SYNGuard: Dynamic threshold-based SYN flood attack detection and mitigation in software-defined networks

Mohamed Rahouti | Kaiqi Xiong | Nasir Ghani | Farooq Shaikh

Abstract
SYN flood attacks (half-open attacks) have been proven a serious threat to software-defined networking (SDN)-enabled infrastructures. A variety of intrusion detection and prevention systems (IDPS) have been introduced for identifying and preventing such security threats, but they often result in significant performance overhead and response time. Therefore, those existing approaches are inflexible for large-scale networks and real-time applications. For this reason, a novel and adaptive threshold-based kernel-level intrusion detection and prevention system by leveraging SDN capabilities are proposed. The proposed systems to detect and mitigate the aforementioned threats within an SDN environment. The experimental results demonstrate the efficacy of the mechanism to detect and mitigate SYN flood attacks within an SDN environment.

1 | INTRODUCTION
Software-defined networking (SDN) has emerged as a next-generation networking technology in computer science and engineering. SDN technology allows for merging existing infrastructures of networking to a centralised framework where SDN-enabled devices such as Open vSwitch (OVS) must support an open communication protocol such as OpenFlow [1]. SDN has been used to address various networking problems such as intrusion detection and prevention and traffic monitoring. In SDN environments, the centralised controller allows for a flexible and efficacious configuration and network management to enhance communication performance and quality of service (QoS). Moreover, while benefiting from the controller's global view, administrators of network can leverage various SDN features and functionality to secure and optimise a networking environment through an integration of science de-militarised Zone (DMZ), a QoS rule set, or even implementing an in-line intrusion detection system (IDS) and/or intrusion prevention system (IPS).

However, the use of SDN-based technologies introduces many new attack vectors for malicious actors [2]. Some key examples here include denial of service (DoS) attacks [3], link discovery service (LDS) exploitation [4], and Man-in-the-Middle (MITM) attacks [2]. Indeed, many of these exploits have already been demonstrated in real SDN settings. Accordingly, various efforts have also looked at improving the security of SDN operation. Of particular concern is large scale DoS flooding attacks, which can be used to overwhelm both the data and control planes in SDN. For example, SLICOTS [5], OPERETTA [6], and the solution presented in [7] try to thwart transmission control protocol (TCP) SYN flooding attacks by surveilling failed SYN requests. However, there is a further need to develop more effective DoS security solutions due to network traffic dynamics. In particular, these methods should leverage the real-time traffic monitoring capabilities of SDN controllers to implement more dynamic schemes for DoS/flooding detection and mitigation.

On the other hand, it has been demonstrated that the centralised SDN controller itself, as well as SDN-enabled devices, are a potential target of DoS attacks, such as SYN flooding attacks. Indeed, an adversary might aim to exhaust the controller's resource, consume control plane and/or infrastructure plane bandwidth, or overload flow tables in switches [8]. In Ref. [2], Hong et al. highlighted security threats in SDN, including DoS, and studied its relevant...
attack scenarios in various SDN controller software. However, grand challenges concerning the detection and prevention of DoS attacks remain unsolved. Existing intrusion detection and prevention systems (IDPS) solutions to prevent such attacks introduce serious challenges, including system performance [9], network communication constraints [10], and response time [3]. Existing IDPS detection mechanisms are based on a static threshold, which needs to be manually set by a network administrator before launching the software. Moreover, in such traditional IDPS solutions, detection modules might fall short in detecting unknown or new network attacks due to constrained attack signatures and comparison mechanisms.

Now, of particular interest is the TCP SYN flooding attack. This attack model has been presented and proven a serious threat in SDN networks by Ziegler [11]. Adversaries here can generate a large number of SYN packets to saturate targeted resource (e.g. server), which leads to availability and QoS degradation. This type of attacks is based upon the TCP 3-way handshake, where the handshaking process does not complete the establishment between source and target nodes (i.e. half-open connection). Unlike normal TCP connection, the adversary generates a large number of half-open TCP connections without sending ACK packets/messages to the target node. This scenario leads to the target’s resource exhaustion as well as the throughput of both data and control planes. Note that this attack can be achieved by sending SYN packets at a very high rate, which will strangle the processing and throughput capacities at both the OpenFlow switch and controller levels.

Even though many studies tried to address this security concern in SDN infrastructures, they are still unable to identify threats, where an adversary completes the TCP handshake without communicating any further packets and/or present significant delays in legitimate TCP traffic processing due to the static detection thresholds (in these solutions). Here, it is essential to note that previous mitigation solutions based upon static detection thresholds fall short of providing optimal in-structure and mitigation times, where adversary flooding traffic is based upon a dynamic pattern nature. Hence, such variable pattern-based DoS attacks will require the elaboration of attack vector signatures based upon the changing traffic parameters and patterns over time.

We propose an IDPS solution named SYNGuard, a lightweight modular kernel-level filtering solution inspired by Amann et al. [12]. Our framework is introduced to identify and prevent SYN flooding threats in an SDN environment. Notably, our framework is self-threshold-adjusting and runs as an independent test access point (TAP) at the application layer level using switch port analyser (SPAN) on forwarding devices. Furthermore, the Bloom filter [13] and Aho-Corasick filter [14] are deployed in our solution as partial string matching and exact string matching techniques, respectively. By leveraging the strengths of these two approaches and SDN capabilities, our solution allows for an efficient and flexible interchange between these two techniques of string matching in addition to threat signature database update to guarantee efficacious detection and mitigation of threats within the kernel space in real time.

While our IDPS signature methods are used to compare a list of given strings or a set of rules with incoming network traffic signatures, we propose a novel approach based on the additive-increase/multiplicative-decrease (AIMD) algorithm to dynamically calculate and update the detection threshold to capture SYN flooding attacks. The adapted additive increase-multiplicative decrease (AIMD) approach is a feedback control approach that merges both the linear growth of the attack window and a reduction when attacks are identified. To demonstrate the performance efficacy of SYNGuard, we experimentally compare it with Zeek [15] and Snort [16], two well-known IDPS solutions through the global environment for network innovations (GENI) [17], a real-world heterogeneous and federated testbed. Our framework is implemented in Floodlight [18], an open-source SDN controller software, as a representational state transfer (REST) application programme interface (API). In order for one to thwart this attack, two key research challenges should be considered. (1) How to maintain the main operability of the SDN environment functioning while being under SYN flooding attacks? and (2) how to manage the malicious SYN traffic when mixed with legitimate user networking traffic?

To address these challenges while enforcing security policies of the SDN controller during a data-to-control layer SYN attack, our presented mechanism places contingent policy rules in the corresponding OpenFlow switches in a proactive manner. The proactive flow rules refer to low-level data plane flow rules, which are created by any application based on the SDN controller state [9,19], and consist of forwarding rules of all incoming traffic. Moreover, the insertion of traffic rules in a proactive way will help maintain the main operability of the SDN architecture.

Now note that the proactive installation of traffic rules does not guarantee complete protection of the centralised SDN controller; for example during SYN saturation attack against the data-to-control layer, the controller might be susceptible to the overloading issue. This is plausible due to saturation traffic that can remain outside the SDN-enabled applications logic if it does not match the existing proactive forwarding rules (i.e. in this case, the traffic will be forwarded back; the forwarding may lead to SDN controller’s overloading problem). Herein, a possible solution could be that the controller rejects such traffic to protect the controller, but this will sacrifice considerable legitimate user traffic. Furthermore, in our mechanism, we incorporate the cache of the data layer through the migration module of the controller with table – miss packets to cache packets for a fixed time frame prior to sending them to the SDN controller in a rate restricted mode. In this way, the data layer forwarding devices (e.g. such as Open vSwitch) are significantly protected from SYN flooding packets, and data-to-control layer saturations are efficiently detected.

The main contributions are summarised as follow:

- We first examine the behaviours and impact of the SYN saturation against the data-to-control layer and construe its effect on the OpenFlow-enabled environment.
• We propose a novel approach based on the additive increase-multiplicative decrease (AIMD) algorithm to dynamically calculate and update the threat detection threshold in real time to efficiently capture malicious SYN flooding traffic.

• As dynamic SYN flooding is proven a serious security threat in SDN environments, we develop SYNGuard, a lightweight kernel space security framework to identify and mitigate the SYN flood type of DoS attacks. SYNGuard is a lightweight and protocol-independent defence solution to thwart data-to-control layer SYN attack in SDN networks. Most notably, SYNGuard is a scalable framework and can adapt to larger topological scales and Science DMZ technology growth.

• We implement the SYNGuard solution through an open-source SDN controller software and test it over GENI, a real-world heterogeneous and federate testbed. In addition, we test the solution in various attack scenarios in a software OpenFlow switch (Open vSwitch) platform. Our experimental findings demonstrate SYNGuard efficiency in offering a scalable security mechanism for SDN-enabled environments against SYN flooding threats.

The rest is organised as follows. Section 3 presents our research background. Section 2 provides related work and Section 4 describes the approach and design of our solution. In Section 5, we present our evaluation findings. Finally, we give concluding remarks of our research in Section 6.

2 RELATED WORK

In the past, SDN has been widely deployed to enhance network efficiency [20] and security [21, 22]. However, recent studies have revealed various vulnerabilities of existing security approaches in SDN environments [23–25], and thus, various studies have been conducted to improve the SDN resiliency. In particular, Mahout [26] proposed a mechanism for flooding threats prevention in SDN. However, the solution is based on statistic aggregation, which renders it impractical as data-to-control layer saturation attacks may exploit micro-flows. KernelDetect [21,27] suggested a lightweight kernel space-based IDPs using modular string searching and filtering approaches to address DoS threats. AVANT-GUARD [28] presented a solution to alleviate the impact of saturation attacks using flow alteration management in OF devices. However, its design [28] is based on a SYN proxy implementation and limited to TCP saturation attacks, where only flows that accomplish TCP handshake are exposed.

In Ref. [29], Tian et al. presented FlowSec and Blackbox, two approaches to counter the DoS threat by enforcing the number of packets sent to controller per second and keeping track of attack levels while mitigating it. While PrioGuard [30] is a DoS mitigation scheme using a non-cooperative repeated game, FloodDefender [31] is a protocol-independent solution for DoS mitigation. FloodDefender aims at securing both the control and data layers through leveraging table-miss engineering and packet filter approaches. FMD [32] is another solution to prevent re-forwarded request-based flooding attacks in a multi-SDN environment. FMD deploys an adaptive rate adjustment method to adjust the re-forwarding rate.

In addition, MinDoS [33] prevents DoS attacks using a priority manager, where the flows are forwarded to multiple buffer queues of varying priority levels to enhance controller security. In Ref. [34], Bharathi et al. proposed path randomisation and flow aggregation-based solution to solve DoS impact on switch flow tables. Furthermore, SDNManager is a lightweight framework for DoS prevention in SDN based on flow bandwidth change forecasts. Lastly, SECO [35] and SECOD [36] are secure controller mechanisms to prevent DoS attack against both the control plane and data plane of SDN.

Furthermore, SYN flooding traffic can be identified by assessing the ratio of SYN packets number to ACK flows produced by the same entities [37]. However, it is impractical to elaborate this mechanism since malicious packets identification from a large traffic pool, and efficacious marking of threatening flows in runtime are challenging tasks to achieve. In Ref. [8], Deng et al. conducted a study on how to solve flooding threats and improve network resiliency by leveraging the SDN capabilities and infrastructure characteristics. However, the current detection and mitigation knowledge are incomplete yet with regard to sophisticated SYN flooding attacks (noting that the lever-aged SDN modules could be themselves vulnerable to malicious SYN behaviours).

To the best of our knowledge, most of these studies attempted to improve the internal interactions within modules in the application-controller-switch infrastructure to address external DoS resiliency threats in the SDN environment. However, none of them considered a dynamic/self-adapting detection threshold where the flooding traffic and burst may vary in real-time and change parameters and patterns over time.

3 RESEARCH BACKGROUND AND PROBLEM

In this following section, we provide a brief background discussion about SDN, we introduce the adversary and attack model of data-to-control layer saturation attack through SYN flooding, and lastly, we describe our research problem.

In an SDN-enabled network, switching devices (such as OVS platforms) process packet flows according to the rules injected by the SDN controller. Namely, these flow rules are generated and transmitted by the SDN controller to data plane switches in either a proactive or reactive manner. Specifically, in the proactive mode, the controller tries to populate rules prior to the arrival of user traffic at the forwarding device. Conversely, in the reactive mode, the controller injects and adjusts flow rules dynamically in real-time. Hence this study focuses on using reactive techniques to handle TCP SYN flooding threats against SDN controllers.
3.1 Overview of SYN flooding attacks

TCP SYN flooding attacks are DoS attacks that try to overwhelm a victim’s host computer with a large quantity of ICMP, SYN/SYN fragments, or UDP packet traffic. In particular, these attacks appear when a single or multiple hosts are flooded by TCP SYN segments initiating unaccomplished TCP connection requests (and hence are unable to respond to legitimate connection requests) [38]. Specifically, TCP client and server hosts use a three-way handshake mechanism to establish a connection, as shown in Figure 1. Successful setup requires the client to send a SYN packet to the server host, which in turn replies with a SYN_ACK packet and immediately allocates TCP stack resources for this established connection query. Once this is achieved, the server enters a listening state, called transmission control block (TCB) [5], and stays there until it receives a final ACK message from the client (within a given timeout interval). It is only once the client replies with an ACK message that the TCP session is deemed as successfully established, and data can be exchanged.

Now, as per the above, TCP session establishment is very susceptible to SYN flooding attacks, that is where an adversary sends a large number of half-open connections to the server without sending closing ACK packets, also shown in Figure 1. These floods can lead to resource exhaust at the server, preventing legitimate new connections from being established. Moreover, in some cases, many distributed adversarial hosts can even send a large number of SYN packets using spoofed IP addresses (i.e. DDoS attacks), making it even more difficult for a server to identify the sources of an attack.

3.2 Threat and attack model in SDN

The threat model here assumes both data and control plane vulnerability to TCP SYN flooding attacks. Furthermore, it is also assumed that (1) an attacker is capable of sending a large amount of TCP SYN-based traffic to an OpenFlow network, and (2) adversarial traffic comprised a mixture of both legitimate and malicious packets, making it difficult to differentiate. Accordingly, Figure 2 shows a realistic scenario that illustrates how an attacker can flood the SDN infrastructure with TCP SYN packets. Here, the OVS device receives an incoming packet stream containing a mixture of legitimate (normal user traffic) packets and malicious SYN flooding packets (from the adversary host). It is assumed that the single adversary host is capable of producing a large number of SYN packets to saturate the switching device.

Now consider the detailed interactions between the end hosts, switches, and SDN controller shown in Figure 2. Foremost, the adversary Host A (host) sends SYN flooding packets (step a), and the receiving switch checks its flow table for a matching forwarding rule. If no flow rule matching the destination IP address is found, that is, table-miss, the switch forwards it to the SDN controller (step b) in the form of Packet_In message. The SDN controller processes this Packet_In request and searches for the corresponding path to the target host (assuming that there is at least one path available). It then inserts the flow rule into the switch’s flow table (step c). Next, the switch forwards the SYN packet to the target host, Host B (step d). Finally, when Host B receives the SYN packet, it allocates a new buffer for this request and replies with a SYN_ACK packet (step e). Similarly, once the destination switch receives the SYN_ACK message, it again forwards it to the SDN controller for flow rule query (step f). Upon receiving this Packet_In message from the switch, the SDN controller once again computes the return route and inserts it in the switch flow table (step g). Finally, the switch sends a SYN_ACK message to Host B (step h). Given this handshake-based communication model, an adversary can readily insert false source IP addresses and make the SDN controller insert spurious new forwarding rules (step g). Namely, the forwarded SYN ACK packets to Host B will be based on invalid IP addresses.

In light of the above, if the adversary host transmits a large number of ‘spoofed’ SYN packets, it can easily flood the resources at the target host. More importantly, the adversary can also overwhelm switch lookup tables with spurious flow rules, consuming precious memory. Furthermore, SYN floods can impose a high computational workload at the SDN controller.
that is *Packet_In* processing, and also saturate bandwidth usage at the control plane due to excessive messaging that is *Packet_In* messages and flow rules.

### 3.3 Motivation and problem Scope

In this subsection, we discuss key challenges related to this research problem and motivate our need for an efficacious IDPS solution to thwart TCP SYN flooding threats against SDN infrastructures.

As discussed earlier, SDN environments can face many specialised threats from malicious adversaries. Hence the simple pre-placement of flow rules in data plane switches to dismiss any non-matching flows is not practical. Instead, SDN control implementations typically consist of different processing policies since network traffic is usually very dynamic. For example, policies can differ according to varying network conditions or user types [37]. Indeed, dynamic processing policies can render controller applications susceptible to security SYN flooding threats because such policies require frequent updates during the data layer transition [37]. Recent versions of the OpenFlow protocol also allow the SDN controller to receive information about data layer transition via incoming *Packet_In* messages. Hence when a SYN flooding attack occurs, the controller’s resources can easily be exhausted.

In light of the above, the first objective here is to maintain the core functionality of the SDN infrastructure during SYN flooding saturation attacks. In particular, this is done via the direct installation of flow rules in SDN switching devices in real time. Furthermore, another objective is to process table-miss flows without harming normal/legitimate user traffic flows. In particular, simply dropping all table-miss traffic is not acceptable as it may adversely impact legitimate users as well. Instead, whenever a SYN flooding attack takes place, the table-miss traffic will be sent to the corresponding data layer cache prior to communication with the SDN control layer (via *Packet_In* messages).

### 4 MODELLING AND FRAMEWORK DESIGN

To address SYN flooding attacks in SDN settings, we present SYNGuard, a novel protocol-independent IDPS solution. The proposed scheme runs as an application at the SDN controller and comprises several modules, as shown in Figure 3. As noted earlier, SYNGuard makes exclusive use of SDN capabilities to monitor and collect statistics from incoming data plane traffic and execute flow routing rules/responses. The overall placement of this solution (in relation to the Floodlight controller and OVS platform) is also shown in Figure 4. Namely, this scheme uses OVS switched port analyser (SPAN) ports to mirror all incoming flows on the corresponding device (for analysis). Precisely, SPAN ports support communication traffic between the application and data layers of the SDN setup. Overall, this positioning is critical for effective identification and threat mitigation of SYN flood traffic, as shown in Figure 4. Now the architectural system in Figure 3 consists of four key modules; (1) **Initialisation**, which sets up initial operation of the switches, (2) **Inspection**, which examines the traffic received through the raw operating system socket (e.g. SPAN port); (3)
Mitigation, which tries to limit false positive detection events; and (4) Threshold Evaluation, which dynamically adjusts the attack detection threshold according to the current record of attacks (based upon the SDN controller view).

Furthermore, to determine whether incoming traffic constitutes a TCP SYN flood attack, we identify the pattern of incoming flows over a time frame. Namely, when the current threshold value is surpassed by the quantity of arriving flows, the inspection module in the framework detects/flags a SYN flood attack and generates an alert message. Contrary to existing IDPS solutions, the dynamic threshold mechanism in the proposed framework also relies on the state of the inspection module and the frequency of generated alerts.

Overall, SYNGuard (1) is a lightweight design that uses a modular-based filtering technique, (2) runs as an independent network application that utilizes SPAN interfaces on OVS platforms, (3) adds minimal overhead and traffic processing latency, and (4) operates transparently to end hosts and SDN-enabled applications. The latter ensures improved applicability to larger networking environments. Further details are presented as follows.

4.1 Adaptive detection threshold and signature structure

In the context of DoS/DDoS attack detection, threshold values generally refer to the rate of attack events per second. Specifically, a detection threshold represents the rate at which an IDPS raises an alert for an attack. For example, if the event threshold is set to 70 attack packets per second, an alert will be generated once the incoming packets from a source exceed this bound. Hence, selecting an appropriate detection threshold value is critical to provide an efficacious response to DoS threats while reducing false alerts (i.e., threshold value too high or too low). To address these challenges, a dynamic self-adjusting detection threshold is proposed for SYNGuard. Consider some further details here.

Foremost, it is assumed that legitimate user flows generate TCP SYN requests at steady intervals. Namely, the number of SYN connection requests range from \([r_1, r_2]\), whereas the number of flows per SYN connection is \(n \geq 3\) (i.e., three-way handshake exchange). Now to compute the number of packets per SYN connection, \(n\), the SDN controller sends a flow statistics collection (FSC) request via an OpenFlow Read State message to switches with the matched field set equal to the IP addresses in the flow table. The value of \(n\) can then be estimated using the OpenFlow Packet Count parameter, which is a pre-defined metric that tracks the number of incoming flows based upon the switch flow table that is

\[
n = \min_{pc_i} (pc_i)
\]

(1)

where \(F\) and \(pc_i\) are the set of flows in the flow tables and the Packet Count value of flow \(i\), respectively.

Now consider the dynamic threshold update mechanism for SYN flood detection. Here, the ubiquitous TCP additive increase multiplicative decrease (AIMD) window size control mechanism is adopted. Namely, let \(\Psi_i\) and \(\Psi_D\) represent the updating functions for threshold increase and decrease, respectively. In a default TCP AIMD mechanism,

\[
\Psi_i = S + 1,
\]

(2)

and

\[
\Psi_D = \frac{S}{2},
\]

(3)

where \(S\) is the threshold value bounded by \(S_{\min} \leq S \leq S_{\max}\) that is \(S_{\min}\) and \(S_{\max}\) represent the minimum and maximum threshold ranges, respectively (e.g., in our experiments, they are initially set at 15 and 70, respectively, before launching the system). Leveraging from the above, the SYNGuard solution introduces a modified, improved AIMD mechanism that dynamically adjusts the increase and decrease levels based upon the event state of the SYN flood threat. Namely, now the increase and decrease operations (for the \(i\)-th interval) are given by:

\[
\Psi_i = S_i + \alpha,
\]

(4)

and

\[
\Psi_D = \frac{S_i}{\beta},
\]

(5)
1: Set $P = PACKET\_IN$
2: Set $Th = threshold\_value$
3: while True do
4:   $T = Time\_frame[Timestamps]$; $S_0 = P/T$
5:   if $P.Type == ICMP$ then
6:      Calculate $n/T$
7:      $QP.\_SRC\_ADDR$ + +
8:   if $P.\_SRC\_ADDR$ NOT IN $S$ then
9:      $S(P.\_SRC\_ADDR) = T$
10: else
11:    if $T - S(P.\_SRC\_ADDR) > Th$ AND $Si < Smax$ then
12:       if $Q(P.\_SRC\_ADDR) > Th$ then
13:          Increase $Th$
14:       else
15:          $S(P.\_SRC\_ADDR) = T$
16:     else
17:        Decrease $Th$ if $Si > S\_min$
18: return 0

FIGURE 5 The proposed algorithm for the detection threshold update

where $S_{\_min} \leq S_i \leq S_{\_max}$ and $\alpha$ and $\beta$ represent the threshold increase and decrease values, respectively. Furthermore, $S_i$ is manually initialised at a default threshold value at startup.

As for further improvement, a single update function is also defined for the detection threshold. This function is continuously invoked by SYNGuard to update the current threshold according to both legitimate and total traffic rates. Namely, the two update functions, $\Psi_I$ (Equation 4) and $\Psi_D$ (Equation 5), are now replaced by a next-round threshold value, $S(i + 1)$, such that:

$$S(i + 1) = p(\frac{n}{T} \geq S(i)) \cdot q(i) \cdot \Psi_I(S(i)) + (1 - q(i)) \cdot \Psi_D(S(i))$$

(6)

$$S(i + 1) = q(i) \cdot \Psi_I(S(i)) + (1 - q(i)) \cdot \Psi_D(S(i))$$

(7)

where $T$ denotes the window value. Next, simplifying Equation 6 gives a threshold update function as follows:

$$S(i + 1) = q(i) \cdot \Psi_I(S(i)) + (1 - q(i)) \cdot \Psi_D(S(i))$$

(7)

where $S(i + 1)$ represents the threshold value and it is thus an integer. For notation simplicity, $q(i) \cdot \Psi_I(S(i)) + (1 - q(i)) \cdot \Psi_D(S(i))$ in Equation 7 is considered an integer (or its flooring value otherwise), and $p(\frac{n}{T} \geq S(i))$ and $p(\frac{n}{T} \leq S(i))$ are the probabilities that both events $\frac{n}{T} \geq S(i)$ and $\frac{n}{T} \leq S(i)$ occur, respectively.

Now carefully note that packet signature matching may not always be the best strategy to identify SYN flood attacks; that is since malicious hosts may be capable of injecting any arbitrary data bytes into packet payloads. As a result, signature matching can actually be defeated by an adept adversary. Moreover, certain types of TCP SYN flooding attacks may not contain proper packet payload data for example low-profile SYN flood attacks. Hence, the proposed framework also tries to match packet header information (instead of packet payload data) to improve anomaly detection performance. Specifically, this approach can help increase the effectiveness of threat signature matching and detection and can also handle low-profile SYN attacks (with no particular data format in the packet payloads).

The overall pseudo-code of traffic inspection is shown in Figure 5, where the header information of the packet is checked using string matching techniques. Here, the variables $T, Th$, and $S_i$ represent the timestamp, threshold value, and list of strings (i.e., for string matching purposes), respectively. Besides, $S(\cdot)$ is defined as a counter function to track the number of packets per source address (for threshold comparison). Furthermore, if an ICMP packet is received, the packet counter, $S(\cdot)$, is updated, and its header information is compared with the existing signatures that is string search. Hence, an alert is generated based upon the current threshold and string matching result, that is, anomalous packet load. Lastly, the threshold update function is invoked for appropriate adjustment of the current attack detection rate.

4.2 Framework placement and functionality

Now, once SYNGuard has been initialised, the inspection module (Figure 6) monitors and classifies all TCP SYN requests that is established TCP handshakes, while recording source-side connection timeouts that is failed connections. Once a connection request (i.e. first SYN packet) is received at the data layer, the OVS searches for a matching TCP ACK forwarding rule in its flow table or inquires about it from the SDN controller. Carefully note that the OVS device also verifies whether the received packet type is a TCP SYN (prior to validating the completion of the TCP handshaking session) to identify unsolicited packets.
The overall finite-state machine for the SYNGuard system is also shown in Figure 6. Here, the initial state specifies all the input and global variables used in the inspection and mitigation modules, and jumps start the packet inspection process. Once a malicious TCP SYN event is detected in the Inspection state, a string matching operation is carried out, followed by alert generation. Next, if the current threshold value of detected events is exceeded, the system moves to the prevention state to mitigate the attack. Lastly, the system also invokes the threshold update function to update the current detection threshold.

Finally, as shown in the architecture in Figure 3, once a flow is received at the respective OVS port, the switch will transition from the idle state to the inspection state that is to start traffic inspection prior to forwarding it to the potential target. It is expected that the network administrator will specify/enter a default threshold value at startup as well. Finally, a secure mode option is also supported to allow the IDPS solution to conduct more comprehensive deep packet (payload) inspection using exact string matching that is in the case of sensitive data. However, if the switch is located in a trusted region, the IDPS will only perform partial string matching.

Overall, switches will continuously inspect and examine all incoming flows and keep track of malignant adversary traffic. Based upon this, threat detection thresholds will be updated in a dynamic manner (as per the current trace and intensity of illegitimate traffic). For example, if a sequence of flows is found to be legitimate, the switch will decrease the threshold $S_i$ by a value $\beta$ and also select exact string matching filters (or partial ones). Alternatively, if a particular flow is found to be malicious, the controller will forward it to the prevention module (Figure 3) to either drop or block it.

5 | EVALUATION

The performance of the proposed SYNGuard solution is now tested in a real-world heterogeneous testbed setup with various networking capabilities. Again the NSF GENI facility is used here, and the proposed topology slice is shown in Figure 7. Namely, this setup consists of two end hosts, host and Host B, SDN controllers, and four OVSes. In particular, host represents the adversary, which transmits a mixture of legitimate and malicious traffic to the target at Host (b) Meanwhile, hosts O1-O4 represent OVSes running the IDPS solution (in addition to performing their essential traffic forwarding operations). Finally, hosts C1 and C2 represent the SDN controllers running the Floodlight controller software. Namely, controller C1 manages OVSes O1 and O2, and controller C2 manages OVSes O3 and O4, respectively. Lastly, the dashed and solid lines represent the control and data plane links, respectively. Although a relatively simple topology is used to evaluate the performance of SYNGuard scheme here, the solution can readily be used in larger and more complex networking environments.

5.1 | Traffic generation and event rules implementation

Initial tests are done to evaluate system response time to threats under simulated real-world networking conditions. Namely, two network traffic intent types are considered here that is adverse (malicious) and normal (benign). Packets from these flows are then generated and mixed into the network. Furthermore, both the iPerf and the distributed Internet traffic generator (D-ITG) toolkits are used in tandem to generate packet traffic and quantify/control it across the network. Note that iPerf also allows the saturation of edge network links, which can be used as a baseline technique to evaluate response efficiency to saturation attacks.

In addition, both the Zeek and Snort IDS toolkits are also deployed here for rule-based detection. Namely, pre-defined detection rules are manually added or configured that is if they are already implemented in a particular version. However, to ensure proper performance comparison, the only rules used for Zeek and Snort are those for SYN flood attack detection and mitigation. Since both of these tools can handle a broader range of network threats, comparing their associated overheads with other rules enabled will not be meaningful (and hence they are offloaded or disabled). Accordingly, the sample scripts in Figures 8 and 9 also show the actual detection rules for SYN-based flood traffic in the Snort and Zeek IDS tools, respectively. Namely, in Figure 8, Snort raises a TCP-SYN flagged threat alert based upon a default threshold value, for example, whenever 70 or more packets arrive within a 10 s interval. Meanwhile, in Figure 9, Zeek uses a specified threshold, $n$, which can be manually adjusted during configuration setup. This parameter counts the number of failed TCP connections and raises an alert flag when this count is exceeded.

5.2 | Performance results

Overall, effective IDPS solutions play a critical role in countering malicious attack traffic. As noted, adverse packet flows can result in the end host resource exhaustion, link and switch
saturation, as well as other bottlenecks. Therefore, proper flow inspection is critical for limiting resource exhaust. Along these lines, the performance of the SYNGuard solution is examined with regards to the inspection times, threat response times (mitigation), threat identification accuracy, and performance overheads. Furthermore, comparisons are also made with some existing well-established solutions, including Snort and Zeek.

5.2.1 | Inspection time

Foremost, inspection time for incoming flows is a critical parameter that directly impacts malicious flow response times. Namely, when flows are captured by the IDPS, the related packet information must be inserted into a buffer while awaiting inspection. Expectedly, this waiting time will add to the overall time required for mitigating the attack if an alert is raised. Hence, to evaluate the inspection times of the proposed solution, both the Snort and Zeek solutions are implemented with near equivalent configurations as used in the SYNGuard scheme. Specifically, for evaluation fairness, each IDPS solution is configured with similar threat detection signatures.

To gauge inspection times, a communication scenario is established between two hosts using the hping3 tool. Specifically, smaller-sized packets are generated at a high rate in a very small time window (e.g., 1 ns) to achieve a high network link saturation. Note that the proposed SYNGuard solution automatically adjusts its detection threshold, whereas this value is static in both the Zeek and Snort schemes (and hence needs to be manually set at initialisation time). Hence these static detection thresholds are set to default values of 10 s each. Experimental runs are also conducted for 15 successive trials in an automated manner, and the results are averaged, as shown in Figure 10 and Table 1. Overall, these findings demonstrate that the proposed solution notably outperforms both Snort and Zeek. For example, Zeek gives almost 100% higher inspection times, whereas Snort gives about 10–20% higher latencies. However, for some randomized runs, Snort can match (but not exceed) the performance of the SYNGuard scheme.

5.2.2 Mitigation and response time

Next, consider mitigation time, which is defined as the response time to handle threatening adversary flows. Namely, once a specific flow has been detected and flagged as malicious, the mitigation time is defined as the time between raising an attack alert and taking necessary corresponding action for example drop, block, etc. Indeed, this delay is a critical factor in assuring the availability of key operational resources in SDN environments. Now even though the detection attributes (for malicious traffic) in each IDPS scheme are unique, the mitigation time for the SYNGuard scheme is still compared against Snort and Zeek. Precisely, this value is measured as the time between attack initiation and attack rectification.
First, it is assumed that each IDPS scheme initiates blocking actions against harmful IP addresses once an attack alert is generated. Table 1 summarises the averages of mitigation times, inspection times, and system resource utilization for the three IDPS solutions using the same experimental configuration described earlier that is threshold initialisation, threat detection signature, etc. Note that these experimental results in Table 1 are averaged over 15 runs. As shown in Table 1, SYNGuard outperforms Snort and Zeek for the cases of 100 and 500K SYN flagged packets in terms of all the three performance metrics, inspection time, mitigation time, and the percentage of system load. In the case of 200K SYN flagged packets, SYNGuard still performs better than Zeek in all the three performance metrics. However, while SYNGuard has a lower percentage of system load than Snort, it has slightly higher inspection time and mitigation time compared to Snort. Moreover, Figure 11 further depicts the attack mitigation times for all schemes, where the SYNGuard solution demonstrated superior performance compared with Zeek and Snort systems.

5.2.3 | Overhead evaluation

Now excessive inspection operations can also lead to resource drainage on the platforms running the IDPS software (due to overheads associated with examining packets). Hence to gauge system overheads, Figure 12 also plots the memory utilisation for each IDS scheme for the case of a single attack with 100,000 SYN flagged packets. Here, both the Snort and Zeek solutions are initialised to use their default detection threshold (10 s). Carefully note that link saturation also occurs in this attack scenario. These results clearly show lower memory utilization with the proposed SYNGuard scheme, particularly in comparison with the Zeek solution. Note that Figure 1 also summarises the measured system load in terms of average resource utilization for a variety of network flooding attacks with 100,000, 200,000, and 500,000 SYN flagged attack packets (from adversary host to target Host B, Figure 7).

Finally, Figure 13 plots the CPU utilisation overhead for SYNGuard, Snort, Zeek, normal SDN (unprotected with no IDS), SLICOTS [5], and OPERETTA [6]. In particular, SLICOTS and OPERETTA are state-of-art IDS solutions (developed by Mohammadi et al. [5] and S. Fiebhera et al. [6], respectively) to prevent SYN flooding attacks in SDN networks. These schemes work by surveilling failed TCP SYN requests and blocking malicious adversary hosts. Specifically, SLICOTS is implemented as an extension of the OpenDayLight controller software, whereas OPERETTA is developed as an SDN framework using the POX controller. Overall, the results in Figure 13 show that the SYNGuard solution gives the lowest CPU utilization of all that is many factors lower than the Zeek toolkit. However, the Snort solution performs

| Metric          | IDPS     | 100   | 200   | 500   |
|-----------------|----------|-------|-------|-------|
| Inspection (sec.)| SYNGuard | 0.0031| 0.0068| 0.0117|
|                 | Snort    | 0.0041| 0.0059| 0.0145|
|                 | Zeek     | 0.0073| 0.0102| 0.0130|
| Mitigation (sec.)| SYNGuard | 0.0029| 0.0081| 0.0112|
|                 | Snort    | 0.0045| 0.0073| 0.0130|
|                 | Zeek     | 0.0076| 0.0100| 0.0128|
| System (%)      | SYNGuard | 8.04  | 12.70 | 15.09 |
|                 | Snort    | 9.02  | 15.08 | 19.17 |
|                 | Zeek     | 17.53 | 26.05 | 34.19 |

**TABLE 1** Average inspection and mitigation times as well as system load

**FIGURE 11** SYN flood mitigation time (100K flagged packets and 10 s threshold used in Zeek and Snort)

**FIGURE 12** Memory utilization for a single attack (100K SYN flagged packets and 10 s threshold used in Zeek and Snort)
and at-scale real-world testbed, to conduct a comprehensive and comparative analysis of our modular framework with popular IDPS solutions, Zeek and Snort. The performance evaluation demonstrated our solution outperforms the other two examined IDPS solutions in terms of accuracy and efficiency with regard to inspection time, mitigation time, and resource consumption. As we considered solving network threats within the infrastructure layer of SDN, in the future, we plan on identifying other serious DoS threats at the SDN layers, particularly spoofed packets-based and TCP reply-enabled DoS threats.

ACKNOWLEDGEMENTS
The authors acknowledge National Science Foundation to partially sponsor Dr. Kaig Xi Dong’s work under grants CNS 1633978, 1620871, 1620862, and 1620868, and BBN/GPO project 1936 through NSF/CNS grant. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of NSF.

REFERENCES
1. McKeown, N., et al.: Openflow: enabling innovation in campus networks. SIGCOMM CCR ACM. 38(2), 69–74 (2008)
2. Hong, S., et al.: Poisoning network visibility in software-defined networks: new attacks and countermeasures. Network and Distributed System Security Symposium. NDSS, San Diego, 8–11 February (2015)
3. Wang, R., Ju, Z., Ju, L.: An entropy-based distributed DoS detection mechanism in software-defined networking. In: International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), International Conference on Big Data Science and Engineering (BigDataSE), and International Symposium on Parallel and Distributed Processing with Applications (ISPA), vol. 1, pp. 310–317. IEEE, Washington DC (2015)
4. Nguyen, T.-H., Yoo, M.: Analysis of link discovery service attacks in SDN controller. In: International Conference on Information Networking (ICOIN), pp. 259–261. IEEE, Da Nang, 11-13 January (2017)
5. Mohammadi, R., et al.: An SDN-based lightweight countermeasure for tcp syn flooding attacks. Trans. Netw Service Manag. 14, 487–497 (2017)
6. Fichera, S., et al.: An openflow-based remedy to mitigate tcp synflood attacks against web servers. Comput. Netw. 92, 89–100 (2015)
7. Chin, T., et al.: An SDN-supported collaborative approach for DDOS flooding detection and containment. In: MILCOM 2015-2015 IEEE Military Communications Conference, pp. 659–664. IEEE, Tampa, 26-28 October (2015)
8. Deng, S., et al.: DoS vulnerabilities and mitigation strategies in software-defined networks. J. Netw. Comput. Appl. 125, 209–219 (2019)
9. Wang, H., Xu, L., Gu, G.: Flood Guard: a DoS attack prevention extension in software-defined networks. In: International Conference on Dependable Systems and Networks (DSN), pp. 239–250. IEEE, Rio de Janeiro, 22-25 June (2015)
10. Dhawan, M., et al.: SPHINX: Detecting security attacks in software-defined networks, Network and Distributed System Security Symposium. NDSS, vol. 15, pp. 8–11, San Diego, 8–11 February (2015)
11. Ziegler, B.: Hacker tangles panix web site. Wall St. J. 12 (1996)
12. Amann, J., Sommer, R.: Providing dynamic control to passive network security monitoring. In: Research in Attacks, Intrusions, and Defenses, pp. 133–152. Springer (2015)
13. Dharmapuriak, S., Lockwood, J.W., Fast and scalable pattern matching for network intrusion detection systems. J. Select. Areas Commun. 24, 1781–1792 (2006)
14. Aicherry, M., Mathurprasanna, M., Kumar, V.: High speed pattern matching for network ids/ips. In: The International Conference on Network Protocols (ICNP), pp. 187–196. IEEE (2006)
15. Paxson, V.: Zeek: the Zeek network security monitor. Comput. Netw. 31(23-24), 2435–2463 (1999). https://www.zeek.org/
16. Roesch, M., et al.: Snort-lightweight intrusion detection for networks. USENIX Large Installation System Administration Conference (LISA) vol. 99(1), pp. 229–238. Seattle, 7-12 November (1999)
17. Berman, M., et al.: GENi: A federated testbed for innovative network experiments. Comput. Netw. 61, 5–23 (2014)
18. Floodlight OpenFlow controller. http://www.projectfloodlight.org/
19. Dobrescu, M., Argyraki, K., Software dataplane verification. USENIX Sympos. Netw. Syst. Design Implem (NSDI), 101–114 (2014)
20. Rahouti, M., et al.: Latencysmasher: A software-defined networking-based framework for end-to-end latency optimization. In: IEEE 46th Conference on Local Computer Networks (LCN), pp. 202–209. IEEE (2019)
21. Chin, T., Xiong, K., Rahouti, M.: SDN-based kernel modular countermeasure for intrusion detection. In: The International Conference on Security and Privacy in Communication Networks (SecureComm), pp. 270–290. Springer, Niagara Falls, 22-25 October (2017)
22. Chin, T., et al.: A machine learning framework for studying domain generation algorithm (DGA)-based malware. 14th International Conference on Security and Privacy in Communication Networks (SecureComm), Singapore, 8-10 August (2018)
23. Scott-Hayward, S., Natarajan, S., Sezer, S.: A survey of security in software-defined networks. Commun. Surv. Tutor. 18, 623–654 (2016)
24. Li, Y., et al.: A software-defined networking-based detection and mitigation approach against KRACK. 16th International Joint Conference on e-Business and Telecommunications (ICETE) and the 16th International Conference on Security and Cryptography (SECRYPT), Prague, 26–28 July (2019)
25. Chin, T., et al.: A phishing detection and mitigation approach using software-defined networking. IEEE Access. 42, 516–42531 (2018)
26. Curtis, A.R., Kim, W., Yalagandula, P., Mahour: low-overhead datacenter traffic management using end-host-based elephant detection. In: International Conference on Computer Communications (INFOCOM), pp. 1629–1637. IEEE, Shanghai, 10-15 April (2011)
27. Chin, T., Xiong, K., Rahouti, M.: Kernel-space intrusion detection using software-defined networking. Endors. Trans. Secur. Safety. 5(15), 10 (2018)
28. Shin, S., et al.: AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks. In: Conference on Computer and Communications Security (CCS), 4-8 pp. 413–424. ACM, Berlin, November 4-8 (2013)
29. Tian Y., Tran V., Kuerban M., DoS attack mitigation strategies on SDN controller. In: Annual Computing and Communication Workshop and Conference (CCWC), pp. 0701–0707. IEEE (2019)
30. Wu, G., Li, Z., Yao, L.: DoS mitigation mechanism based on non-cooperative repeated game for SDN. In: International Conference on Parallel and Distributed Systems (ICPADS), pp. 612–619. IEEE, Singapore, 11–13 December (2018)
31. Shang, G., et al.: Flooddefender: Protecting data and control plane resources under SDN-aimed DoS attacks. In: International Conference on Computer Communications (INFOCOM), pp. 1–9. IEEE (2017)
32. Wu, P., et al.: Fmd: A DoS mitigation scheme based on flow migration in software-defined networking. Int. J. Commun. Syst. 31, e3543 (2018)
33. Wang, T., Chen, H., Qi, C., Mindos: A priority-based SDN safe-guard architecture for DoS attacks. Inst. Electron. Inform. Commun. Eng. 101, 2458–2464 (2018)
34. Bharathi, N., Vetriselvi, V., Parthasarathi, R.: Mitigation of DoS in SDN using path randomisation. In: The International Conference on Computer Networks and Communication Technologies (ICCNCT), pp. 229–239. Springer, Coimbatore, 23-24 May (2019)
35. Wang, S., Chavez, K. G., Kandepan, S.: Seco: SDN secure controller algorithm for detecting and defending denial of service attacks. In: International Conference on Information and Communication Technology (ICoICT), pp. 1–6. IEEE, Malacca, 17-19 May (2017)
36. Wang, S., et al.: Secod: SDN secure control and data plane algorithm for detecting and defending against DoS attacks. In: Network Operations and Management Symposium (NOMS), pp. 1–5. IEEE, Taipei, 23-27 April (2018)
37. Zheng, J., et al.: Realtime DDOS defence using cots SDN switches via adaptive correlation analysis. IEEE Trans. Inf. Forensics Secur. 13(7), 1838–1853 (2018)
38. Kumar, P., et al.: Safety: Early detection and mitigation of TCP SYN flood utilising entropy in SDN. Trans. Netw. Service Manag. 15, 1545–1559 (2018)

How to cite this article: Rahouti M, Xiong K, Ghani N, Shaikh F. SYNGuard: Dynamic threshold-based SYN flood attack detection and mitigation in software-defined networks. IET Netw. 2021;10:76–87. https://doi.org/10.1049/ntw2.12009