwidth consumption	Does not prevent the attack completely
Duy et al. [35]	Statistical entropy based defense	Yes	No standard parameter	Does not show the effectiveness of the mitigation method
Kalkan et al. [18]	JESS-joint entropy based detection	Yes	Accuracy, false positive rate	Does not the show the effectiveness of mitigation in terms of CPU utilization of controller
Proposed defense solution	Statistical - IQR based defense	Yes	Attacker detection accuracy, detection time, controller's CPU utilization	Evaluated the performance for a single type of DDoS

IQR is computed by dividing number of data points into different quartiles. A quartile is a type of quantile that exactly divides the number of data points (ordered in ascending manner) into four quarters, which are of approximately equal sizes. The data points are separated using three quartiles indicated by $Q1$, $Q2$, and $Q3$, mathematically. $Q1$, $Q2$, $Q3$ represent first, second, and third quartile, respectively. In simple words, it can be said that the middle 50% of ascendingly ordered data resembles IQR. Mathematically, IQR is defined as the difference between $Q3$ and $Q1$, as shown in Eq. (1).

$$IQR = Q3 - Q1 \tag{1}$$

The following steps are involved in determining the value of IQR.

1. Compute median ($Q2$) of the whole data.
2. Compute first ($Q1$) and third ($Q3$) quartiles by taking median of lower half and upper half of the data.
3. Compute IQR using Eq. (1).

In Ref. [39], Tukey et al. presented $1.5 \times IQR -$ fences to be used as IQR-rule to mark the possible outliers. According to the proposed IQR-rule ($1.5 \times$ IQR), two Eq. (2) and Eq. (3) are utilized as lower bound and upper bound, respectively. The data values that are below lower bound and that are above upper bound are flagged as outliers.

$$lower\ bound = Q1 - 1.5 \times IQR \tag{2}$$

$$upper\ bound = Q3 + 1.5 \times IQR \tag{3}$$

3.2. Architecture and functionalities

In this section, the working of the proposed defense solution is discussed. Fig. 2 presents a flowchart of the overall architecture of the defense solution. The overall architecture is based on two working modules: detection module and mitigation module. The detection module is responsible for the detection of malicious traffic. The proposed solution utilizes the concept of outlier detection to identify malicious traffic. Detection method is based on IQR statistical measure acting as a classifier to classify the traffic into attack and normal classes. The key objective of the detection module is to identify the hosts that present significantly different behaviour. The host that shows anomalous behaviour is called an outlier host. As per the proposed detection method, the host from where a large amount of malicious traffic received is marked as an outlier host. There are two equations (Eq. (2) and Eq. (3)) for calculating IQR. To detect the spoofed SYN flooding attack, Eq. (3) is used in the proposed defense solution. The existing Eq. (3) adds some delay in the attack detection. Therefore, the equation is modified as per the requirement of the detection method. For the required modification, many experiments are conducted to determine an appropriate value of *upper bound*. The selection of *upper bound* depends on the improvement in the attack detection time. The modified Eq. (3) is shown in Eq. (4), where y is a tuning parameter. In the proposed solution, tuning parameter is used to enhance the performance of the detection method in terms of responsiveness (attack detection time).

$$upper\ bound = Q3 + y \times IQR \tag{4}$$

The working logic of the detection method can be understood in the context of our work as follows. When network traffic is monitored in the case of spoofed SYN flooding DDoS attack, it can be observed that the victim host receives a large number of SYN requests from attacker hosts as compared to other normal hosts. These spoofed SYN requests are sent to controller as *packet_in*. On the receiving a number of *packet_in* messages, some important measures like *Q1*, *Q2*, *Q3*, *IQR*, and *upper bound* are computed at a time interval. The *upper bound* is utilized as a threshold to check the number of *packet_in* messages received at the controller. It acts as an essential condition to decide if a host is suspicious or normal. In the normal (non-attack) scenario, the *packet_in* received at the controller by switch are below the *upper bound*. Thus, there is no outlier (malicious behaviour) detected. However, in an attack scenario, the count of *packet_in* received is greater than the *upper bound* limit. This kind of behaviour will be flagged as an attack scenario. The victim host and source of the attack are the output provided by the detection module. The source of the malicious

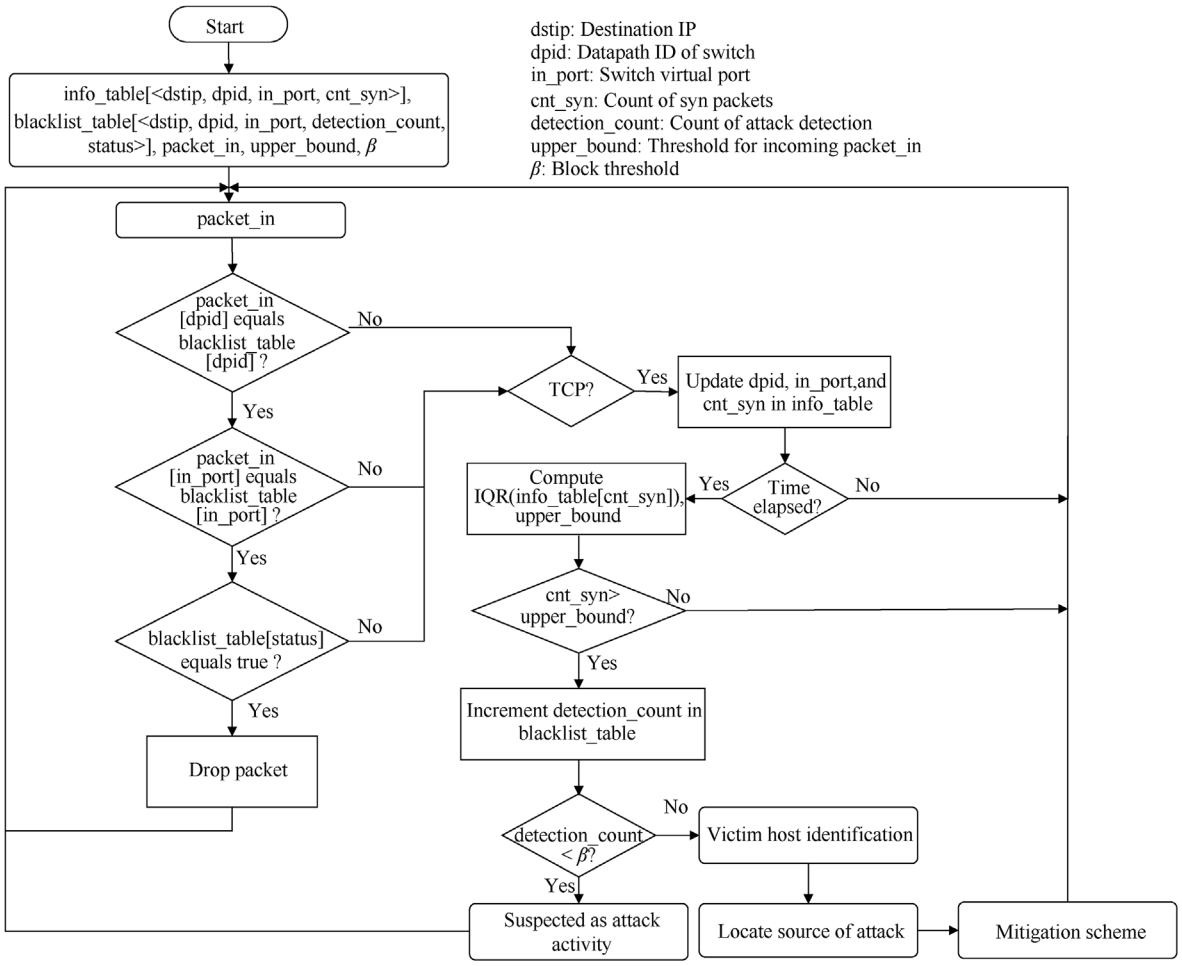

Fig. 2. Flowchart for proposed defense solution.

traffic can be identified as an outlier. This logic is deployed as an attack detection method in the proposed solution. In the proposed work, source of the attack is located with *dpid* (datapath identifier) of the respective switch and its *in_port*. When the attachment point of the attacker host (*dpid*, *in_port*) is identified, working of the mitigation module is started.

The proposed defense solution consists of three procedures:

PAcket_IN_HANDLER, *DETECTION_MODULE*, and *MITIGATION*. Pseudo-codes of these procedures are presented in Algorithms 1, 2, and 3. Table 2 lists out different symbols and their respective explanations used in the proposed algorithms.

3.3. Description of PAcket_IN_HANDLER procedure

Pseudo-code of *PAcket_IN_HANDLER* procedure is presented in Algorithm1. The *PAcket_IN_HANDLER* procedure is implemented as a modified version of *packet_in_handler*, i.e., standard method of the SDN controller (Ryu). *packet_in_handler* is executed after the reception of the *packet_in* message from the SDN switch. This method is responsible for the handling of *packet_in* messages and installing the new flow rules in the switch to process the further incoming packets. Similarly, *PAcket_IN_HANDLER* procedure is executed whenever a *packet_in* message is received at the controller. According to the proposed solution, rules are set in the switches in such a way that spoofed SYN packets are sent to the controller by default. In the proposed solution, two threshold values are used to check on the suspicious traffic. Firstly, *upper_bound* acts as a threshold for identifying if the respective host is normal or suspicious. After that, *dcount* (p) is used as a block threshold. The block threshold reduces the chance of permanent blocking of falsely detected hosts as suspicious. This is why, when an attack scenario is identified, corresponding dpid, inport (attachment point of the suspicious host) as the source of malicious traffic are stored in a suspicious state. The network traffic from such a suspicious host is allowed to the victim host until the block threshold meets its limit condition. The source of the victim host is permanently blocked when the block threshold reaches its threshold and marked as an attacker. We have considered the value of p as 5. In the *upper_bound* condition, a tuning parameter (y) is also used to quickly detect the attack situation.

This procedure performs some major functions:

- Early detection of the blacklisted host and dropping the malicious packet without going through the whole detection process
- Maintains an info table (*info*) to store the information related to the incoming packet_in messages
- Executes *DETECTION_MODULE* procedure after fixed time interval to check on the suspicious hosts.

3.4. Description of DETECTION_MODULE procedure

The *DETECTION_MODULE* procedure is an important part of the proposed defense solution. This procedure is responsible for two

Table 2
Symbols and definitions.

Symbol	Definition
pkt	Incoming *packet_in* message
info[< dstip, dpid, in_{port}, c_{syn} >]	Database to store information related to *packet_in* message. Where, *dstip* represents destination IP address of the received *packet_in*, *dpid* represents datapath id of switch from where *packet_in* message comes, *inport* represents in_port of the received *packet_in*, and *csyn* represents total number of received *packet_in*.
blacklist [< dstip, dpid, inport, dcount, status >]	Database to store information of blacklisted and suspected hosts. Where, dst_{ip} represents the destination IP of the blacklisted host, *dpid* represents the corresponding datapath id of blacklisted host, in_{port} represents in_port of the corresponding *dpid* of the blacklisted host, d_{count} is number of times when the host has been identified as suspicious, and *status* indicates the status of the host present in blacklist table (i.e., *False* represents the suspected host and *True* represents the permanently blocked host)
flags.tcp	Flag to check if the SYN flag in tcp protocol is set or not.
InInfo	Flag to check if the host is present in info table or not
InBlacklist	Flag to check if the host is present in info table or not.
active	It indicates that *DETECTION_MODULE* procedure is executed to check for the attackers present in info table after a specified time interval every time.
Q1	First quartile
Q3	First quartile
upper_bound	It indicates the threshold for the received *packet_in* messages from an *in_port* of a datapath.
α	Tuning parameter
β	Threshold for blocking malicious host permanently
flag	Flag to check if a host satisfies the upper bound condition for once or not, i.e., set *True* for suspected host and set *False* for no attacker detected.
a_{dpid}	*dpid* of the permanently blocked host (i.e., detected as attacker)
a_{inport}	Corresponding *in_port* of *dpid* of the permanently blocked host

major functions: Checks and identifies the malicious hosts as outliers present in the info table (*info*); Maintains a blacklist table (*blacklist*) to store the information related to suspected and permanently blocked hosts. Pseudo code of *DETECTION_MODULE* is presented in Algorithm 2. The necessary computations (i.e., *Q1* and *Q2*) are made on the count of *packet_in* messages received from different hosts to detect the malicious hosts. In *DETECTION_MODULE* procedure, the IQR method is deployed to compute the thresh old (i.e., *upper_bound*). All the hosts in *info* are checked against *upper_bound* and if any of the host does not satisfy the threshold condition, a “suspicious scenario situation" is labeled, and then the victim host (*dstip*) with their respective *dpid, inport* is inserted into the blacklist table (*black list*). After the victim host is detected, its *dpid* of the corresponding switch and *inport* are identified as the source of the victim host. The source host is marked as an attacker and blocked permanently if it is under suspicion for p times. Then the attachment point (*dpid*, *inport*) from where attack traffic comes, is passed to the *MITIGATION* procedure. *MITIGATION* procedure is executed every time upon detection of the attacker host. Also, the entry of the located attacker host is removed from *info*.

Algorithm 1. Psuedo-code of PAcket_IN_HANDLER procedure

1: info[<dst_{ip}, dpid, in_{port}, c_{syn}>], blacklist[<dst_{ip}, dpid, in_{port}, d_{count}, status>]
2: pkt_i. ▷Global Variables
3: **procedure** PAcket_IN_HANDLER(pkt_i)
4: **for** i ← 1, len(blacklist) **do**
5: **if** blacklist[dst_{ip}.dpid] = pkt_i.dpid AND blacklist[dst_{ip}.in_{port}] = pkt_i.in_{port} AND blacklist[dst_{ip}.status] = True **then**
6: Drop the packet
7: **return**
8: end if
9: end for
10: **if** pkt_i.flags.tcp = S **then**
11: InInfo ← False
12: **for** i ← 1, len(info) **do**
13: **if** info[dst_{ip}] = pkt_i.dst_{ip} AND info[dst_{ip}.dpid] = pkt_i.dpid AND info[dst_{ip}.in_{port}] = pkt_i.in_{port} **then**
14: InInfo ← True
15: info[dst_{ip}.c_{syn}] info[dst_{ip}.c_{syn}] + 1
16: **end if**
17: **end for**
18: **if** InInfo = False **then**
19: info[dst_{ip}] ← pkt_i.dst_{ip}
20: info[dst_{ip}.dpid] ← pkt_i.dpid
21: info[dst_{ip}.in_{port}] ← pkt_i.in_{port}
22: info[dst_{ip}.c_{syn}] ← 1
23: **end if**
24: **if** (active = True) then
25: **if** len(info) > 0 **then**
26: call DETECTION_MODULE (info) procedure
27: active ← False
28: **end if**
29: **end if**
30: **end if**
31: **end procedure**

3.5. Description of MITIGATION procedure

After the processing of *DETECTION_MODULE* is over, mitigation of the attack is performed. The pseudo-code of the *MITIGATION* procedure is presented in Algorithm 3. The primary function of the mitigation module is to prohibit malicious packets. This is achieved by SDN-in-built functionalities. As per these functionalities, a flow modification rule is installed in the flow table of switch, enabling the dropping of packets from the detected attachment point (*dpid*, *inport*) of the attacker host. The working flow of mitigation strategy is described in Fig. 3.

Algorithm 2. Psuedo-code of DETECTION_MODULE procedure

1: **procedure** DETECTION_MODULE(info) B Detects the attackers present in info table
2: sort info on cs_{yn} column
3: compute Q1 and Q3 on c_{syn} column
4: IQR ← *Q3-Q1*

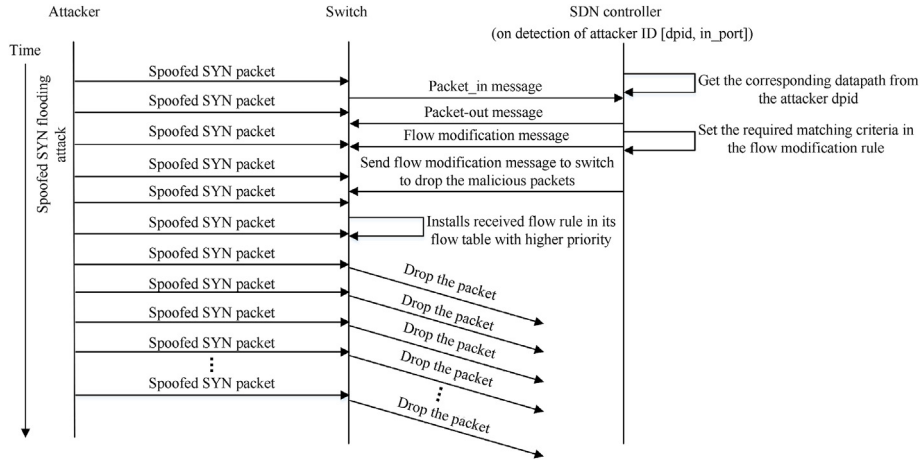

Fig. 3. Mitigation procedure after attacker's detection in the proposed defense solution.

5: upper_bound ← Q3 + (a × IQR)
6: flag ← False
7: **for** i ← 1, len(info) **do**
8: **if** *info*[dst_{ip} .c_{syn}] > upper_bound **then**
9: Suspicious traffic is detected
10: flag ← True
11: InBlacklist False
12: **for** j 1, len(blacklist) **do**
13: **if** blacklist[dst_{ip}] = info[dst_{ip}] AND blacklist[dpid] = info [dpid] AND blacklist[inport] = info[inport] **then**
14: InBlacklist True
15: **if** blacklist[dcount] < β **then**
16: blacklist[dcount] ← blacklist[dcount] + 1
17: **if** blacklist[dcount] = β **then**
18: blacklist[status] True
19: blacklist[dst_{ip}] is identified as victim
20: blacklist[dpid, in_{port}] is identified as source of attack
21: respective in_{port} of *dpid* is permanently blocked
22: a_{dpid} ← blacklist[dpid]
23: a_{in}port blacklist[in_{port}]
24: *call* MITIGATION (a_{dpid},a_{inport})
25: remove info entry(i)
26: **end if**
27: **end if**
28: **end if**
29: **end for**
30: **if** InBlacklist = False **then**
31: blacklist[dst_{ip}] ← info[dst_{ip}]
32: blacklist[dst_{ip} .dpid] ← info[dpid]
33: blacklist[dst_{ip} .in_{port}] ← info[in_{port}]
34: blacklist[dst_{ip} .d_{count}] ← 1
35: Host is suspected to be an attacker
36: **end if**
37: **end if**
38: **end for**
39: **if** flag = False **then**
40: No attack is detected
41: **end if**
42: **end procedure**

Algorithm 3. Psuedo-code of Mitigation procedure

1: **procedure** MITIGATION(a_{dpid}, a_{inport})
2: get corresponding datapath
3: set matching criteria with respect to a_{inport}
4: send a flow modification message to switch
5: drop packets corresponding to a_{inport} of a_{dpid}
6: **end procedure**

4. Simulation setup and performance analysis

In this section, the required network setup to carry out the experiments is discussed. The experimental topology is presented in Section 4.1. Section 4.2 provides a discussion of the considered parameters to evaluate the performance of the proposed solution and analyzes the performance results. All the experiments are performed on an SDN emulator – Mininet *v*2.3.0d6 [40]. Mininet creates a virtualized network scenario consisting of hosts, switches, and links on a single Linux kernel and provides the flexibility of custom topologies. The OpenFlow supported Open vSwitch (OVS) *v*2.9.5 [41] is used, which supports the virtual ports and routing of network traffic. In SDN network, a 64-bit identifier called *dpid* (Datapath Identifier) is associated with each virtual switch (OpenFlow-based). Each *dpid* contains some virtual ports identified by *in_port*. Ryu *v*4.32 [42] is utilized as SDN controller, on which the defense solution is implemented. A standard protocol "OpenFlow" *v* 1.3 is used to make the communication between switches and controller possible. For generating TCP traffic, a Python based packet generation tool named Scapy [43] is used. For conducting experiments, attacker host is allowed to send attack traffic for 600 s. The detection module is executed repeatedly after every 25 s to check for suspicious hosts. Table 3 presents a brief overview of the setup details.

Table 3
Experimental setup.

Resource	Configuration
Machine characteristics	*Intel® Core™* i7-7700 CPU, 1 TB HDD, 8 GB RAM
Controller	Ryu *v*4.32
Network emulator	Mininet *v*2.3.0d6
OpenFlow version	1.3
Attack generation tool	Scapy
Software switch	Open vSwitch
Victim OS	Ubuntu 18.04 LTS (64-bit)
Attacker OS	Ubuntu 18.04 LTS (64-bit)
Type of network traffic	Spoofed SYN packets

4.1. Experimental topology

The experimental topology consists of 1 OVS and 21 hosts. All the hosts use a bandwidth of 100 Mbps. Open vSwitch is connected with Ryu controller. The simulations are con-ducted on different attack scenarios (as presented in Fig. 4). Each attack scenario consists of number of attacker and normal hosts (non-attacker). In all the scenarios, every attacker sends the spoofed SYN packets at rates varying from 50 to 300 (50, 100, 150, 200, 250, 300) packets/second. These scenarios are discussed as below:

- *Scenario-A1*: In this scenario, the number of attacker and normal hosts are 1 and 4, respectively. The attacker sends spoofed SYN packets at different rates. One host is considered as a victim.
- *Scenario-A2*: The number of attackers is 2 and 8 hosts act as normal hosts in *Scenario-A2*. Each of the attackers sends the spoofed SYN packets at different rates as similar to *Scenario-A1*. One host acts as a victim.
- *Scenario-A3*: In this scenario, 3 hosts are used as attackers and 12 hosts are used as normal hosts. All the attackers send the spoofed SYN packets at different packet rates.
- *Scenario-A4*: In this scenario, the number of attacker and normal hosts is 4 and 16, respectively. The attack is simulated at different packet rates from all the attacker hosts.

4.2. Performance evaluation

This section discusses an analysis of the results of the proposed defense solution in respect of different performance parameters.

4.2.1. Evaluation parameters

The effectiveness of the proposed defense solution is measured using different evaluation parameters. The proposed solution is based on two working modules, i.e., detection and mitigation. The detection module is evaluated in terms of detection time and detection accuracy. The impact of spoofed SYN flooding attack on the controller is analyzed in terms of *packet_in* messages received and CPU usage of the controller under attack with mitigation strategy. The definitions of the evaluation parameters are as follows:

- *Average attacker's detection time (AADT)*: It is defined as the time interval between the first SYN packet sent by the attacker and attack detection by the SDN controller. *AADT* is presented in Eq. (5).

$$AADT = \frac{\sum_{i=1}^{n} t_{\text{detection}_i} - t_{\text{launch}_i}}{n_{\text{attacters}}} \tag{5}$$

where $t_{\text{detection}_i}$ and t_{launch_i} are the time of attacker detection and initiation of the first attack packet from the ith attacker. $n_{\text{attacters}}$ is the number of attackers present in the network.

$$ADA = \frac{a_{\text{ture}} + n_{\text{ture}}}{a_{\text{ture}} + a_{false} + n_{\text{ture}} + n_{\text{false}}} \tag{6}$$

where a_{true}, n_{true} represent the number of correctly detected attackers and normal hosts (non-attackers), respectively. a_{false}, n_{false} are wrongly detected attackers and normal hosts, respectively.

- *Attacker's detection accuracy (ADA)*: It is defined as the ratio of total number of correctly detected observations (attackers and normal hosts) over all the observations made by the detection method. ADA is shown in Eq. (6).
- *Number of packet_in messages*: It represents the total number of *packet_in* messages received at the controller. These are sent by the switch as they are unable to handle the spoofed packets. *Packet_in* is an OpenFlow message that can be captured on an OpenFlow port by monitoring the respective interface.
- *Average CPU usage of the controller*: It is defined as the CPU consumption of the running SDN controller. The CPU usage is calculated by monitoring the respective process id (PID) of the controller for a duration. In this work, CPU usage is monitored for the Ryu controller.

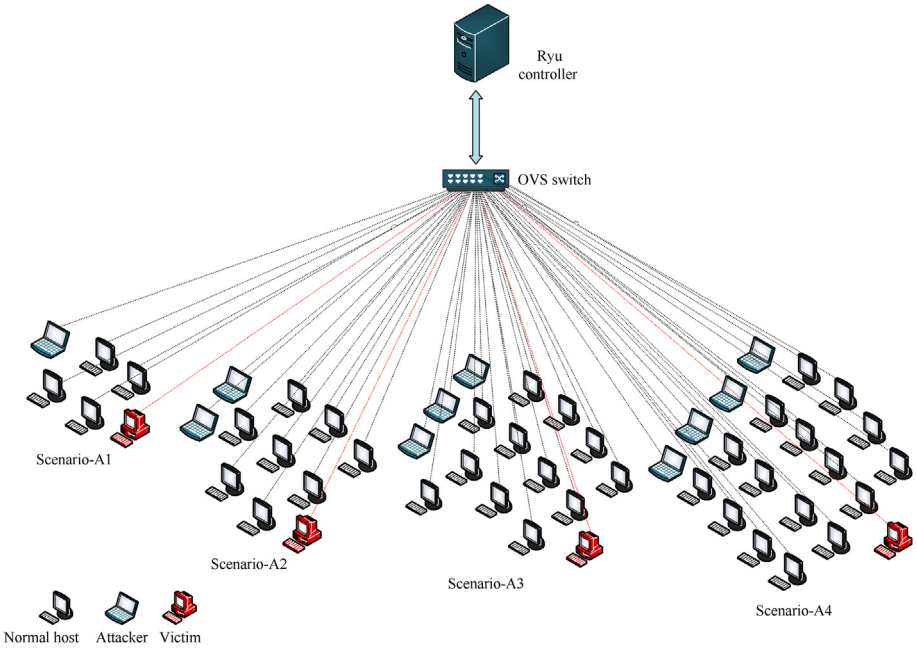

Fig. 4. SDN topology used in the experiments.

4.3. Simulation results

For performance evaluation of the proposed defense solution, many experiments are conducted on all the scenarios as discussed in Section 4.1. In this section, the performance of the proposed detection method is analyzed in terms of attacker's detection time and detection accuracy. We also analyze the impact of spoofed SYN flooding attack on the controller in terms of CPU usage and received *packet_in* messages for both the cases, i.e., under attack with and without defense.

4.3.1. Performance of detection method in terms of attacker's detection time

The timely detection of any malicious behaviour is very important for the applications of cyber security. Considering this major aspect, we have evaluated the performance of the proposed detection method in terms of detection time. Fig. 5 presents average attacker detection time to identify the present attackers in the SDN network with varying packet rates for all the scenarios. The detection time is computed by calculating an average of the time taken to detect each attacker in a scenario. These scenarios are indicated by *ScenarioA1*, *Scenario-A2*, *Scenario-A3*, and *Scenario-A4*. It can be seen that attacker's detection time does not increase over 157 s for all the scenarios. Scenarios-A1 outperforms all other attack scenarios (A2, A3, and A4) over all the other packet rates in terms of attacker detection time. The detection time increases when the number of attackers is increased. The proposed mechanism achieves maximum detection time to detect the most aggressive attackers in the case of Scenario A4, i.e., attack at the packet rate of 300 packets/second. It can be concluded from the achieved results that an increase in number of attackers and attack packet rate does not affect the detection time as much. Although, it takes less time to detect only one attacker at less packet rate than multiple attackers at a higher rate as shown in Fig. 5.

4.3.2. Performance of detection method in terms of attacker's detection accuracy

Accuracy is another important parameter to evaluate the performance of a designed scheme. We assessed the detection accuracy of the proposed defense solution in different attack scenarios, as shown in Fig. 6. It shows the performance of the defense solution in terms of accuracy, considering the presence of both the attackers and normal hosts. Fig. 6 shows the achieved accuracy concerning elapsed time and different packet rates. Fig. 6(a) presents the achieved accuracy by the proposed solution in case of attack *ScenarioA1*. It can be seen from the figure that the accuracy of the defense solution is perfect as 100% in *Scenario-A1* at all the packet rates. Also, accuracy does not get affected by the elapsed time with the presence of normal hosts. Accuracy achieved by defense solution in case of attack *Scenario-A2* is presented in Fig. 6(b). It can be observed that in the case of *Scenario-A2*, the defense solution performs best by achieving 100% detection accuracy at the packet rates of 100, 150, 200, 250, and 300 packets/second during the elapsed time. With the packet rate of 50 also, it achieves 100% detection accuracy initially. Later, accuracy decreases with time after some time has elapsed, and it reaches 90%. Similarly, Fig. 6(c) and (d) present the accuracy variation as per the time and different packet rates in the case of *Scenario-A3* and *Scenario-A4*, respectively. It can be observed from the results that in the *Scenario-A3* also, accuracy remains 100% initially with all the packet rates. With the time elapsed, accuracy decreases to 93% at the packet rate of 50 packets/second. In the case of *Scenario-A4*, the defense solution performs with accurate detection of attackers and normal hosts at the packet rates of 50, 100, and 150 during the elapsed time. Whereas the detection accuracy at the packet rates of 200, 250, and 300 varies as per the elapsed time. Initially, the proposed solution achieves 50% accuracy at the rate of 300. After that, accuracy increases to its maximum of 100% with the detection of all the attackers correctly. Finally, it decreases to the maximum of 95% as per the time. With the packet rates of 200 and 250 also, accuracy decreases with the time duration and the proposed defense solution achieves the overall accuracy of 95%. The detection accuracy decreases after some time because after detection of all the attackers correctly, some normal hosts can be identified as attackers mistakenly. However, by considering all the attack scenarios with all the used packet rates, it can be concluded that the proposed defense solution achieves a maximum of 100% accuracy in case of (*Scenario-A1*). In the DDoS attack scenarios (*Scenario-A2, A3, A4*), the defense scheme performs better by achieving a maximum of 95% and a minimum of 93% accuracy concerning time and all the packet rates. The number of attackers and normal hosts are added incrementally with respect to the network topologies used in the different attack *Scenario-A2*, *A3*, *A3*. This creates an increase in the transmission of *packet_in* messages even from the normal hosts, making the normal hosts suspected as an attacker due to being detected as an outlier. However, the block threshold (β) avoids the permanent blocking of normal hosts.

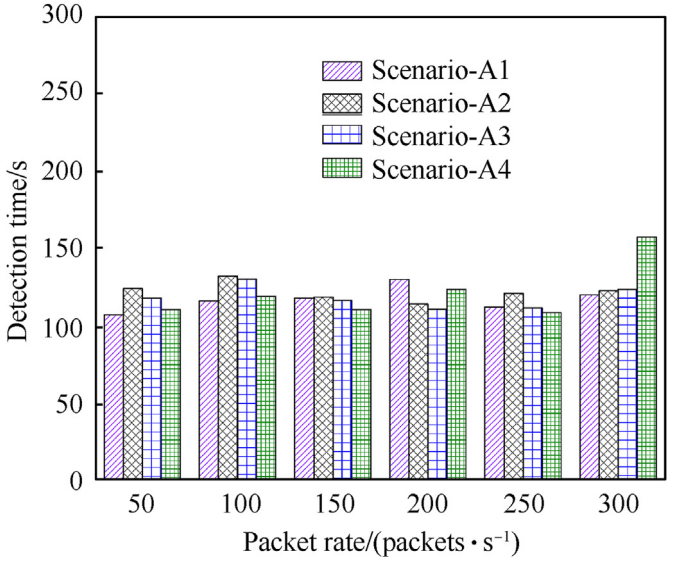

Fig. 5. Average detection time to detect attackers with varying packet rates for different scenarios.

4.3.3. Impact on controller in terms of packet_in messages

In the case of spoofed flooding attack, a large number of *packet_in* messages is forwarded to the controller by the SDN switch. This flooding of *packet_in* messages creates a huge CPU overhead on the controller to be handled. The *packet_in* messages are important to analyze the impact of the attack on the controller. Reducing the controller's burden is one of the prime requirements of the SDN network in case of flooding DDoS. Therefore, in our work, we analyze the performance of the proposed solution in terms of number of packet_in messages received at the controller. Fig. 7 shows a comparison of packet_in messages received in the case of under attack without defense and with defense for all the different attack scenarios (Scenario-A1, A2, A3, A4) with varying packet rates (50, 100, 150, 200, 250, 300 packets/second). Increasing the packet rate and the number of attackers intends to the increased number of packet_in messages on the controller. It can be observed from the results that total number of packet_in

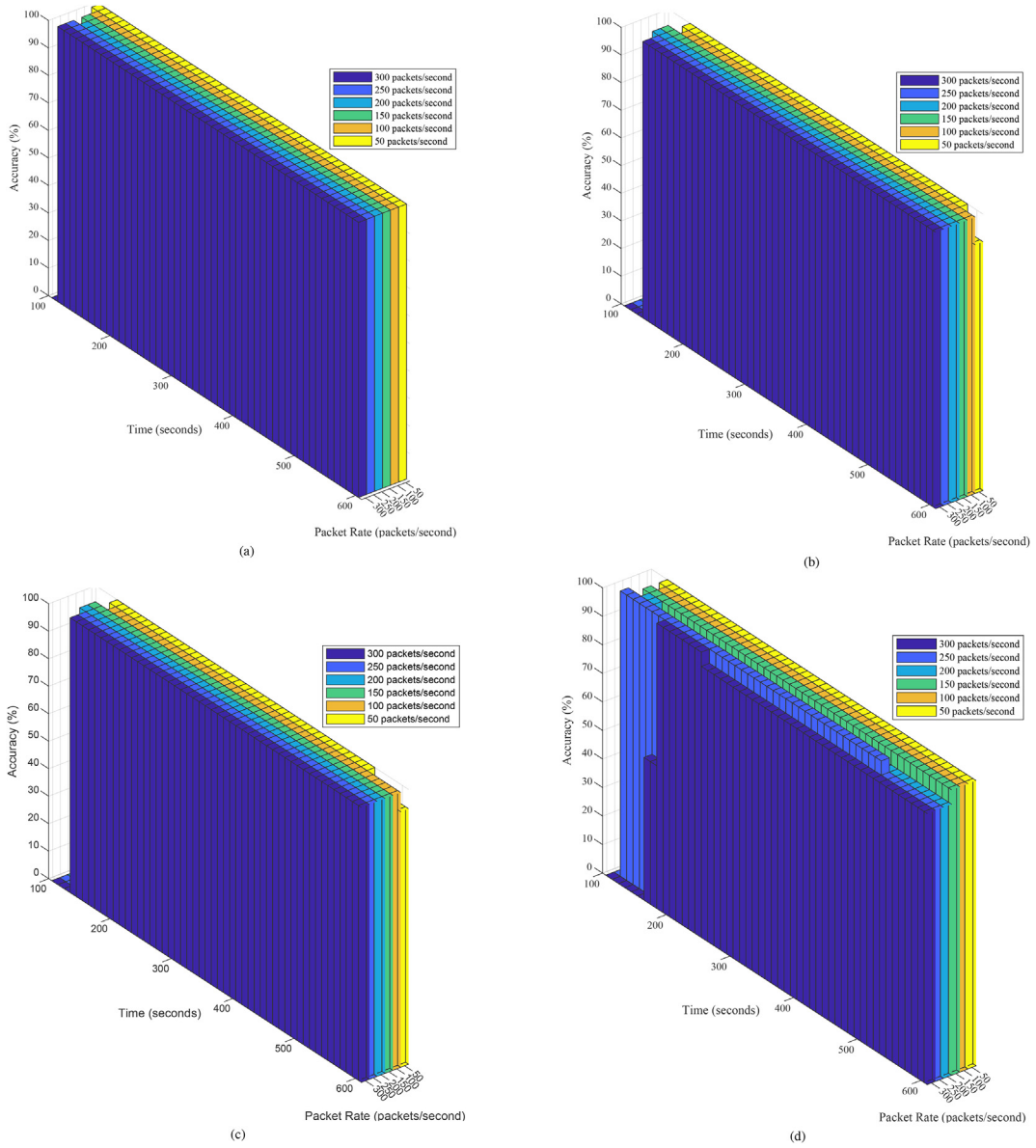

Fig. 6. Detection accuracy of the proposed defense scheme.

messages increases with more intensity with the increase of number of attackers as compared to increase in the packet rate, which overburdens the controller. This is ensured from the comparison of packet_in messages values of Scenario-A1$_{\text{Under Attack}}$, A2$_{\text{Under Attack}}$, Scenario-A3$_{\text{Under Attack}}$, and Scenario-A4$_{\text{Under Attack}}$ at different packet rates. Average value of packet_in messages received in case of Scenario-A1$_{\text{Under Attack}}$ and Scenario-A1$_{\text{Under Attack With Defense}}$ at different packet rates are 39080 and 15171, respectively. The average value of packet_in messages received in case of Scenario-A2$_{\text{Under Attack}}$ and Scenario-A2$_{\text{Under Attack With Defense}}$ at different packet rates are 82766 and 33365, respectively. Similarly, the average value of the packet_in messages received in the case of Scenario-A3$_{\text{Under Attack}}$, Scenario-A3$_{\text{Under Attack With Defense}}$ and Scenario-A4$_{\text{Under Attack}}$, Scenario-A4$_{\text{Under Attack With Defense}}$ at different packet rates are 129964, 51038, and 176486, 62378, respectively. It can be seen that the proposed solution reduces the number of incoming packet_in messages at the controller after identification of the potential attackers. This happens because the proposed solution firstly detects the existing attackers and blocks the respective dpid and in_port of the attacker to drop the attack packets. This blocking mitigates the impact of flooding attack on the controller effectively.

4.3.4. Impact on controller in terms of CPU usage

The controller manages all the crucial functionalities for the entire SDN network making it the most attractive target point of the attackers. Spoofed flooding attacks exhaust the CPU on the controller by forwarding a large number of *packet_in* messages as discussed in Ref. [30]. This exhaustion of the controller degrades the performance of the network. Thus, the controller's CPU usage is one of the essential factors to be per second for *Scenario-A1*, *Scenario-A2*, *Scenario-A3*, and *Scenario-A4*, respectively. The monitored CPU utilization for the case of without attack remains consistent at a very less amount for all the scenarios. It can be seen in Fig. 8(a) that the CPU usage lies between 16.43% and 30. 1% at an increasing packet rate. The proposed solution reduces the CPU usage up to a maximum of 62.35% and a minimum of 37.07% in case of Scenario-A1$_{\text{Under Attack With Defense}}$. The attack Scenario-A2$_{\text{Under Attack Without}}$

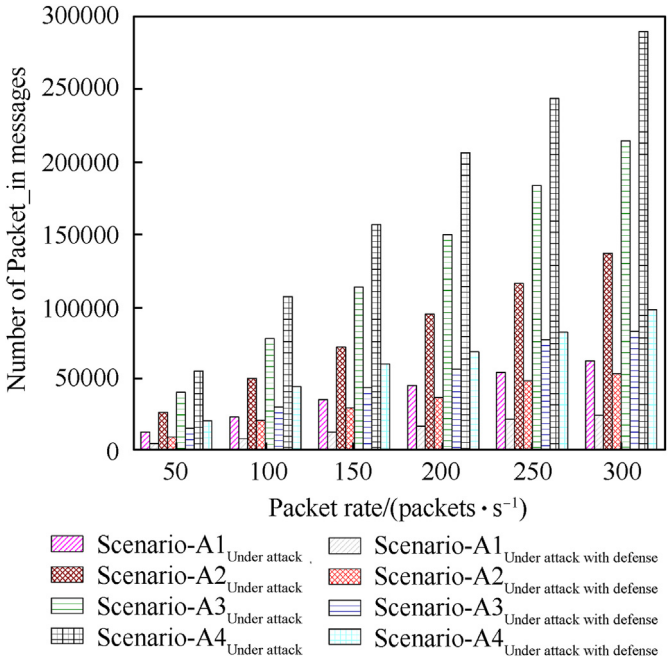

Fig. 7. Number of *packet_in* messages with varying packet rates for different scenarios.

Defense consumes the controller's CPU between the range of 26.45% and 40. 15%. The solution decreases the CPU consumption up to a maximum of 57. 11% and a minimum of 51.68% as shown in Fig. 8(b). The CPU utilization for Scenario-A3Under Attack Without Defense lies between 31.06% and 51.53%. Fig. 8(c) shows the performance of the mitigation solution under the running attack traffic (Scenario-A3Under Attack With Defense), which reduces the CPU usage up to a maximum of 56.91% and a minimum of 50.99%. The average CPU usage of the controller in case of Scenario-A4Under Attack Without Defense varies between 36. 10% and 66.34% as shown in Fig. 8(d). The improvement in the CPU consumption observed in the case of Scenario-A4Under Attack With Defense varies between 42.43% and 57.58%. The proposed solution reduces the CPU consumption in the case of attack. The CPU usage initially goes at a higher value, and then it decreases suddenly. It happens because the defense solution detects the attackers present in the network and then instructs the switch to drop the attack packets by blocking the corresponding *dpid*, *in_port* of the attacker. It can be concluded that the proposed defense solution is capable of mitigating the DDoS attack traffic coming from attackers and reducing the impact of attack on the controller's CPU.

5. Conclusions and future work

The SDN controller is the most attractive target point of the DDoS attackers due to its centralized nature. Currently, securing the controller from such attacks is the biggest challenging task in SDN. In this paper, we presented an statistical measure Interquartile Range (IQR) based technique to secure SDN networks from spoofed SYN flooding attack. The detection logic is applied to the *packet_in* messages received at the controller in every fixed time interval. After detection, mitigation is performed by blocking the identified attackers. The performance of the proposed defense solution is evaluated in terms of detection time, detection accuracy, *packet_in* messages, and CPU utilization of the controller. The proposed defense solution is capable of timely detection as well as mitigation of the attack at the switch itself. The proposed solution has been designed to detect a single type of DDoS attack. Further, the defense solution is not evaluated on a large scale of normal hosts (non-attacker). In future, we aim to evaluate the proposed solution for various DDoS attacks on a larger scale in SDN.

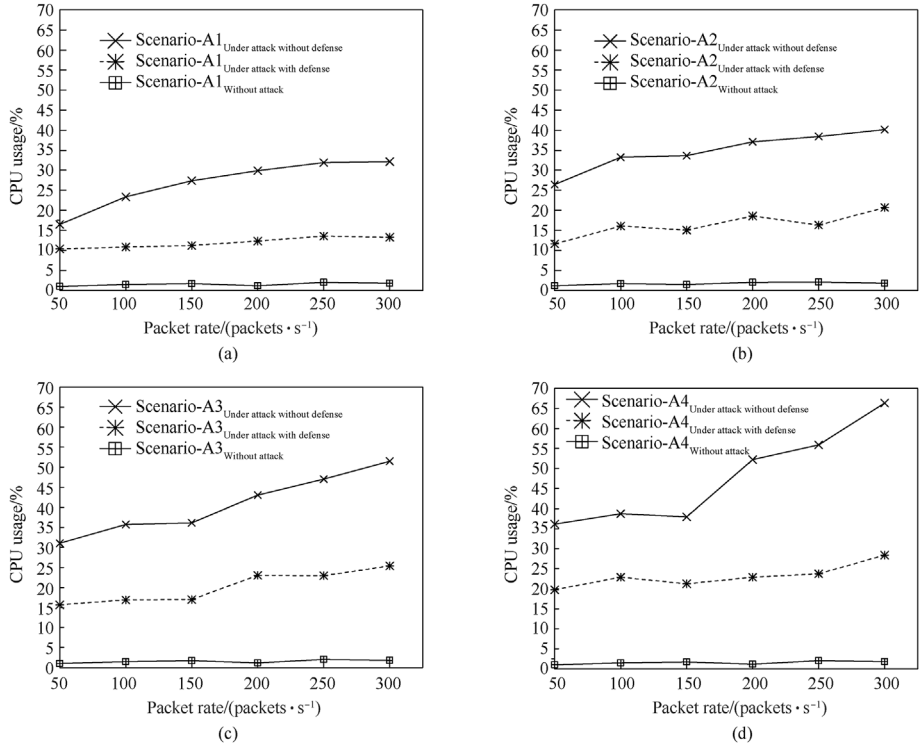

Fig. 8. Average CPU usage with varying packet rates for different attack scenarios.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Hakiri Akram, Gokhale Aniruddha, Pascal Berthou, Schmidt Douglas C, Gayraud Thierry. Software defined networking: challenges and research opportunities for future internet. Comput Network 2014;75:453–71.
[2] Kreutz Diego, Ramos Fernando MV, Verissimo Paulo, Rothenberg Christian Esteve, Azodolmolky Siamak, Steve Uhlig. Software-defined networking: a comprehensive survey. Proc IEEE 2015;103:14–76.
[3] Kirkpatrick Keith. Software-defined networking. Commun ACM 2013;56:16–9. September.
[4] McKeown Nick, Anderson Tom, Balakrishnan Hari, Guru Parulkar, Peterson Larry, Rexford Jennifer, Scott Shenker, Turner Jonathan. Openflow: enabling innovation in campus networks. Comput Commun Rev 2008;38(2):69–74.
[5] Tourrilhes J, Sharma P, Banerjee S, Pettit J. SDN and OpenFlow evolution: a standards perspective. Computer Nov 2014;47(11):22–9.
[6] Paul Goransson, Black Chuck, Culver Timothy. Software defined networks: a comprehensive approach. Morgan Kaufmann; 2016.
[7] Kim Hyojoon, Feamster Nick. Improving network management with software defined networking. IEEE Commun Mag 2013;51(2):114–9.
[8] Swami Rochak, Dave Mayank, Ranga Virender. Software-defined networking-based DDoS defense mechanisms. ACM Comput Surv 2019;52(2):28.
[9] Swami Rochak, Dave Mayank, Ranga Virender. DDoS attacks and defense mechanisms using machine learning techniques for SDN. In: Security and privacy issues in sensor networks and IoT. IGI Global; 2020. p. 193. 214.
[10] Douligeris Christos, Mitrokotsa Aikaterini. DDoS attacks and defense mechanisms: classification and state-of-the-art. Comput Network 2004;44:643–66.
[11] Specht Stephen M, Lee Ruby B. Distributed denial of service: taxonomies of attacks, tools and countermeasures, princeton architecture laboratory for multimedia and security. 2003. Technical report, technical report.
[12] Ramachandran Shyamala, Shanmugam Valli. Impact of DoS attack in software defined network for virtual network. Wireless Pers Commun 2017;94(4):2189–202.
[13] Elejla Omar E, Anbar Mohammed, Belaton Bahari, Hamouda Shady. Labeled flow-based dataset of icmpv6-based ddos attacks. Neural Comput Appl 2019;31(8):3629–46.
[14] Gupta BB, Badve Omkar P. Taxonomy of dos and ddos attacks and desirable defense mechanism in a cloud computing environment. Neural Comput Appl 2017;28(12):3655–82.
[15] Neelam Dayal, Maity Prasenjit, Srivastava Shashank, Khondoker Rahamatullah. Research trends in security and DDoS in SDN. Secur Commun Network 2016;9(18):6386–411.
[16] Qiao Yan, Yu F Richard, Gong Qingxiang, Li Jianqiang. 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 2015;18(1):602–22.
[17] Kumar Prashant, Tripathi Meenakshi, Nehra Ajay, Conti Mauro, Lal Chhagan. SAFETY: early detection and mitigation of TCP SYN flood utilizing entropy in SDN. IEEE Transactions on Network and Service Management 2018;15(4):1545–59.
[18] Kalkan K, Altay L, Gäijr G, Alagöz F, Jess. Joint entropy-based DDoS defense scheme in SDN. IEEE J Sel Area Commun Oct 2018;36(10):2358–72.
[19] Niyaz Quamar, Sun Weiqing, Ahmad Y. Javaid. A deep learning based DDoS detection system in software-defined networking (SDN). 2016, 07400. CoRR, abs/1611.
[20] Chen Zhuo, Jiang Fu, Cheng Yijun, Gu Xin, Liu Weirong, Peng Jun. XGBoost classifier for DDoS attack detection and analysis in SDN-based cloud. IEEE Int Conf Big Data and Smart Comput (BigComp) 2018:251–6. IEEE, 2018.
[21] Verma Abhishek, Ranga Virender. CoSec-RPL: detection of copycat attacks in RPL based 6LoWPANs using outlier analysis. Telecommun Syst: Moelling, Anal Des Manag 2020:1–19.
[22] Seyed Mohammad Mousavi and Marc St-Hilaire. Early detection of DDoS attacks against SDN controllers. In: International conference on computing, networking and communications (ICNC). IEEE; 2015. p. 77–81. 2015.
[23] Moustafa Nour, Jill Slay. The evaluation of Network Anomaly Detection Systems: statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. Inf Secur J A Glob Perspect 2016;25(1–3):18–31.
[24] Raja Majid Ali Ujjan, Zeeshan Pervez, Dahal Keshav, Bashir Ali Kashif, Rao Mumtaz, González J. Towards sFlow and adaptive polling sampling for deep learning based DDoS detection in SDN. Future Generat Comput Syst 2020;111:763–79.
[25] Afsaneh Banitalebi Dehkordi, Soltanaghaei MohammadReza, Zamani Boroujeni Farsad. The ddos attacks detection through machine learning and statistical methods in sdn. J Supercomput 2020:1–33.
[26] Bensalah Faycal, Elkamoun Najib, Baddi Youssef. SDNStat-sec: a statistical defense mechanism against DDoS attacks in SDN-based VANET. In: Advances on smart and soft computing. Springer; 2021. p. 527–40.
[27] Arivudainambi D, Varun Kumar KA, Chakkaravarthy S Sibi. Lion ids: a meta-heuristics approach to detect ddos attacks against software-defined networks. Neural Comput Appl 2019;31(5):1491–501.
[28] Saad Redhwan MA, Anbar Mohammed, Manickam Selvakumar. Rule-based detection technique for icmpv6 anomalous behaviour. Neural Comput Appl 2018;30(12):3815–24.
[29] Kalkan Kubra, Gur Gurkan, Alagoz Fatih. Defense mechanisms against DDoS attacks in SDN environment. IEEE Commun Mag 2017;55(9):175–9.
[30] Rochak Swami, Mayank Dave, and Virender Ranga. Detection and analysis of TCP-SYN DDoS attack in software-defined networking. Wireless Pers Commun, (in press).
[31] Kalkan Kübra, Gür Gürkan, Alagöz Fatih. SDNScore: a statistical defense mechanism against DDoS attacks in SDN environment. In: IEEE symposium on computers and communications (ISCC). IEEE; 2017. p. 669–75. 2017.
[32] Ramin Fadaei Fouladi, Ermis Orhan, Anarim Emin. A DDoS attack detection and defense scheme using time-series analysis for SDN. J Inf Secur Appl 2020;54:102587.
[33] Conti Mauro, Gangwal Ankit, Singh Gaur Manoj. A comprehensive and effective mechanism for DDoS detection in SDN. In: International conference on wireless and mobile computing, networking and communications (WiMob). IEEE; 2017. p. 1–8. 2017 IEEE 13th.
[34] Andrés Felipe Murillo Piedrahita, Rueda Sandra, Mattos Diogo MF, Duarte Otto Carlos MB. FlowFence: a denial of service defense system for software defined networking. In: Global information infrastructure and networking symposium (GIIS). IEEE; 2015. p. 1–6. 2015.
[35] Phan The Duy, Van-Hau Pham. A role-based statistical mechanism for DDoS attack detection in SDN. In: 5th NAFOSTED conference on information and computer science (NICS). IEEE; 2018. p. 177–82. 2018.
[36] Mohammadi R, Javidan R, Conti M. SLICOTS: an SDN-based lightweight countermeasure for TCP SYN flooding attacks. IEEE Transactions on Network and Service Management June 2017;14(2):487–97.
[37] Buragohain Chaitanya, Medhi Nabajyoti. FlowTrApp: an SDN based architecture for DDoS attack detection and mitigation in data centers. In: 3rd international conference on signal processing and integrated networks (SPIN). IEEE; 2016. p. 519–24. 2016.
[38] Barnett Vic, Lewis Toby. Outliers in statistical data. 1984.
[39] Hoaglin David C, John W. Tukey and data analysis. Statistical Science; 2003. p. 311–8.
[40] Mininet. http://mininet.org/; 2019.
[41] Open vSwitch [Online], https://www.openvswitch.org//. [Accessed December 2018]. Accessed.
[42] Ryu. https://www.osrg.github.io/ryu/; 2017.
[43] Scapy. https://scapy.readthedocs.io/en/latest/; 2008.