Computers and Electrical Engineering 101 (2022) 108019

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

Software defined network-based HTTP flooding attack defender☆

Reza Mohammadi [a,*], Chhagan Lal [b], Mauro Conti [c,b], Lokesh Sharma [d]

[a] Computer Engineering Department, Bu-Ali Sina University, Ahmadi Rooshan Ave., Hamedan, 38695-65178, Hamedan, Iran
[b] TU Delft, Mekelweg 5, 2628 CD, 2600 Delft, Netherlands
[c] University of Padua, Via Trieste, 63-35121, Padova, Italy
[d] Manipal University Jaipur, Off Jaipur-Ajmer Expressway, Jaipur 30307, India

ARTICLE INFO

Keywords:
HTTP flooding attack
SDN
Entropy
Hellinger distance

ABSTRACT

In recent years, the explosive growth of the Internet has led to an increment in the number of Distributed Denial of Service (DDoS) attacks. *HTTP Flooding* is a critical DDoS attack that targets HTTP servers to prohibit users from receiving HTTP services. Moreover, it saturates the link bandwidth and consumes network resources. Because the attack is launched at the application layer, it is difficult to defend against it using current countermeasures such as firewall or Intrusion Prevention System (IPS).

In this paper, we propose *SHFD*, which leverages the Software-Defined Networking (SDN) paradigm to mitigate HTTP flooding attacks. We implement SHFD as a defender module on the SDN controller to detect and mitigate the attack in the first place. Experimental results gathered from Mininet confirm that SHFD brings a significant improvement of 13% in detection time and 29% in the number of blocked malicious flows compared to the state-of-the-art approaches.

1. Introduction

In the last few decades, Internet users have increased significantly. At the same time, the network providers are improving the core infrastructure accordingly to provide fast and uninterruptible services to the end-users. It aims to support enhanced customer satisfaction and increase the new customers base. HTTP is one of the most used communication protocols to perform daily affairs, such as checking email, downloading required files, and fulfilling research-related activities [1]. This makes it an attractive target for attackers to launch Denial of Service (DoS) attacks against HTTP servers to make them unavailable for legitimate users. In most cases, the attackers perform Distributed DoS (DDoS) attacks and exploit the resource of other hosts to launch an HTTP flooding attack. It makes the attack stronger and more effective. In DDoS, many compromised hosts (botnet) send a high number of GET or POST requests toward the HTTP server to overload and exhaust its resources [2]. As a result, the server will crash and not deliver any services to legitimate users. HTTP flooding is the most common attack implemented at the application layer [3], and its detection by the network equipment in legacy networks is challenging.

Software Defined Networking (SDN) is a new framework for network management that decouples data plane from the control plane. SDN-based network architecture facilitates dynamic configuration and enables network providers to program the networks to provide desirable services and satisfy Service Level Agreement (SLA) for their customers [4,5]. Furthermore, SDN can be seen as an opportunity to establish security mechanisms to detect and prevent malicious attacks that cause disturbance in receiving service from servers by legitimate users [6].

☆ This paper is for regular issues of CAEE. Reviews were processed by Associate Editor Dr. A. Pasumpon Pandian and recommended for publication.

* Corresponding author.
E-mail address: r.mohammadi@basu.ac.ir (R. Mohammadi).

https://doi.org/10.1016/j.compeleceng.2022.108019
Received 5 November 2021; Received in revised form 17 April 2022; Accepted 18 April 2022
Available online 13 May 2022

1.1. Motivation and objectives

Application layer attacks such as HTTP flooding is challenging to detect by the legacy network's active equipment (i.e., routers or switches). For this reason, legacy network designers use firewalls or Intrusion Detection Systems (IDSs) to protect HTTP servers from being affected by the various attacks [7]. As another solution, they might configure the servers to serve only a limited number of requests for a specific host. Unfortunately, these solutions can only protect the target servers. They cannot prevent network resource exhaustion like link bandwidth, CPU wastage, and memory consumption at active devices like switches and routers. Moreover, these solutions might not detect the origin of DDoS attacks properly.

To overcome the limitations mentioned above, we propose a new SDN-based solution to detect and mitigate HTTP flooding attacks early, thus keeping the servers and network resources safe from disturbance and damage caused by the attacks. To this end, we leverage the programming capabilities of SDN and implement our proposed solution, called SHFD, as a security module running at the SDN controller. This module periodically collects the network statistics and uses Entropy to check whether the network is in a safe or unsafe state. If the network is unsafe, SHFD uses Hellinger distance to detect which hosts perform the unusual behavior. Then, it blocks the suspicious hosts at the edge of the network. This policy protects the network entities from resource exhaustion and prevents target servers from being overloaded by malicious hosts. One of the advantages of our proposed solution is that it can detect both *slow and high rate HTTP flooding attacks* in their early phases, thus reduces the adverse impact. Specifically, our main contributions can be summarized as follows.

- We propose a new security module named SHFD for detecting slow and high rate HTTP flooding attacks in SDN-based networks.
- We propose the use of Shannon's Entropy and Hellinger distance techniques in SHFD to provide a more accurate classification of legitimate and malicious HTTP flows.
- We implement the performance evaluation testbed by installing SHFD on Ryu controller, and we test its performance over different attack rates. Moreover, we compare the performance of SHFD against the state-of-the-art solutions for HTTP flooding in different SDN-based networks.

1.2. Organization

The rest of this paper is organized as follows. Section 2 presents some preliminaries about HTTP flooding attacks and mathematical materials used in SHFD. In Section 4, we review SDN and non-SDN related work for HTTP flooding. Section 7 presents our proposed solution (i.e., SHFD) and explains it in detail. In Section 5, experimental settings and the performance evaluation are presented. Finally, we conclude the paper in Section 6.

2. Preliminaries

In this section, first, we will describe the HTTP flooding attack in detail. Next, we discuss some mathematical concepts regarding Entropy and Hellinger distance.

2.1. HTTP flooding attack

HTTP flooding attack is a type of volumetric attack in which a group of compromised hosts (botnet) sends GET or POST requests to overwhelm a web server [8]. The goal is to make the target web server out of service. Moreover, bandwidth saturation and network equipment resource exhaustion are other drawbacks of this attack. Usually, one can implement an HTTP flooding attack in various ways, but two critical and popular methods are as follows [9,10].

2.1.1. Slow rate HTTP flooding

Also known as Slowloris [11], it establishes many connections to the victim web server and attempts to hold the connections open as long as possible. In a normal HTTP request, as depicted in Fig. 1, each line in the request message ends with CRLF (Carriage Return + Line Feed) character. Also, the message finishes with two CRLF characters, which denotes a blank line [9]. Upon receiving this blank line, the web server sends an HTTP response. Until it receives the blank line, it waits for the client to complete the headers and puts the blank line at the end of the message. This waiting leads to server resource consumption (overflows the maximum concurrent connection pool of web server). If the number of incomplete HTTP connections is too high, then the web server stops responding and finally crashes [10]. As shown in Fig. 1, Slowloris misuses this behavior and sends many incomplete HTTP requests followed by incomplete header fields with random values for each connection with delay. It does not finish the requests with a blank line. In nutshell, it sends incomplete HTTP GET request and forces the target web server to maintain the connection open for a long time. In a DDoS attack, a malicious attacker compromise many hosts, install Slowloris on them, and finally command them to create a huge number of incomplete HTTP connection to a victim web server.

2.1.2. High rate HTTP flood

Like any DDoS attack, it aims to bring down the victim web server and prohibit legitimate users from receiving HTTP services. The compromised hosts persistently send a massive number of complete HTTP POST or GET requests toward the web server, leading to a denial of the server's acceptance of benign users' requests. In most cases, the attackers request the large size web pages to make the attack more effective and also saturate the network bandwidth [10].

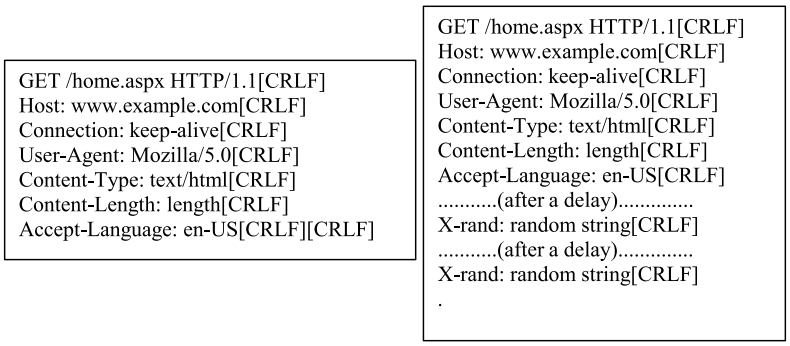


Fig. 1. Complete vs. incomplete (Slowloris) HTTP GET request.

2.2. Shannon's entropy

In information theory, Claude Shannon proposed entropy as a metric to measure the randomness or uncertainty of data [12]. The entropy of a discrete random variable is defined as follows.

$$E(X) = \sum_{i=1}^{n} -P(r_i)\log_2 P(r_i), \tag{1}$$

where, n is the number of outcome, $P(r_i)$ is the probability of the ith occurrence of random variable in the data set. Eq. (1) denotes that maximum entropy can be achieved if the random variables follow the uniform probability. Contrariwise, uncertainty decreases when the random variables do not follow uniform probability. This concept can be used in various fields such as detection of abnormal behavior of users in a network [13,14]. Accordingly, in this paper, we use this concept in SHFD to detect the presence of a possible attack in SDN-based networks. To normalize Eq. (1) and calculate the entropy in range [0, 1], we use Eq. (2) as follows [15].

$$NE(X) = \frac{E(X)}{\log_2 n}. \tag{2}$$

Furthermore, in SHFD, we use the number of HTTP flows in a specific time window as a random variable for Eq. (1) as follows.

$$W = \left\{(r_1, s_1), (r_2, s_2), \ldots., (r_n, s_n)\right\}, \tag{3}$$

where r_i is source IP address of the ith host, s_i is the number incomplete HTTP connections, and n is the number of clients in the network.

2.3. Hellinger distance

In a network, many users behave differently while using the network. Therefore, the probability distribution of traffic flows associated with a user is not the same as others. Hellinger distance is one of the most valuable metrics to measure the similarity of two different probability distributions [16]. Using it, the similarity of behavior of network users can be calculated as follows [17].

$$HD(P, Q) = \frac{1}{\sqrt{2}}\left(\sqrt{\sum_{i=1}^{k}(\sqrt{p_i} - \sqrt{q_i})^2}\right), \tag{4}$$

where $P = \left\{p_1, p_2, \ldots., p_k\right\}$ and $Q = \left\{q_1, q_2, \ldots., q_k\right\}$ are two discrete probability distributions, each of which consists k different samples. In SHFD, we use Hellinger distance to measure the difference between normal and abnormal behavior of users. In our schema, P and Q consists of the number of HTTP flows for each host time windows.

3. Related work

This section reviews the relevant state-of-the-art HTTP flooding defense mechanisms in legacy networks and SDN.

3.1. HTTP flooding defense in legacy networks

In legacy networks, the flow traffic of application-layer attacks such as HTTP flooding should be investigated on the application layer. With the absence of a centralized entity and lack of programmability, attack detection is more complex than SDN-based networks. Hence, the other layers in the devices, such as a hub, switch, router, and transport layer gateways, cannot detect

application-layer attacks. For this reason, most of the solutions proposed by the researchers rely on detecting these types of attacks at the target server or in an Intrusion Detection and Prevention System(IDPS).

Authors in [18] propose a detection mechanism, namely HAP, for HTTP flooding attacks in a cloud environment. To detect the attack, HAP uses web server logs and extracts valuable features from them. Next, the anomalies are detected by applying the affinity propagation clustering technique. Although HAP has a high detection rate in detecting the attacks in the dataset or regenerated data in a real network, it only relies on the server log data. It cannot prevent the adverse effects of HTTP flooding on network bandwidth and resources. Another solution for the detection of HTTP flooding in the cloud environment is proposed in [19]. The solution uses a monitoring and classifier module at the edge of the cloud, which monitors the incoming traffic from the tenants and detects the slow HTTP requests. Then, it puts the source of the attack on a blocklist. Legitimate requests are sent toward the servers and processed by them. This solution is also implemented on the cloud side and, thus, cannot prohibit resource and bandwidth exhaustion.

In [20] the authors proposed a Fuzzy-based system on the server-side to recognize whether a network user behavior is legitimate. The Fuzzy system takes two inputs: Request Index and Repetition Index. Then, the output is given to a scheduler to decide whether to permit or deny a user's request. In addition to these solutions, some other techniques use the machine learning approach to detect the HTTP flooding attack [21]. In general, all machine learning and non-machine learning solutions focus on detecting the attack on the server-side in which the network resource exhaustion has not been considered.

3.2. HTTP flooding defense in SDN-based networks

Due to the novelty of SDN, there are some works in the detection or prevention of HTTP Flooding attacks in SDN-based networks. In [22] the authors introduced SHDA as an SDN-based defense mechanism to detect and mitigate slow HTTP DDoS attacks. In SHDA, after a certain number of incomplete HTTP requests originated from a specific host (i.e., C_{th}), the web server wants the SDN controller to act as a proxy for future HTTP requests. In this step, the controller sets a timer (i.e., T_c) and counter (i.e., N_{th}) for the host and monitors the future HTTP traffic. If the number of incomplete requests violates a threshold before the timer expiration, SHDA considers the host malicious and blocks it. Although SHDA is a simple mechanism, it is sensitive to the value of the timer and threshold. Unsuitable adjusting of these values leads to false positives. Moreover, it is needed to change the web server to communicate with the controller, which is impractical in most situations. Similar to SHDA, [23] proposes a mitigation method for HTTP flooding in SDN-based networks. This solution detects the attack on the server-side. To do this, the servers count the number of incoming requests. If this number for a host in a certain time is greater than a predefined threshold, the server notifies the controller to block that host. However, this technique has most of the same disadvantages as SHDA. In [24] the authors have proposed a countermeasure for HTTP flooding, which needs collaboration between the targeted web server and SDN controller. In their method, the web server monitors the incoming HTTP requests to identify the suspected attack request. If the number of requests exceeds a threshold, the web server stops processing and forwards the request to the controller. Finally, the controller blocks the origin of that request. One of the main disadvantages of this method is that it needs to change the web server to monitor the requests and inform the controller. Thus, it imposes a considerable overhead on the web server.

In [25] the authors introduced a system for mitigating Slow DDoS attacks in SDN. The system is implemented as a separate host in the network, and it receives a copy of all traffic flows in the network. To check the presence of an attack, their system, first, measures the Round Trip Time (RTT) value of the traffic flows. If this value is exceeded a specific value, the second phase will be started. The second phase classifies the attacks based on their behavior. Upon detection of an attack, the system will notify the SDN controller to block the attacker. Their system cannot prevent bandwidth exhaustion, leading to false positives due to the improper threshold value. In [26], the authors have proposed a new hardware-based countermeasure using FPGA technology. In their method, if the number of HTTP GET requests violates the predefined threshold in a specific time interval, the source IP address of the requester is added to a blacklist. For efficient memory utilization, Source IP, Destination IP, URL is first hashed and then stored in the blacklist. The hosts whose address is on the blacklist will be blocked in the mitigation phase. Although their proposed mechanism detects and mitigates HTTP Flood attacks, it needs to use special FPGA-based hardware, which is impossible in most situations.

A machine learning-based detection method for various DDoS attacks in SDN has been proposed in [27]. To do this, the authors simulate different types of attacks in a simulation environment. Then, the controller gathers some critical features of traffic flows from the switches. Finally, it uses various classifying algorithms for the features to detect which traffic is normal or abnormal. Their simulation results showed that the decision tree algorithm outperforms other techniques in detecting anomalies. However, the focus of their method is not on HTTP flooding and only can detect the presence of an attack in the network. Moreover, it cannot recognize and block the source of the attack. Furthermore, machine learning techniques are data-hungry and need many data for accurate detection [28]. It is worth noting that because in SDN, the controller plays a vital role, implementing machine learning techniques on the controller needs a high level of computational resources. For this reason, these techniques are not appropriate solutions in a resource-limited SDN environment. At the end of this section, we summarize the related works in Table Table 1.

We found that very few works have been proposed to effectively mitigate HTTP flooding in SDN from the above literature review. Moreover, most existing solutions rely on considering threshold values that can cause false-positive or false-negative results. For this reason, our method is one of the pioneers' research in this scope and is helpful for network providers to apply in SDN networks.

4. Proposed method: SHFD

This section presents our proposed solution called SHFD to detect/prevent HTTP flooding attacks. SHFD leverages the capabilities of SDN to detect the attack situation and block the malicious hosts. Unlike most HTTP flooding countermeasures that use static thresholds, SHFD uses dynamic threshold values based on the behavior of network users.

Table 1
Comparison of existing approaches for HTTP flooding in SDN.

References	Year	Short Analysis	Disadvantages
[22], [23]	2017, 2020	• Implemented on web server and controller • Can detect the attack and its source • Server detects malicious behavior using thresholds and inform controller to block the source of the attack	• Uses two threshold values and one timer • Might produce inaccurate results • Sensitive to the behavior of users • Needs to change server source code
[26]	2017	• Implemented on FPGA-based openflow switches • Can detect the attack and its source • Switch reacts to malicious behavior if the thresholds exceed • Switch blocks the source of the attack	• Uses a threshold value for each host and URL • Sensitive to the behavior of users • Might produce inaccurate results • it is Only applicable for FPGA-based OF switches
[24]	2021	• Implemented on the target web server • Can detect the attack and its source • Detector reacts to malicious behavior if the thresholds exceed • Detector informs the controller to block the source of the attack	• Uses a threshold value for detection • Imposes considerable overhead on the target web server for attack detection • Needs to change the web server to enable it for attack detection
[25]	2018	• Implemented on a separate machine in the network • Can detect the attack and its source • Detector reacts to malicious behavior if the thresholds exceed • Detector informs the controller to block the source of the attack	• Uses a threshold value for detection • needs to measure RTT for each flow • Might produce inaccurate results • needs an extra machine as a detector
[27]	2020	• Implemented on the SDN controller • Can only detect the presence of attack	• Uses machine learning for detection • needs dataset or generating data for learning the algorithms • Only can detect the attack • For accurate results, needs a lot of data
SHFD (our solution)	2022	• Implemented on the SDN controller • Can detect the presence of attack • Can recognize and block malicious hosts • Can be used to address low as well as high rate HTTP flooding	• It consumes some memory at controller to store data structures that keeps track of the host activities

4.1. Architecture

SHFD is implemented as a security module on the SDN controller and can be enabled by a network administrator to defend against HTTP flooding attacks. SHFD consists of two separate phases, which are as follows.

4.1.1. Attack detection phase

In this phase, SHFD listens to incoming new HTTP requests to the controller. These requests are TCP SYN packets to establish a TCP connection between the client and target web server, and after that, the client will send HTTP GET or POST messages. First, SHFD extracts the source IP address from the SYN packet and stores it in a list named *HostReqList*. This list specifies the number of incomplete HTTP headers during a time window for each host. Then, SHFD installs an end-to-end bidirectional path between the pair of *client-web server*. To track the future incoming packets for these flows, the first switch along the path is configured to forward the upcoming packets of the same flows to the next hop and deliver a copy to the controller. This configuration enables SHFD to check whether a host sends complete or incomplete HTTP requests after establishing the TCP connection. By applying this policy, SHFD can track each HTTP flow for any host in the network. Upon receiving an incomplete connection request, SHFD increments the counter of that host in *HostReqList* for the pair of source–destination IP addresses by 1. SHFD uses a time window that the network administrator defines to periodically monitor the network users' behavior. At the end of each time window, SHFD creates TW set using Eq. (3). Next, SHFD computes entropy (i.e., E_{Tw}) for the set and calculate normalized entropy value using Eqs. (1) and (2), respectively. Moreover, it stores *HostReqList* of each host into a vector named *HostReqVector*, and reset the counter of each host in *HostReqList* to zero for the next time window. If the entropy's value in the current time window violates the threshold value, SHFD suspects that the network is in an attack (unsafe) state. Therefore, it goes to the next phase to recognize the attack's origin. Furthermore, to decrease the detection time in case of attack, it decrements the current time window length for the next round by using Eq. (5) as follows.

$$TW_i = TW_{i-1} \times 0.8 \times \frac{TW_{InitValue}}{2}, \tag{5}$$

where $TW_{InitValue}$ is the initial value for TW, which is determined by the administrator at the initial step. Moreover, TW_{i-1} and TW_i are the previous and current time windows, respectively. Regarding Eq. (5), it is obvious that in case of entropy violation, the value of TW is in range $[\frac{TW_{InitValue}}{2}, TW_{InitValue}]$. By applying this policy, SHFD must check the entropy violation condition sooner

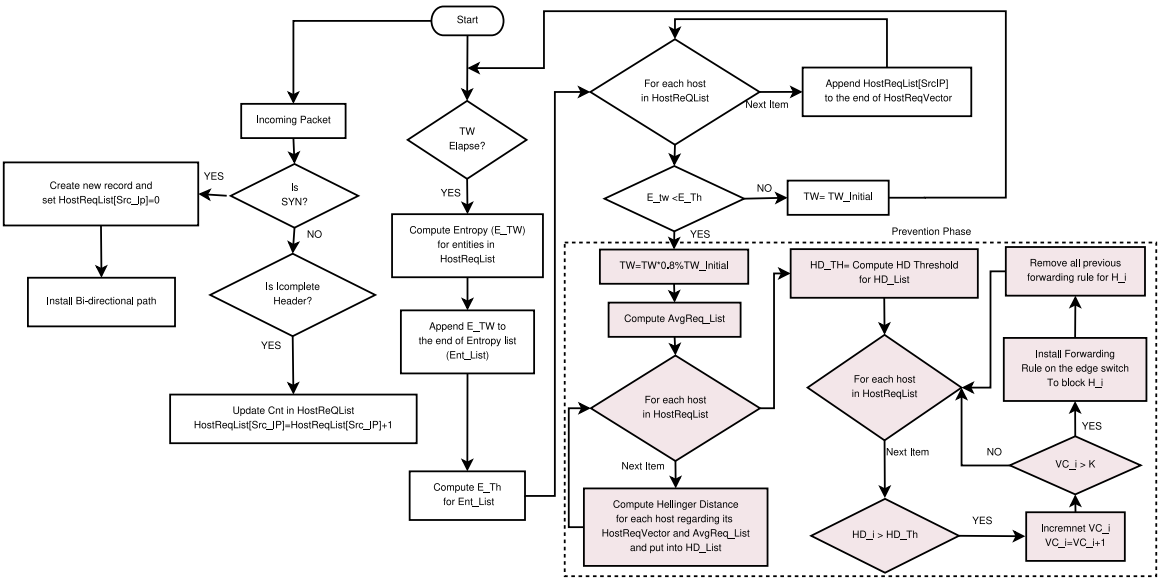


Fig. 2. Detection and prevention phases in SHFD.

in the next time window. Hence, if the network behavior is an attack, it can detect it sooner. If the entropy value does not violate the threshold, SHFD set TW to $TW_{InitValue}$ to reduce the overhead of unnecessary monitoring.

As aforementioned, one of the benefits of SHFD is that it uses an adaptive threshold for entropy. To do this, it stores entropy values in a list named E_List. Once SHFD compares the current entropy with a threshold (i.e., E_{Th}), it calculates the threshold as follows.

$$E_{Th} = \mu - 3 \times \sigma, \tag{6}$$

where μ and σ are the mean and standard deviation of the entropy values in E_List, respectively. Eq. (6) helps SHFD to adapt itself to the network conditions. There is no need to reset the threshold by the network administrator. The admin only needs to provide an initial threshold value. It is worth mentioning that, since SHFD uses the IP address of the hosts for detecting the origin of the attack, if an attacker spoofs the IP address, it can mislead SHFD. Fortunately, there is much literature to detect the IP spoofing attacks in SDN [29,30]. Therefore, an administrator can use one of these defense mechanisms together with SHFD. Algorithm 1 shows the pseudo-code of SHFD. In the *while* loop in line 5, the controller checks the incoming incomplete HTTP request and increases the counter for each source IP (lines 8,9). In *if* statement of line 10, it checks the timer, and upon timer expiration, it calculates the threshold value. Then, in *if* statement in line 14, it checks whether the Entropy of the current time window is less than the Entropy threshold. If this condition is *true*, then the network behavior is abnormal, and the suspected hosts should be recognized. To do this, from lines 15 to 21, it first calculates the average number of requests that all requester hosts send in the network and then computes Hellinger distance for each of them. Finally, in the *foreach* loop in line 22, it calculates the number of violations of each host from the Hellinger distance threshold. If the number of violations is greater than K for each host, then the host is blocked (line 25).

4.1.2. Attack prevention phase

In this phase, SHFD begins to check which hosts are suspicious of being malicious. To do this, first, it creates a vector named *AvgReq_List* to calculate average incomplete HTTP headers for all hosts during different time windows so far. Then, SHFD computes Hellinger distance between the *HostReqVector* of each host and *AvgReq_List* using Eq. (7) to recognize abnormal behavior. This value is also stored in a list, and then the average of Hellinger distances of all hosts (i.e., HD_{Th}) is calculated. Now, SHFD checks the Hellinger distance against the HD_{Th}. If this value for each host is larger than HD_{Th}, SHFD can recognize that the host is a potential attacker. To reduce the probability of false-positive recognition, SHFD considers a counter named *VC (Violation Count)* for each host in this step. If the number of Hellinger distance violations of a host is greater than K, then SHFD can be sure that the host is an attacker. Next, it installs a forwarding rule to block the attacker host on the edge switch to which the host is connected. Then, it sends a command to all switches in the network to remove any forwarding rule related to the IP address of the attacker.

It is worth mentioning that, considering K is optional, we have considered it as a parameter for administrators to enable them to change the strictness of SHFD in attacker recognition. The policy used in SHFD has three benefits: (i) it preserves the target web server from the subsequent attacks from the same attacker, (ii) it mitigates the bandwidth saturation and resource exhaustion in network devices, and (iii) it reduces the number of wasted forwarding rules in the flow table on the network switches. Fig. 2 shows the flow diagram of the attack detection and prevention phase of SHFD.

For the sake of simplicity, we illustrate functionalities of SHFD for a malicious host in a scenario with one switch in Fig. 3.

4.2. Complexity of SHFD

The time complexity of SHFD depends on the number of malicious hosts. According to Algorithm 1, if we consider n is the number of hosts in the network and assume that all of them are attackers, and k is the number of network switches, then the computational complexity of SHFD in the worst case is $O(6n + 2n^2 + k) \sim O(n^2 + k)$. It implies that SHFD has polynomial complexity, and applying it to defense against HTTP flooding does not lead to significant overheads on the SDN controller.

Algorithm 1 SHFD: Detection and Prevention Phase

Input: $PACKET\_IN(P)$ // New packet arrived to the controller
foreach $SrcIP \in \mathcal{N}etwork\_Hosts$ **do**
  $HRL[SrcIP] \leftarrow 0$; // HostReqList
  $VC[SrcIP] \leftarrow 0$; // Violation Counter
  $HRV_{SrcIP} \leftarrow []$; // HostReqVetcor
while *true* **do**
  $P \leftarrow New\_Packet()$
  if P==*TCP_SYN* **then**
    $Install\_BiDirectionalPath(P.SrcIP, Webserver)$
  if P==*Incomplete HTTP Header* **then**
    $HRL[P.SrcIP] \leftarrow HRL[P.SrcIP] + 1$;
  // Time WindowTimer Expiration
  if $TW exceeded$ **then**
    $E_{Tw} = ComputeEntropy(HRL)$
      $Ent\_List \leftarrow Ent\_List \cup E_{Tw}$
      $\mu = Mean(Ent\_List)$
      $\sigma = Standard Deviation(Ent\_List)$
      $E_{TH} = \mu - 3 \times \sigma$ // Entropy Threshold
    foreach $SrcIP \in \mathcal{H}RL$ **do**
      $HRV_{SrcIP} \leftarrow HRV_{SrcIP} \cup HRL[SrcIP]$
      $HRL[SrcIP] \leftarrow 0$;
    if $E_{Tw} \leq E_{TH}$ **then**
      $TW = TW \times 0.8 \times \frac{TW_{Initial}}{2}$
      $AvgReq\_List \leftarrow []$
      foreach $m = 1$ *to* $Size(HRV)$ **do**
        $sum \leftarrow 0$
        for $SrcIP \in \mathcal{N}etwork\_Hosts$ **do**
          $sum \leftarrow sum + HRV_{SrcIP}[m]$
        $avg \leftarrow \frac{sum}{Size(Network\_Hosts)}$
        $AvgReq\_List \leftarrow AvgReq\_List \cup avg$
      $HD\_List \leftarrow []$
      foreach $SrcIP \in \mathcal{N}etwork\_Hosts$ **do**
        $H_{SrcIP} = HellingerDistance(HRV_{SrcIP}, AvgReq\_List)$;
        $HD\_List \leftarrow HD\_List \cup H_{SrcIP}$
      $H_{Th} = Mean(HD\_List)$ // Hellinger Distance Threshold
      foreach $SrcIP \in \mathcal{N}etwork\_Hosts$ **do**
        if $H_{srcIP} > H_{Th}$ **then**
          $VC[SrcIP] = VC[SrcIP] + 1$
        if $VC[SrcIP] > K$ **then**
          $RemoveRuleFromSwitches(SrcIP)$
          $InstallBlockRule(SrcIP)$

5. Performance evaluation

In this section, we conduct a comprehensive simulation study and analyze the results to evaluate the performance of SHFD and compare it with SHDA [22] and normal SDN. There is no defense mechanism against HTTP flooding attacks in normal SDN, and we will show how SHFD can improve the security of normal SDN.

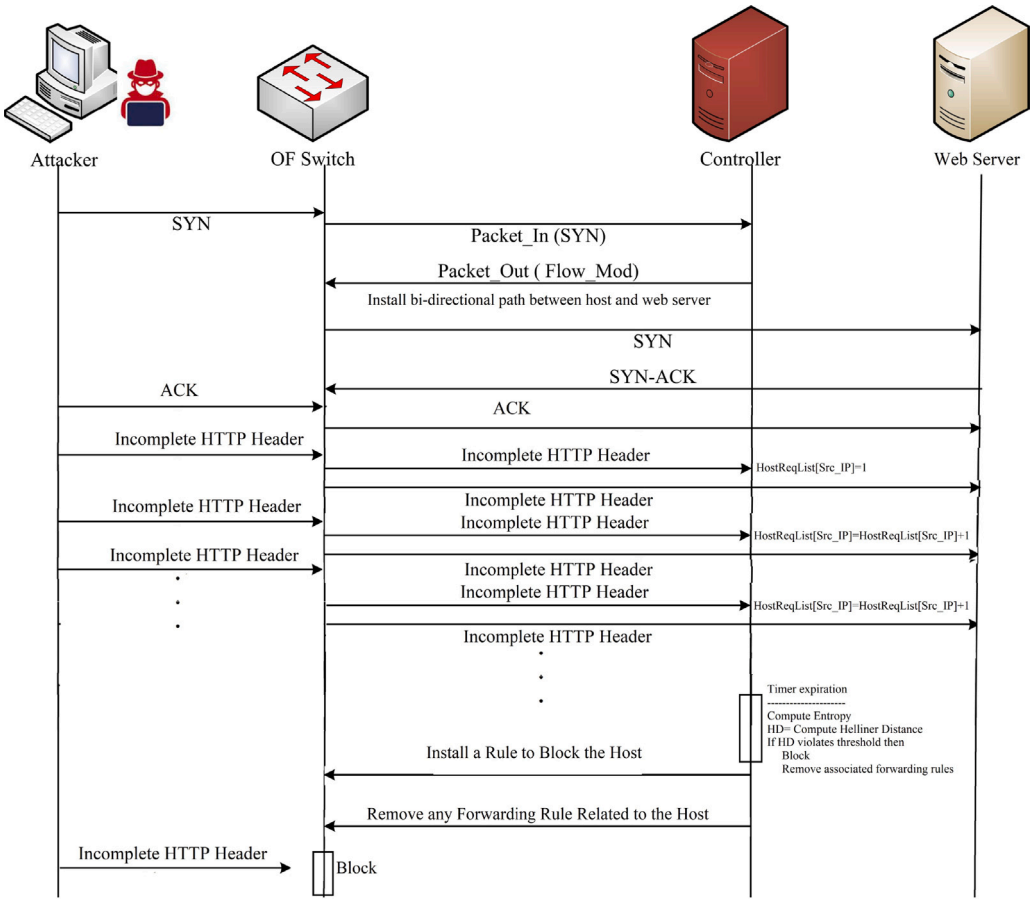


Fig. 3. SHFD functionalities for malicious host.

5.1. Simulation setup

We implement SHFD as a security module in Ryu controller. To simulate the data plane of the testbed network, we use Mininet, a popular SDN simulation tool. We run our experiments on a PC with four processing cores and 3 GB RAM. As mentioned in Section 4, SHDA is one of the state-of-the-art solutions to tackle HTTP flooding attacks implemented in the controller. As mentioned in Section 4, SHDA uses three threshold values (i.e., $C_{th}, T_c, and\, N_{th}$) to detect an HTTP flooding attack. In our simulation, we set these thresholds to $C_{th} = 512, T_c = 20, and\, N_{th} = 20$.

To better evaluate the results, we also consider *SDN_Normal_With_Attack* and *SDN_Normal_Without_Attack* which denote normal SDN scenarios without any security module for HTTP flooding attack in case of attack and non-attack, respectively. Simulations are performed on the testbed network, which is shown in Fig. 4. The testbed network consists of six OpenFlow switches which are controlled by the SDN controller. SHFD is implemented in this controller and performs the detection and mitigation processes. As it can be seen in this figure, there are three legitimate hosts (H1-H3), six attackers (H4-H9), and two flash crowds (H10, H11). We have considered ten different HTML files, each of which has a size of 51 Byte. Benign hosts send complete HTTP GET requests to access one of these HTML files to a web server randomly between 10 and 15 s. To investigate the reaction of SHFD against flash crowd hosts, we set these hosts to send an incomplete HTTP request for a short time (10 s). We have used SlowHTTPtest to perform HTTP flooding attack. The attacker hosts use this tool to send incomplete HTTP requests at different rates. In order to evaluate the performance of SHFD in the different traffic loads of attacks, we have configured five different scenarios in which the attacker hosts send from 10 to 50 incomplete HTTP requests per second. In fact, the attack rate indicates the number of incomplete HTTP connections established and maintained open for a long time. Simulation time for all scenarios is set to 600 s, and the attack start time in all scenarios is set to 220 s. It is because SHFD detects the presence of an attack and recognizes malicious hosts using Entropy and Hellinger distance, respectively. Therefore, the simulation must include normal behavior to identify the subsequent abnormal behaviors. In the testbed network, there is no attack from the beginning of the simulation until the second of 220th second. Hence the standard deviation is low. As the attack starts, the standard deviation increases, allowing SHFD to detect the presence of the attack by computing the Entropy.

We evaluate the experimental results using the following performance metrics.

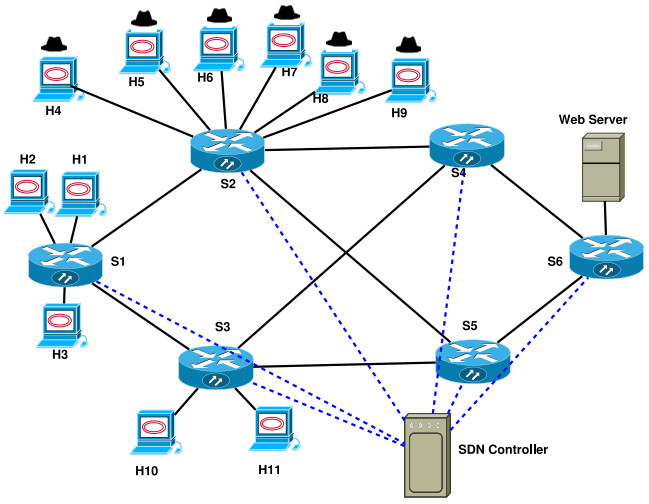
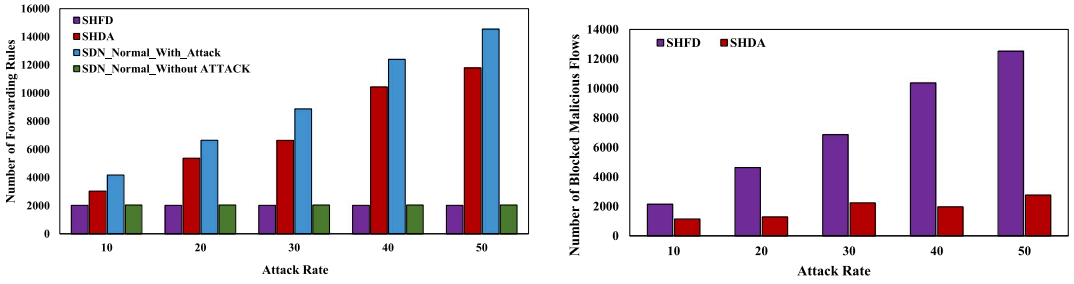


Fig. 4. The test-bed network.



((a)) Number of forwarding rules ((b)) Number of blocked rules

Fig. 5. Number of forwarding and blocked rules vs. attack rate.

- Number of installed forwarding rules: denotes the total number of remaining forwarding entries in OF-switches at the end of a simulation.
- Attack detection time: the time between sending the first incomplete HTTP request by an attacker and detecting the attack by the defense mechanism.
- Bandwidth Consumption: denotes the amount of traffic traversed via OF-switches during a simulation run.
- Number of Blocked Malicious flows: the number of incomplete HTTP requests detected and blocked by the defense mechanism.
- F1-Score: is a measure to test the accuracy of a defense mechanism in detecting an attack. The higher value for F1-Score means higher accuracy. F1-Score is defined as follows.

$$F1\_Score = \frac{TP}{TP + \frac{FP+FN}{2}} \tag{7}$$

where TP is true positive, FN is false negative and FP is false positive rate, respectively.

6. Results

The number of forwarding rules in the OF-switches flow table is a crucial parameter that should be considered for a defense mechanism. Most of the attacks in SDN attempt to exhaust the flow tables. If the flow table of an OF switch is full, installing a new rule by the controller has to remove one of the previously installed rules. This situation might sacrifice a benign traffic flow.

As depicted in Fig. 5(a), one of the side effects of HTTP flooding attack is the flow table exhaustion at OF-switches. This figure shows that the difference between *SDN_Normal_Without_Attack* and *SDN_Normal_With_Attack* at the attack rate of 50 incomplete HTTP requests per second is approximately 13 000 entries. It denotes that the HTTP flooding attack has effectively populated the TCAM memory of switches. Fig. 5(a) also shows that SHFD outperforms SHDA. This is because of two reasons. First, upon detection of a malicious host, SHFD blocks that host at the edge switch and prevents it's future HTTP connections. This policy decreases the

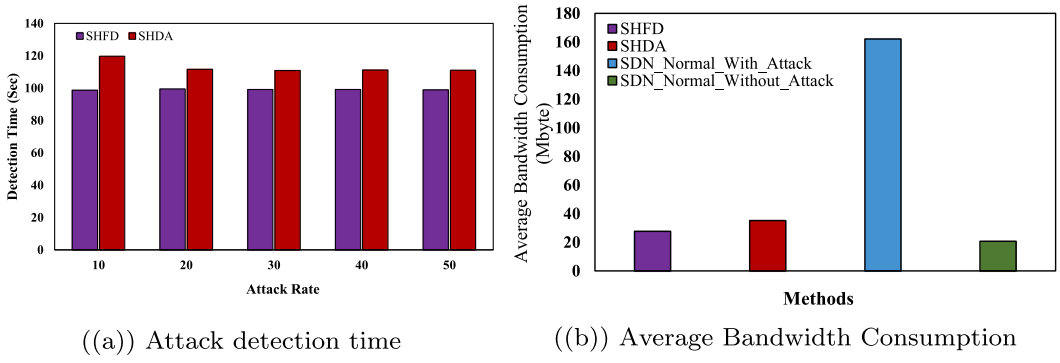


Fig. 6. Attack detection time and average bandwidth consumption vs. different attack rates.

number of installed forwarding rules on the switches. Second, after blocking the malicious host, SHFD removes all worthless rules related to the malicious host from all switches in the network. Unlike SHFD, SHDA only blocks the malicious host, but it does not remove the already installed flows of that host from the forwarding tables.

Accordingly to Figs. 5(a), 5(b) shows that the difference between SHFD and SHDA in terms of the number of blocked forwarding rules significantly increases with attack rate. It depicts that SHFD is an effective defense mechanism that decreases the TCAM memory of relevant switches along the path between the malicious hosts and web server by removing worthless rules related to the malicious hosts.

One of the primary essential factor for a defense mechanism, especially in HTTP flooding, is early detection of the attack so that the impact of the attack can be minimized or, in some cases, even the attack could be prevented in the first place. As depicted in Fig. 6(a), SHFD for different rates of attack has superiority to SHDA, and in all scenarios, it detects the attack sooner than SHDA. Early detection in SHFD leads to prohibiting future incomplete HTTP requests and, as a result, decreases the number of installed forwarding rules (as shown in Fig. 5(a)). Moreover, it causes lower web server resource consumption. As discussed in Section 4, SHDA uses multiple threshold values for attack detection, and for this reason, a network administrator needs to set these values correctly. Unfortunately, optimized value selection for these thresholds is difficult for any administrator due to network users' dynamic behavior. Unlike SHDA, SHFD employs Entropy for detection of the presence of attack in the network. This policy leads to take a prompt reaction against the attack. Furthermore, after detection, SHFD immediately begins the mitigation phase and computes Hellinger distance to recognize and block the malicious hosts. Moreover, adaptive threshold values in SHFD allow it to adapt itself to the network traffic behavior quickly.

The adverse effect of DDoS attacks in SDN includes saturation of the network bandwidth, increase in the switches' energy consumption, and packet drop (due to congestion) for other benign traffic flows. Fig. 7 shows the bandwidth consumption during over the time. It indicates that SHFD has lower bandwidth consumption from the start as it detects the attack sooner than SHDA. Moreover, to demonstrate the effectiveness of SHFD and SHDA in bandwidth reduction against high attack rate, we compute the average bandwidth consumption for the attack rate of 50 requests per second, as shown in Fig. 6(b). It illustrates how both SHFD and SHDA effectively protect the network against the adverse effect, providing sufficient bandwidth for benign network traffics.

As mentioned in Section 5.1, we have considered two benign hosts as the flash crowds, which send an incomplete HTTP request for 10 s, while other hosts in the testbed network normally request web pages from the web server. It is to check how SHFD treats the flash crowd traffic. A defense mechanism should correctly distinguish normal behavior from abnormal. As shown in Fig. 8, SHFD significantly outperforms SHDA in terms of F1-Score. The figure indicates that because SHFD does not rely on constant threshold values and uses Hellinger distance to find misbehavior hosts, it has a higher F1-Score than SHDA. In fact, using Hellinger distance helps to distinguish abnormal from a normal host. As depicted in Algorithm 1, if the Hellinger distance for a specific host violates from K, the host will be blocked. Because flash crowds only send many requests for a short period compared to the attackers, employing Hellinger distance in the detection process reduces the likelihood of considering a flash crowd as malicious behavior. Hence, Hellinger distance leads to achieving a higher value for F1-score in SHFD. Contrary, because SHDA uses constant values for threshold, it cannot detect flash crowd traffics, and for this reason, in our simulation testbed, it considers flash crowds as the attackers and blocks them.

7. Results for the high rate flooding

Unlike slow rate HTTP flooding, where the attackers leave the TCP connections open, the attackers complete their HTTP requests in the high rate flooding. However, the duration between consecutive requests is kept short, and hence they can quickly saturate network bandwidth and the resource of the target web server. Due to SHFD being implemented as a module on the SDN controller, it can also detect high rate HTTP flooding attacks. To do this, we only need to change SHFD to count complete HTTP requests instead of incomplete ones. This section adds this minor change to SHFD and evaluates its performance in detecting and mitigating a high rate of HTTP flooding attacks. For this reason, we keep all testbed and configuration of the previous section and only change the

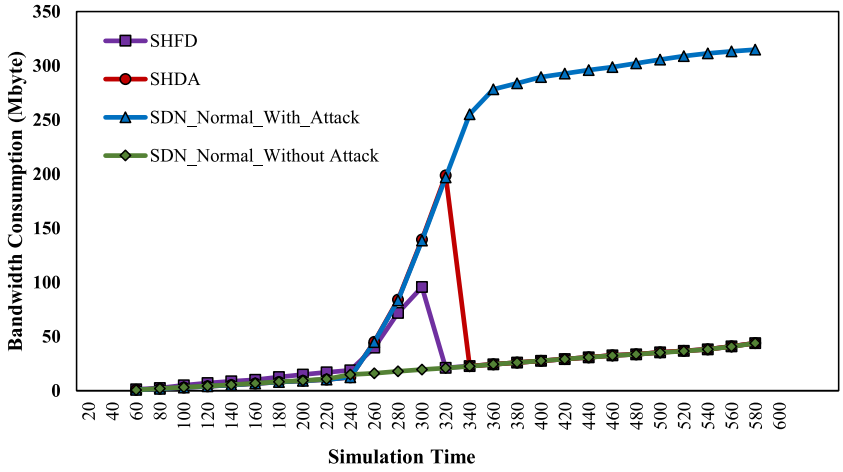


Fig. 7. Bandwidth consumption during simulation time.

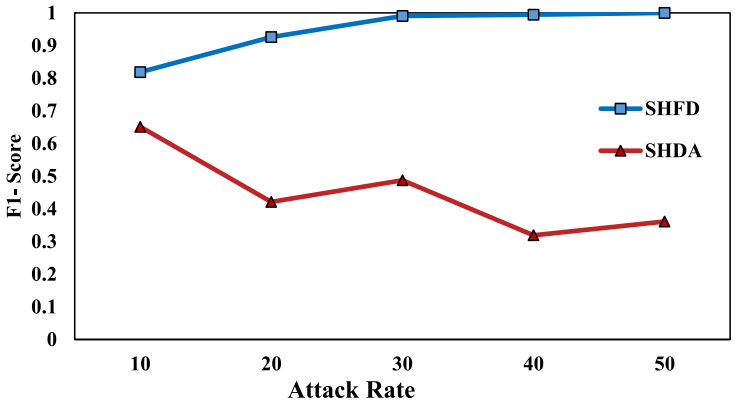


Fig. 8. F1-Score of defense mechanisms vs. different attack rates.

attackers to send complete HTTP requests from 10 to 30 requests/s. We do not consider SHDA for high rate HTTP flooding because, as we discussed in Section 2, it is a method for defense against slow rate attacks in which an attacker sends incomplete requests. Hence, it cannot be applied for complete HTTP requests.

Figs. 9 and 10(a) show that whether SHFD can preserve network bandwidth and also reduce the TCAM memory of OF-switches in case of high rate HTTP flooding attack.

As depicted in Fig. 10(b), SHFD also achieves good performance in terms of F1-Score, which means it can accurately detect and block high rate attacks.

In a nutshell, from Figs. 9 to 10(b), it can be concluded that SHFD can be used to defend against high rate HTTP flooding as well as low rate HTTP flooding attacks. It is worth mentioning that SHFD is an adaptive defense mechanism for HTTP flooding attacks, i.e., unlike other countermeasures, it does not need to set static values for many thresholds and parameters. An administrator only needs to set the initial values for TW (i.e., time window) and K (i.e., number of host violations from normal behavior). If the target network scenario's vulnerability toward DDoS attacks is high, the admin should set a lower value for TW. Otherwise, a higher value for TW should be set to reduce overheads. The results for a high rate HTTP flooding attack show that employing Entropy and Hellinger distance mechanisms is useful and lead to fast detection of these type of attacks.

8. Conclusions and future directions

We propose SHFD, a security module for SDN controllers to mitigate high/slow rate HTTP flooding attacks. SHFD monitors HTTP flows and recognizes abnormal HTTP requests, and blocks the source of malicious requests. SHFD uses statistical tools such as entropy and Hellinger distance to recognize the attack and its sources. The critical advantage of SHFD is that it blocks the attackers at the edge of a network and does not let them exhaust the network resources. Furthermore, unlike other statistical defense mechanisms, it uses dynamic and adaptable thresholds, which do not require administrator efforts to adjust threshold values. To investigate the performance of SHFD, we carried out a comprehensive evaluation in different attack scenarios. Experimental results show that SHFD

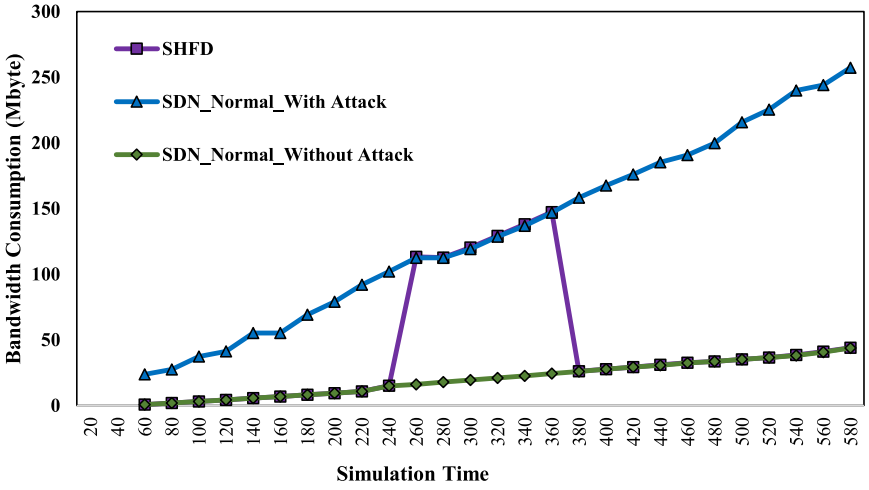


Fig. 9. Bandwidth consumption in high rate HTTP flooding during simulation time.

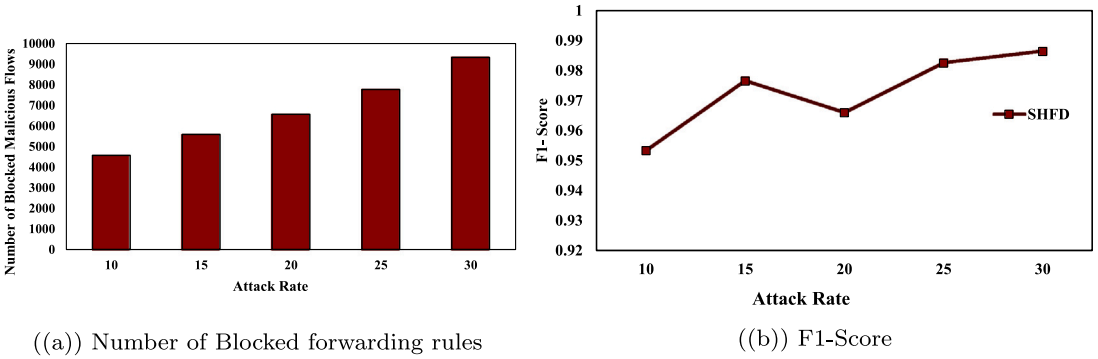


Fig. 10. Number of forwarding and blocked rules of SHFD in high rate HTTP flooding vs. attack rate.

significantly reduces the effects of HTTP flooding attacks compared to state-of-the-art methods. Results also confirm that applying SHFD in SDN can preserve the target web servers against an attacker.

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] Yuchen Z, Weiwei F, Xiaobin T. A solution for HTTP in-network caching based on P4. Front Data Comput 2020;2(3):75–86.
[2] Kesavamoorthy R, Alaguvathana P, Suganya R, Vigneshwaran P. Classification of DDoS attacks–A survey. Test Eng Manag 2020;83:12926–32.
[3] Singh P, Rehman SU, Manickam S. Performance analysis of emm an edos mitigation technique in cloud computing environment. In: International conference on advances in cyber security. Springer; 2019, p. 123–37.
[4] Prajapati A, Sakadasariya A, Patel J. Software defined network: Future of networking. In: 2018 2nd international conference on inventive systems and control (ICISC). IEEE; 2018, p. 1351–4.
[5] Rana DS, Dhondiyal SA, Chamoli SK. Software defined networking (SDN) challenges, issues and solution. Int J Comput Sci Eng 2019;7(1):884–9.
[6] Shaghaghi A, Kaafar MA, Buyya R, Jha S. Software-defined network (SDN) data plane security: issues, solutions, and future directions. In: Handbook of computer networks and cyber security. Springer; 2020, p. 341–87.
[7] Dhanapal A, Nithyanandam P. An OpenStack based cloud testbed framework for evaluating HTTP flooding attacks. Wirel Netw 2019;1–11.
[8] Verma A, Xaxa DK. A survey on HTTP flooding attack detection and mitigating methodologies. Int J Innov Adv Comput Sci 2016;5(5).
[9] Suroto S. A review of defense against slow HTTP attack. JOIV: Int J Inform Vis 2017;1(4):127–34.
[10] https://www.imperva.com/learn/ddos/ddos-attacks/ (last visited 3/27/2021).
[11] Sabri S, Ismail N, Hazzim A. Slowloris DoS attack based simulation. In: IOP conference series: Materials science and engineering, Vol. 1062. IOP Publishing; 2021, 012029.
[12] Shannon CE. A mathematical theory of communication. Bell Syst Tech J 1948;27(3):379–423.

[13] Sharshembiev K, Yoo S-M, Elmahdi E, Kim Y-K, Jeong G-H. Fail-safe mechanism using entropy based misbehavior classification and detection in vehicular ad hoc networks. In: 2019 international conference on internet of things (IThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData). IEEE; 2019, p. 123–8.
[14] Zhang N, Jaafar F, Malik Y. Low-rate dos attack detection using psd based entropy and machine learning. In: 2019 6th IEEE international conference on cyber security and cloud computing (CSCloud)/2019 5th IEEE international conference on edge computing and scalable cloud (EdgeCom). IEEE; 2019, p. 59–62.
[15] Kumar P, Tripathi M, Nehra A, Conti M, Lal C. SAFETY: Early detection and mitigation of TCP SYN flood utilizing entropy in SDN. IEEE Trans Netw Serv Manag 2018;15(4):1545–59.
[16] Bhattacharyya DK, Kalita JK. DDoS attacks: Evolution, detection, prevention, reaction, and tolerance. CRC Press; 2016.
[17] https://en.wikipedia.org/wiki/Hellinger_distance (last visited 3/27/2021).
[18] Sree TR, Bhanu SMS. HAP: detection of HTTP flooding attacks in cloud using diffusion map and affinity propagation clustering. IET Inf Secur 2018;13(3):188–200.
[19] Dhanapal A, Nithyanandam P. The slow HTTP distributed denial of service attack detection in cloud. Scalable Comput: Pract Exp 2019;20(2):285–98.
[20] Singh K, Singh P, Kumar K. Fuzzy-based user behavior characterization to detect HTTP-GET flood attacks. Int J Intell Syst Appl 2018;10(4):29.
[21] Sreeram I, Vuppala VPK. HTTP flood attack detection in application layer using machine learning metrics and bio inspired bat algorithm. Appl Comput Inform 2019;15(1):59–66.
[22] Hong K, Kim Y, Choi H, Park J. SDN-assisted slow HTTP DDoS attack defense method. IEEE Commun Lett 2017;22(4):688–91.
[23] Sanjeetha R, Shastry KA, Chetan H, Kanavalli A. Mitigating HTTP GET FLOOD DDoS attack using an SDN controller. In: 2020 international conference on recent trends on electronics, information, communication & technology (RTEICT). IEEE; 2020, p. 6–10.
[24] Park S, Kim Y, Choi H, Kyung Y, Park J. HTTP DDoS flooding attack mitigation in software-defined networking. IEICE Trans Inf Syst 2021;104(9):1496–9.
[25] Lukaseder T, Ghosh S, Kargl F. Mitigation of flooding and slow ddos attacks in a software-defined network. 2018, arXiv preprint arXiv:1808.05357.
[26] Viet AN, Van LP, Minh H-AN, Xuan HD, Ngoc NP, Huu TN. Mitigating HTTP GET flooding attacks in SDN using NetFPGA-based OpenFlow switch. In: 2017 14th international conference on electrical engineering/electronics, computer, telecommunications and information technology (ECTI-CON). IEEE; 2017, p. 660–3.
[27] Santos R, Souza D, Santo W, Ribeiro A, Moreno E. Machine learning algorithms to detect DDoS attacks in SDN. Concurr Comput: Pract Exper 2020;32(16):e5402.
[28] Restuccia F, D'Oro S, Melodia T. Securing the internet of things in the age of machine learning and software-defined networking. IEEE Internet Things J 2018;5(6):4829–42.
[29] Zhang C, Hu G, Chen G, Sangaiah AK, Zhang P, Yan X, Jiang W. Towards a SDN-based integrated architecture for mitigating IP spoofing attack. IEEE Access 2017;6:22764–77.
[30] Afek Y, Bremler-Barr A, Shafir L. Network anti-spoofing with SDN data plane. In: IEEE INFOCOM 2017-IEEE conference on computer communications. IEEE; 2017, p. 1–9.

Reza Mohammadi is an Assistant Professor in Computer Engineering at Bu-Ali Sina University since 2018. He received his M.Sc. and Ph.D. degrees in Computer Networking from Shiraz University of Technology in 2013 and 2017, respectively. His major fields of interest are SDN, heuristic algorithms, SDN security, Underwater Wireless Sensor Networks, Ad hoc Networks, Underground Wireless Sensor Networks, Internet of Things (IoT) and IoUT.

Chhagan Lal is currently working as a researcher at the Department of Intelligent Systems, TU Delft, Netherlands. He received his Ph.D. in Computer Science and Engineering from the Malaviya National Institute of Technology, Jaipur, India, in 2014. His current research areas include applications of network security, wireless networks, and blockchain technologies.

Mauro Conti is Full Professor at the University of Padua, Italy. He is also affiliated with TU Delft and University of Washington, Seattle. His main research interest is in the area of Security and Privacy. He is Senior Member of the IEEE and ACM. He is a member of the Blockchain Expert Panel of the Italian Government. He is Fellow of the Young Academy of Europe.

Lokesh Sharma is a Deputy Director-Admissions and Associate Professor in the Department of IT at Manipal University Jaipur. His area of research is QoS and QoE implementation in Mobile Ad Hoc Networks. Presently, he is exploring the areas of Software Defined Networks and IoT.