Concurrency and Computation: Practice and Experience

WILEY

RESEARCH ARTICLE

ONOS Flood Defender: A Real-Time Flood Attacks Detection and Mitigation System in SDN Networks

Hussein Younis[1] | Mohammad M. N. Hamarsheh[2]

[1]Department of Cybersecurity, Faculty of Graduate Studies, Arab American University, Ramallah, Palestine | [2]Department of Computer Networks and Security, Faculty of Information Technology, Arab American University, Jenin, Palestine

Correspondence: Mohammad M. N. Hamarsheh (mohammad.hamarsheh@aaup.edu)

Received: 6 June 2024 | **Revised:** 10 November 2024 | **Accepted:** 29 December 2024

Funding: The authors received no specific funding for this work.

Keywords: distributed denial of service attack | flood attacks | intrusion detection system | ONOS controller | software-defined network

ABSTRACT

Cybercriminals are constantly developing new and sophisticated methods for exploiting network vulnerabilities. Software-defined networking (SDN) faces security challenges more than other traditional networks because the controller is a bottleneck device. This necessitates the implementation of robust security systems, including intrusion detection to mitigate the effect of attacks. Distributed denial of service (DDoS) attacks targeting the centralized controller of an SDN network can disrupt the entire network. If the controller becomes unavailable due to an attack, flow rules (FRs) cannot be deployed at the network switches, affecting data forwarding and network management. This study focuses on the detection and mitigation of synchronized (SYN) and normal transmission control protocol (TCP) DDoS flood attacks. It introduces two enhanced statistical detection and mitigation algorithms that work seamlessly with the open network operating system (ONOS) SDN controller, and sFlow-RT engine in real-time. Through a comprehensive set of experiments, our empirical findings demonstrate that the proposed algorithms efficiently detect and mitigate attacks with minimal average detection time, and negligible impact on resource consumption. By utilizing tuned threshold values based on network traffic volume, TCP flood attack detection (TFAD) algorithm and the synchronized TCP flood attack detection (STFAD) Algorithm achieved a minimal average detection time, of 4.032 and 3.430 s, respectively. These algorithms also have high detection accuracy in distinguishing normal traffic when appropriate threshold values are applied. Overall, this research significantly contributes to fortifying SDN networks with robust security measures, enhancing their resilience against evolving cyber threats.

1 | Introduction

Software-defined networking (SDN) involves separating the control and data planes of network intermediate devices. It enables centralized control of intermediate devices and efficiently monitor and manage the network [1, 2]. In SDN structure, a controller represents the control plane and connects with networking devices to ensure a proper network configuration. An application layer has an interface with the control plane and contains different network applications [3].

The application layer of SDN is where network applications and services are deployed, and includes network functions such as load balancers, firewalls, and intrusion detection systems (IDSs) [4]. The control layer manages the network and its resources and includes a centralized controller such as "OpenDaylight," "Ryu,"

and "ONOS" which communicates with the network devices through OpenFlow protocol to configure and manage the network [5]. The infrastructure in SDN consists of network devices such as switches and routers. These devices are responsible for forwarding data packets based on the flow rules received from the controller in the control layer, implementing network policies and rules, and providing network services, such as routing, switching, and security [6]. This approach provides the advantage of enhanced flexibility in managing the network. It also results in better scalability and cost-effectiveness [7]. SDN is becoming increasingly popular in enterprise networks, data centers, and cloud computing environments. However, the adoption of SDN introduces new security challenges that must be addressed to ensure the integrity and reliability of the network [8].

Cybercriminals are evolving new methods to exploit network vulnerabilities, leading to data breaches and financial losses. In Q2 2024, Cloudflare's distributed denial of service (DDoS) Threat Report noted a 20% increase in attacks. State-level actors used advanced techniques, with random DDoS attacks peaking at 16% in May. DNS attacks were common, with short-lived DDoS attacks under 500 Mbps [9]. SDN is vulnerable to several types of attacks, just like any other traditional network including DDoS [10, 11]. If the controller is targeted, the effect of these attacks on the SDN centralized control applications is much more than standard network. Routing and all other network applications, in case of DDoS attack will not be available because of a single point of failure for the controller [12, 13].

The TCP and TCP-Synchronized (TCP-SYN) Flood attacks are forms of DDoS attacks that take advantage of the standard TCP three-way handshake to exhaust resources on the victim's server and make it unresponsive [14]. During a TCP flood attack, the attacker sends a high volume of TCP packets without setting any TCP flags and with different sequence numbers for each request to the victim's server. In TCP-SYN flood attacks, the attacker sends a high volume of SYN packets after opening a TCP connection to the victim's server. In both attacks, the attackers continuously send packets without waiting for the ACK packets to be sent by the server. This results in the server waiting for the ACK packet that never arrives, which exhausts server resources and prevents it from responding to genuine requests. The deluge of incoming TCP packets can also consume network bandwidth and lead to network congestion, which further affects the server performance [15]. It is crucial to develop robust protection algorithms in SDN networks to safeguard them against distinct types of attacks that can potentially disrupt network services [16, 17]. These algorithms are implemented as software applications at the controller that enables rapid detection and response to security incidents, minimizes the impact of attacks, and ensures the continuity of network operations [18].

This study proposes statistical algorithms capable of detecting and mitigating two types of attacks in real-time. The main motivation behind these algorithms is to reduce the impact of the attacks and stop them as early as possible. Moreover, to our knowledge, this is one of the limited number of studies that have utilized the ONOS controller. We use statistical methods because these simple approaches are lightweight and computationally efficient, and that is crucial for fast real-time detection and mitigation in SDN networks. Furthermore, statistical methods can effectively capture some patterns and anomalies in network traffic without requiring extensive training data, making them well-suited for some types of attacks.

Utilizing statistical analysis, we developed a robust and efficient TCP flood detection system. The main contribution of this study is introducing and assessment of the operational efficiency of two statistical algorithms through empirical testing to determine the average detection time, and resource utilization under various attack scenarios. The algorithms proposed are TFAD and STFAD, which are implemented in real-time with the ONOS SDN controller and sFlow-RT engine.

The paper's key contribution is its innovative approach to fortifying SDN environments against flood attacks. It introduces two advanced statistical algorithms, TFAD and STFAD, tailored to integrate smoothly with the ONOS SDN controller and sFlow-RT engine in real time. These algorithms efficiently detect and mitigate attacks, achieving minimal average detection time and minimal impact on resource consumption. The study underlines the importance of implementing robust security measures in SDN networks to enhance their resilience against evolving cyber threats. By leveraging threshold-based algorithms, the ONOS SDN controller's flow-rules subsystem, and REST APIs for creating flow rules, the paper offers a comprehensive framework for safeguarding SDN environments against TCP flood attacks.

The rest of the paper is organized as follows. Section 2 reviews the related work that proposes techniques to defend SDN networks. Section 3 presents two proposed algorithms to detect and mitigate attacks on SDN networks. Section 4 discusses our SDN testbed including the experimental setup and the generation of normal and abnormal network traffic. Section 5 analyses the experimental results and the evaluation of the proposed detection and mitigation algorithms. Section 6 demonstrates the comparison of the proposed algorithms with previous work. The last section concludes the paper and summarizes the contributions and potential future directions.

2 | Related Work

Researchers have proposed diverse models to detect and mitigate DDoS on SDNs. The most prominent way to classify these proposed models is based on their characteristics, which include [19]:

1. The reliance on statistical approaches and tools that capture, monitor, and analyze whole or specific network flows to detect and mitigate DDoS attacks.
2. The reliance on machine learning models including the type of datasets and types or numbers of features that are used in training and testing these models to detect SDN attacks and the overall accuracy achieved by proposed models in detecting SDN attacks.

Over the past few years, researchers have made significant efforts to identify and prevent DDoS attacks by utilizing various architectures. However, these efforts have primarily focused on using POX, Floodlight, OpenDaylight, or RYU controllers, and there

TABLE 1 | A comparative analysis of the literature on detection and mitigation of SDN attacks.

References	Year of publication	Controller	Machine learning	Statistical	Application created	Detection	Mitigation
Chouikik et al. [20]	2024	Open Daylight	✗	✓	✗	✓	✓
Linhares et al. [21]	2023	Floodlight	✓	✓	✗	✓	✓
Aslam, Srivastava, and Gore [22]	2022	ONOS	✓	✗	✓	✓	✓
Girdler and Vassilakis [23]	2021	POX	✗	✓	✗	✓	✗
Oo et al. [24]	2020	ONOS	✗	✓	✗	✓	✓
Tuan et al. [11]	2020	POX	✓	✗	✗	✓	✓
Birkinshaw, Rouka, and Vassilakis [25]	2019	POX	✗	✓	✗	✓	✗
Our proposal	2024	ONOS	✗	✓	✓	✓	✓

are currently limited solutions for mitigating DDoS attacks using ONOS controllers. The general description of some studies is summarized in Table 1. Furthermore, some of the previous studies are limited to detecting attacks without providing an effective mitigation solution.

In [20], the researchers examine the impacts of DDoS attacks on SDNs and the effectiveness of IDSs in mitigating these risks. The proposed approach relies on the "SNORT IDS" which continuously examines network traffic in real-time comparing the contents of network packets against a predefined database of rules and signatures. These rules define the typical behavioral patterns linked to known attacks or malicious activities. By analyzing the network traffic against these rules, SNORT can detect and alert suspicious or potentially harmful behavior occurring on the network. The study utilizes the Open Daylight controller and the Mininet emulator to simulate a DDoS attack on an SDN network and evaluate the performance metrics, such as throughput and delay time, under attack scenarios with and without the presence of an IDS. The findings reveal that without the IDS, the network exhibits significant fluctuations in throughput, indicating heightened vulnerability to the DDoS attack. However, the study provides valuable insights into DDoS attack mitigation in SDN environments, there are significant areas that require further exploration and validation in practical scenarios.

Aslam, Srivastava, and Gore [22] proposes a solution called the ONOS flood defender application (OFD App). The app utilizes supervised and ensemble machine learning techniques to detect and mitigate DDoS attacks by tracing the attack traffic back to its source. The results show that ensemble machine learning techniques, such as Random Forest classifier (RFC) and XGBoost classifier, perform well in detecting DDoS attacks with high accuracy. The proposed framework of the OFD app is implemented in a Mininet emulator and ONOS SDN controller, demonstrating effective performance in terms of time, accuracy, and system overhead. The app aims to prevent severe damage to legitimate users by efficiently mitigating DDoS attacks.

Girdler and Vassilakis [23] employed a statistical approach to design an IDS that can identify ARP request spoofing, ARP reply spoofing, ARP reply to destination spoofing, and block-listed MAC (Media Access Control) addresses. However, the system can only handle ARP packets and follows predefined conditions regarding source and destination MAC address in Ethernet and ARP packets. The system was developed on top of a POX controller, using a virtual environment created by the "VirtualBox hypervisor" and "OpenVSwitch." They used the "Scapy" Python library for attack generation and "Wireshark" for network traffic collection. The findings of the study indicate that, on average, the system requires 02.20 s to identify a single instance of ARP spoofing attack and an additional 2.31 s to counteract it. This duration is significantly longer when compared with previous research in this field.

In [24], the researchers present a DDoS Mitigation application that utilizes a modified adaptive threshold algorithm (MATA) for real-time detection and mitigation of flooding attacks in SDN. Their approach involves monitoring network traffic, calculating dynamic thresholds based on baseline traffic, and applying flow rules to discard malicious packets while minimizing false alarms. The system uses sFlow for traffic monitoring and an ONOS controller application for mitigation. However, MATA provides an effective approach for dynamically detecting and mitigating various flooding attacks in SDN environments, with significant improvements in accuracy and reduced false alarms compared with previous methods. MATA requires an initial period to establish baseline traffic, which may introduce some overhead and delay before effective detection can occur. Also, the effectiveness of MATA relies on the accuracy of the baseline traffic information. If the baseline is incorrectly defined, it could lead to misclassifications of normal and attack traffic.

Tuan et al. [11] used a K-nearest-neighbor (KNN) machine learning classifier for DDoS attack detection and mitigation, including TCP/SYN and ICMP flooding. They trained and tested on two datasets—CAIDA 2007 and a generated dataset from the BoNeSi tool with five features. Their mitigation approach involved monitoring packets using the SDN POX controller, calculating entropy based on the IP address and port source, classification via KNN, and dropping attacker packets. KNN achieved 98% accuracy in mitigating DDoS attacks. Although the feature selection process involves many features, it also may impose severe restrictions for real-time operation.

Birkinshaw, Rouka, and Vassilakis [25] developed an intrusion detection and prevention system (IDPS) using threshold-based techniques such as Credit-Based Threshold Random Walk and Rate Limiting algorithms to detect DDoS attacks and the Port Bingo algorithm for detecting port scanning attacks in SDN networks. The IDPS is implemented on a POX controller monitoring all traffic in a virtual SDN network using Mininet and Wireshark. The detection techniques show a decrease in false positives by adjusting the threshold values with efficient detection of TCP, UDP, and ICMP DDoS attacks. The proposed IDPS only relies on techniques based on predefined thresholds. As a result, one potential issue arises when an attack intentionally manipulates the network traffic to stay just below the established threshold value. But in that case, DDoS attack will not be effective in preventing the service.

3 | The Proposed Algorithms

Our proposed detection and mitigation algorithms use a statistical approach to detect DDoS in SDN networks. They are threshold-based techniques that use the average time difference between successive packets to detect abnormal traffic. Network traffic is monitored and compared with a few pre-defined threshold values for different parameters only if the time condition is satisfied. Setting up the predefined threshold values depends on various network parameters such as bandwidth, packet loss, latency, CPU/memory utilization, and security events.

The network traffic is continuously monitored as illustrated in Figure 1. When an incoming TCP packet P arrives, packet counters are updated, and the packet is checked to ensure that it is not from a blocklisted group. The interarrival time (IAT) between the arriving packet and the last packet received from the same group stored in the database is then calculated using Equation (1):

$$IAT_n = T_n - T_{n-1} \tag{1}$$

where T_n, is the arrival timestamp of the nth packet, and T_{n-1}, is the arrival timestamp of the previous packet in the same group. If the IAT is within each algorithm's time window (T) for, the counters are incremented. Otherwise, the related counters are reset. These counters are then compared against predefined thresholds to detect abnormal behavior or a potential attack. If their values exceed the predefined thresholds, appropriate measures are taken to mitigate the threat by alerting network administrators and blocking suspicious traffic.

In TCP flood attack detection (TFAD) Algorithm, the threshold value is the maximum amount of TCP traffic flowing between the source and destination specific ports measured by bytes per time window (T). In synchronized TCP flood attack detection (STFAD) Algorithm, the threshold value is the maximum difference between [SYN] and [SYN, ACK] TCP packets during a TCP transfer between two hosts within the time window (T). However, threshold-based techniques have a problem if the attackers know the threshold values in advance and limit the attack traffic to be just less than the threshold value. The combination of the proposed detection algorithms sometimes provides a simple solution to solve this problem.

The time window (T) is determined to be slightly longer than the typical IAT which is the period between consecutive packets from the identical session in normal network traffic. Figure 2 illustrates two cases where the arrival rates of the traffic are different while the number of packets is the same. In each case, a host send nine packets to a server within 10 s, thus, it is impossible to differentiate between normal and abnormal traffic based solely on the number of sent packets [26] If the time window (T) is chosen to be 2 s for this system, and a threshold value of arrival rate (k) is three packets per T, then abnormal traffic can be distinguished. Two packets are transmitted every 2 s in the first case and consequently, for every T, the number of transmitted packets remains below the threshold (k) all the time. However, in abnormal traffic scenarios, five packets (packets 3, 4, 5, 6, and 7) are sent consecutively within a period less than T. As a result, the number of transmitted packets within T is five which exceeds the threshold value (k). This distinction allows detection algorithms to accurately identify and distinguish normal and abnormal traffic patterns.

The proposed detection algorithms are implemented in JavaScript language and embedded in sFlow-RT as an external application. The sFlow-RT continuously gathers information from sFlow agents embedded in network devices and converts the raw measurements into actionable metrics. These actionable metrics are accessible through the JavaScript API (Application Programming Interface) [27]. The JavaScript API of sFlow-RT allows to define custom flow events handler based on OpenFlow protocol attributes that define the characteristics of a specific

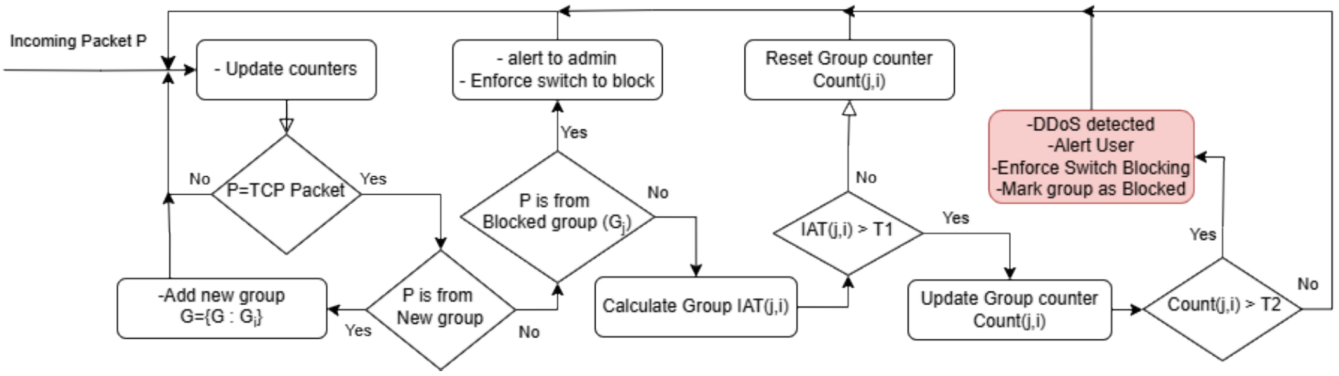

FIGURE 1 | The flow of proposed detection and mitigation algorithms.

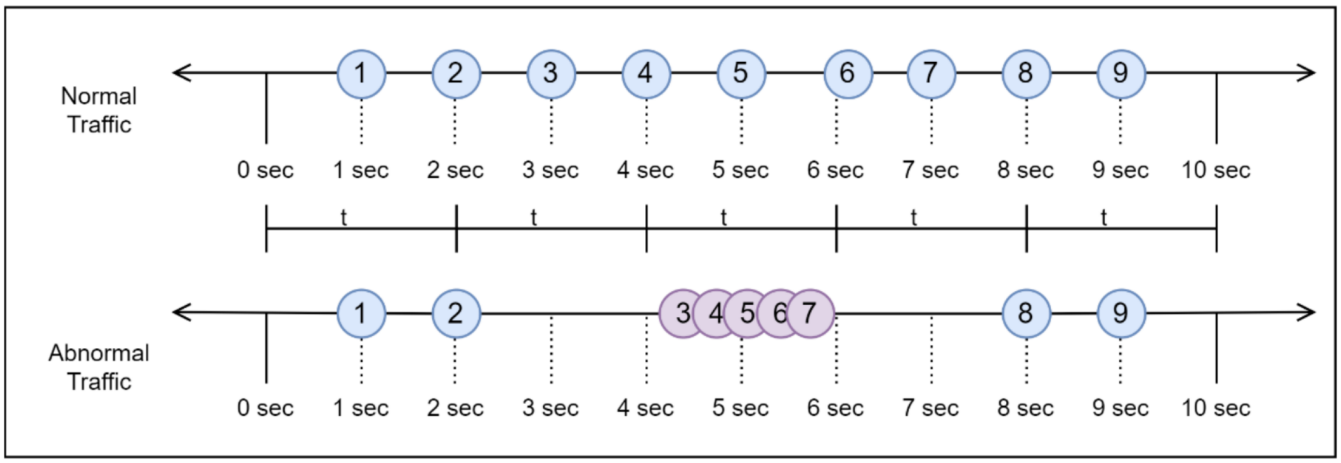

FIGURE 2 | The effect of the time difference between successive packets.

TABLE 2 | The predefined flows for the two algorithms with name and keys.

#	Algorithm	Predefined flow name	Keys
1	TFAD	tcp_flow	ipsource, ipdestination, tcpsourceport and tcpdestinationport
2	STFAD	tcp_syn_flow	ipsource, ipdestination, tcpsourceport, tcpdestinationport and tcpflags

flow like source and destination IP addresses, source and destination port numbers, protocol type, TCP tags, and other relevant packet header information. When a packet arrives at a network switch or controller that supports OpenFlow protocol, the flow keys are extracted from the packet's header and compared against the flow entries in predefined flows and trigger them via a handler called *setFlowHandler* [28, 29]. The predefined flows that we used for the two algorithms are demonstrated in Table 2.

The mitigation phase relies on the flow-rules subsystem provided by the ONOS SDN controller that defines how network traffic should be processed and forwarded at the switches within the network. This subsystem consists of match conditions, including OpenFlow attributes and the corresponding actions that apply to the matching incoming packets [30]. The corresponding actions determine what should be done to the matched incoming packets. The mitigation actions allowed include forwarding a packet to a specific port, modifying packet headers, dropping a packet, or sending a packet to the controller for further processing. In the proposed algorithm, we drop the incoming packets that match the characteristic of the attack flow and send an alert to the administrator who can restore the connections to the normal operation.

The ONOS SDN controller has representational state transfer (REST) APIs to create a flow rule via a POST request where the data is represented in JavaScript Object Notation (JSON) format [31]. The flow rule specifies conditions that packets must meet to match the rule, such as Ethernet type, IP address, and TCP port number. If a packet matches the rule, it will be processed according to the specified priority and timeout values. The flow rule is associated with a specific device ID and can include actions to be taken on matching packets. The descriptions of common condition attributes are illustrated in Table 3.

In our study, the selection of threshold values for the TFAD and STFAD algorithms for the testing environment is a crucial step to determine the effectiveness of the anomaly detection mechanisms [21]. For the TFAD algorithm, we considered a range of threshold values ranges from 1500 to 1700 bytes per time window. However, for the STFAD algorithm, we used threshold values ranging from 170 to 180 IAT per time window. These values are chosen based on our observation of the peak traffic for these threshold values. Based on these findings and observations from our experiments, we determined the optimal threshold values for each algorithm to be used in our network environment.

3.1 | TCP Flood Attack Detection and Mitigation Algorithm

The TCP flood attack detection (TFAD) Algorithm assumes that the attacker sends to the victim server many large TCP packets without setting any TCP flags. The attacker first creates a TCP connection with the victim server by completing the handshaking process and then uses the four tuples of the connection to send large packets with unmatched sequence numbers. TFAD algorithm monitors and tracks these TCP packets and identifies them using the four tuples, source and destination IP addresses, source, and destination TCP port numbers. The accumulative size of TCP packets is calculated for that flow. If it exceeds the predefined threshold value in bytes per time window, the traffic is classified as abnormal, and appropriate measures are taken to mitigate its effect as illustrated in Algorithm 1. The definitions of variables used in the TFAD algorithm are described in Table 4 where the threshold value represents the maximum amount of traffic flowing in one TCP connection given in bytes per time window (T).

TABLE 3 | ONOS flow rule attributes and their descriptions.

#	Attribute	Description
1	Priority	Determines the precedence of a rule flow. When multiple rules match a packet, the rule with the highest priority value will be applied. Higher priority values indicate higher precedence.
2	Timeout	Specifies the duration for which a rule remains active before it expires. After the timeout period, the rule is automatically removed from the flow table.
3	isPermanent	Indicates whether a rule is permanent or temporary.
4	deviceId	Represents the unique identifier of the network device to which the rule flow is applied.
5	Treatment	Defines the actions to be taken for a packet that matches a particular rule.
6	Selector	Defines the match conditions for a rule flow. The selector specifies the criteria that incoming packets must meet to match the rule.
7	ETH_TYPE	Refers to the Ethernet type field in the packet header.
8	IPV4_DST and IPV4_SRC	Refer to the IPv4 destination and source addresses in the IP header.
9	IP_PROTO	Refers to the IP protocol field in the IP header. TCP is protocol six.
10	TCP_SRC	Refers to the source TCP port number of the packet.
11	TCP_DST	Refer to the destination TCP port number of the packet.

ALGORITHM 1 | Pseudocode of TFAD Algorithm.

Predefined variables: *threshold*, *time_window*(*in seconds*), $database_2$.
Input: P (in bytes)

If a packet P arrives at a network switch or controller and the sFlow-rt handle it, **then:**
  Generate an ID for the packet P based on its IP address of source and destination, TCP source port and TCP destination port.
  If P is tracked in, $database_2$, **then:**
    Fetch the related stored record R for P from $database_2$ based on ID.
    Calculate the duration D between the arrival time of R and the arrival time for the P.
    If the duration D is less than or equal to the *time_window*, **then:**
      Increment the attribute packets_size of R by the size of P.
    Otherwise, *set* the attribute packets_size of R to the size of P.
  Otherwise, track the P by creating a record $\overline{R}$ for it in $database_2$ based on ID and set the attribute packets_size of $\overline{R}$ to the size of P.
  If the packets_size value for each record in $database_2$ is greater than the threshold, **then**:
    Create a new logger instance in the controller console based on the record attributes as a TCP Flood Attack.
    Create a flow rule in the controller via post request that drops all packets associated with the record destination IP and destination port.
  End If
End If

3.2 | Synchronized TCP Flood Attack Detection and Mitigation Algorithm

In a TCP-SYN flood attack, the attacker sends many TCP [SYN] packets to the victim without waiting for TCP [SYN, ACK] packets and never sends an [ACK] to finish establishing the connection. The synchronized TCP flood attack detection (STFAD) Algorithm detects this type of attack. It monitors and tracks incoming TCP packets and identifies the flow by the source and destination IP addresses, source and destination TCP port numbers and TCP tags attributes. The number of TCP [SYN] packets sent by the attacker and the number of TCP [SYN, ACK] packets sent by the victim are cumulatively counted. If the counter value exceeds the predefined threshold value within a given time, the traffic is classified as abnormal, and appropriate measures are taken. The Pseudocode of the STFAD algorithm is illustrated in Algorithm 2. The threshold value here represents the maximum difference between [SYN] TCP packets and [SYN, ACK] TCP packets for a TCP connection within a fixed time window (T).

The proposed algorithms can be broken down into two stages. The first is the initialization of variables to store the configuration settings and tracking data. The second is the flow handler that processes TCP flow events to extract the flow data like IP addresses and TCP information and generate a key for tracking, check if the key is already being tracked, and evaluate the TCP threshold to act accordingly. If the counter of a specific flow exceeds a predefined threshold, necessary actions are taken including sending a hypertext transfer protocol (HTTP) request to block that traffic and updating the control information and logging relevant details. The time complexity for the flow handler is linked to the number of TCP packets processed. The mitigation stage includes adding a rule to block suspicious traffic. It constructs a message and sends an HTTP request to block traffic on the switch. Thus, the time complexity can be approximated as $\boldsymbol{O(n)}$, where n represents the number of TCP packets processed.

TABLE 4 | TFAD algorithm variables definitions.

#	Variable	Definition
1	*Threshold*	It is a predefined value that the TFAD Algorithm uses to decide whether the traffic is normal or abnormal. It represents the maximum amount of traffic flowing in one TCP connection given in bytes per *time_window*.
2	*time_window*	Present the time duration in seconds.
3	$database_1$	Present a collection of key value pairs that all tracked packets are stored in. The key presents the ID of the packet, and the values present all packet attributes.

ALGORITHM 2 | Pseudocode of STFAD Algorithm.

Predefined variables: *threshold*, *time_window*(*in seconds*), $database_3$
Input: *P* (in bytes)

If a packet *P* arrives at a network switch or controller and the sFlow-rt handle it, **then:**
  Generate an ID for the packet *P* based on its IP address of source and destination, TCP source port and TCP destination port.
  If *P* is tracked in, $database_3$, **then:**
    Fetch the related stored record *R* for *P* from $database_3$ based on ID.
    Calculate the duration *D* between the arrival time of *R* and the arrival time for the *P*.
    If the duration *D* is less than or equal to the *time_window***then:**
      If the [SYN] flag of *P* is set, **then**:
        Increment the attribute SYN_counter of *R* by one.
      If the [ACK, SYN] flags of *P* are set, **then**:
        Increment the attribute ACK_SYN_counter of *R* by one.
    Otherwise:
      If the [SYN] flag of *P* is set, **then**:
        Set the attribute SYN_counter of *R* to one.
      If the [ACK, SYN] flags of *P* are set, **then**:
        Set the attribute ACK_SYN_counter of *R* to one.
  otherwise, track the *P* by creating a record $\overline{R}$ for it in $database_3$ based on ID.
  If the [SYN] flag of *P* is set, **then**:
    Set the attribute SYN_counter of $\overline{R}$ to one.
  If the [ACK, SYN] flags of *P* are set, **then**:
    Set the attribute ACK_SYN_counter of $\overline{R}$ to one.
  if the SYN_counter for the record *R* or $\overline{R}$ stored in $database_3$ has a value > 0 and the ACK_SYN_counter = 0, **then**:
    Identify this flow as coming from the source and find the related destination from $database_3$.
    If the difference between the value of SYN_counter from the source and the value of ACK_SYN_counter from the destination exceeds the predefined threshold value, **then:**
      Create a new logger instance in the controller console based on the record attributes as a [SYN] TCP Flood Attack.
      Create a flow rule in the controller that drops all packets associated with the record destination IP and destination port.
    End If
  If the ACK_SYN_counter for the record *R* or $\overline{R}$ stored in $database_3$ has a value > 0 and the SYN_counter = 0, **then**:
    Identify this flow as coming from the destination and find the related source from $dataset_3$.
    If the difference between the value of SYN_counter from the source and the value of ACK_SYN_counter from the destination exceeds the predefined threshold value, **then:**
      Create a new logger instance in the controller console based on the record attributes as a [SYN] TCP Flood Attack.
      Create a flow rule in the controller that drops all packets associated with the record destination IP and destination port.
  End If
End If

The variables used occupy a fixed amount of memory related to the number of flows attacking the victim and thus, the space complexity is a linear space complexity denoted as $\boldsymbol{O(m)}$ where m is the number of attacking flows. Variables are used for tracking and control information related to the monitored TCP packets, control settings, and rule implementations. The space complexity associated with these objects scales with the quantity of tracked flows and controls.

4 | Experimental SDN Testbed

Various tools, programming languages, and techniques were used to implement and test the proposed algorithms. We used an open-source network emulator called Mininet to emulate the SDN network which allows the creation of a virtual network

TABLE 5 | The versions of the tools, software, operating system, and programming language used in this paper.

Item name	Item version	Usage
Mininet	2.3.0	Used to emulate the SDN network.
ONOS controller	2.7.0	Used to manage and control network devices in an SDN environment.
sFlow-RT	3.0	Used to stream telemetry from sFlow agents embedded in network devices and convert the raw measurements into accessible actionable metrics.
Hping3	3.a2.ds2	Used to generate normal and abnormal network traffic.
Python	3.10.6	Used to construct the Mininet topology, launch http server and to perform attacks.
Ubuntu	22.04.2	Used as guest operation system for virtual machines.
VirtualBox	7.0	Used to create a virtual environment to construct our scenarios.

topology on a single machine [32]. Additionally, Mininet provides a command-line interface (CLI) and allows a Python script to configure and construct the network topology and fully supports the SDN network and its integration with various external or internal SDN controllers [33]. In our scenario, we selected the ONOS controller due to its characteristics of scalability, high availability, high performance, and the rich set of functions that it provides, including APIs and tools for network programming [34]. Additionally, there are only a few studies that used and investigated the SDN network with ONOS controller. The proposed detection algorithms relied on "sFlow-RT" software to monitor network traffic. "sFlow-RT" developed by InMon Corp uses the sFlow protocol to collect and analyze network traffic data, and provides a RESTful API for configuring measurements, retrieving metrics, setting thresholds, and receiving notifications [35]. Table 5 describes the versions of the tools, software, operating systems, and programming language used in this paper.

The proposed algorithms implementation and experiments were conducted using the open-source virtualization software called "VirtualBox." The host computer used is a high-performance machine with an AMD Ryzen 95900HX processor, 32 GB of RAM, and an NVIDIA GeForce RTX 3080 graphics card running the latest version of Windows 11. The virtual machines used Ubuntu 20.04 with reserved resources of four CPU cores, 12 GB of RAM, and a 22 GB HDD. Figure 3 shows the block diagram of the virtual setup and network topology.

The SDN network topology in Mininet includes three switches (S1, S2, and S3), six hosts (1–6), and a remote controller (C1). Since the environment is virtual, the Open vSwitch (OVS) is used to enable SDN capabilities with the OpenFlow protocol [36]. The bandwidth link between switches S1 and S3 is limited to 5 Mbit/s with a queue length of 500 packets [37], while other links have a 10 Mbit/s bandwidth due to the unlimited capacity of Mininet [38, 39]. Python code is used for the implementation of Mininet testbed topology.

The ONOS controller was installed on a separate virtual machine accessed via IP address on port 6653 for SDN and TCP port 8181 for its graphical user interface. Default built-in applications were configured to run on the ONOS controller, including the OpenFlow Provider Suite for ARP/NDP host location and reactive forwarding for end-to-end traffic management. The sFlow-RT software was installed on the same Mininet virtual machine and can be accessed at TCP port 8008. To operate the virtual SDN environment, the ONOS controller and sFlow-RT are accessed via the CLI, while Mininet topology is operated via the terminal shell with the required parameters, including the path for the Mininet topology python script, the sFlow-RT script path, type of links, remote controller address, switch type, and OpenFlow protocol. To run the experimental setup, ONOS controller and sFlow-RT engine are launched and the Mininet topology is constructed with the required parameters, including the sFlow-RT script path, Linux traffic-controlled links (tc) with capacity of 10 Mb/s, remote controller IP address, switch type = OVS, and OpenFlow13 protocol. Figure 4 shows the snapshot of the SDN network identified by the ONOS controller including the hosts and the switches.

For the transmission control protocol (TCP) and synchronized TCP flood attacks, the function *Popen* of the *subprocess* module of Python is used to create a new process and execute a terminal command line within the python environment. The code consists of a loop where each iteration, normal and abnormal network traffic is generated based on the type of the attack and using a command line network tool hping3 where the output of this script is used to determine the accuracy of the proposed detection algorithm. Also, a python code is launched on the victim host on port 80 as an HTTP server called "http.server" to capture the response network traffic on the victim host.

5 | Results and Discussion

In the first experiment, a deliberate spike in network traffic was generated. The objective of this experiment is to simulate a TCP flood attack directed toward host h6. Figure 5a represents the flow of data traffic measured in bytes per second during this attack as observed on the attacker's side. The attack commenced precisely at 21:32:29.867 and the traffic rapidly intensified, reaching its peak rate which exceeded 1 MB per second. The TFAD algorithm as shown in Figure 5b successfully detects the attack at 21:32:33.168 using a threshold value of 400 KB per second. The detection time which is the time between the initiation of the attack and the detection of the attack is 3.3 s. As a result, network flow at the victim host h6 significantly decreased to zero bytes per second.

In the second experiment, SYN flood attack is conducted from host h1 to host h5. As depicted in Figure 6a, the attack

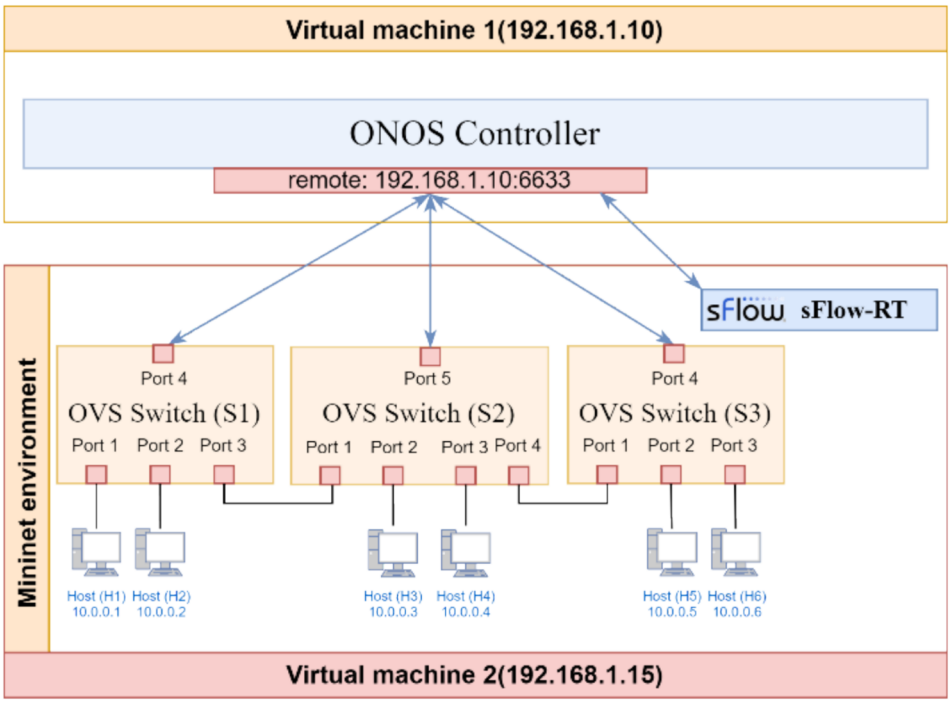

FIGURE 3 | The virtual enlivenments setup and the network topology.

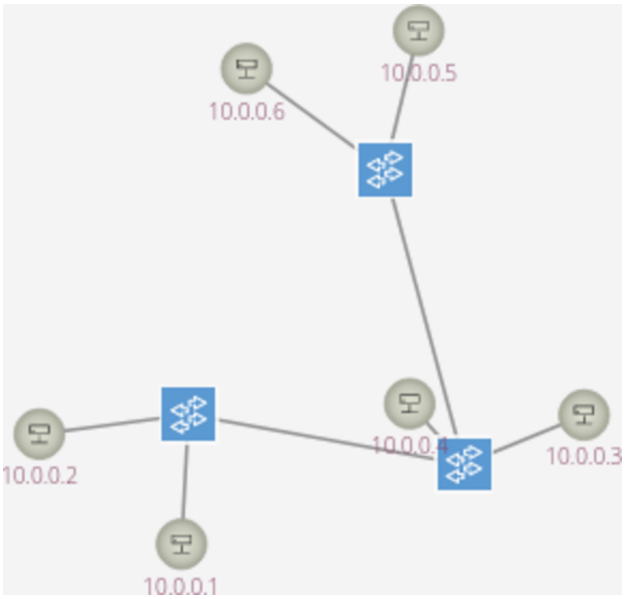

FIGURE 4 | A snapshot for network topology identified by ONOS.

commenced at 20:32:25.029 and promptly identified by STFAD algorithm at 20:32:26.169, taking approximately 1.14 s for detection. The number of [SYN] TCP packets sent from the attacker reaches a total of 1005, whereas the victim responds with only four [SYN, ACK] TCP packets. This resulted in a difference of 1001 between them, surpassing the predefined threshold value set at 1000. After the attack is detected, a rule flow established to effectively discard all packets that satisfied predetermined conditions involving source IP address (10.0.0.1). As a result, network flow on host h5 dramatically decreased to zero bytes per second, as shown in Figure 6b.

The evaluation of the proposed algorithms in detecting DDoS attacks involved the use of a Python script. For the determination of suitable threshold value for each algorithm, the attacks are conducted many times. Each experiment is repeated three times for each threshold value to ensure reliability. The results illustrate that the peak value of the traffic generated in our experiment range between 1500 and 1700 bytes per second. Furthermore, within a two-second period, the number of TCP-SYN requests generated ranged from 170 to 180 requests.

To examine the efficiency of the proposed algorithms, we use two performance parameters, accuracy and $F1$ score. The accuracy is the percentage of correctly predicted packets, and $F1$ score is the harmonic average between the precision and the recall. The precision is the percentage of legitimate packets from all the packets that reached the destination, and the recall is the percentage of legitimate packets that reached the destination. The $F1$ score is used to balance precision and recall and its value ranges between 0 and 1 where one is the best value. The formulas for accuracy and $F1$ score are as follows:

$$Accuracy = \frac{(TP + TN)}{(TP + FP + TN + FN)} \tag{2}$$

$$Precision = \frac{TP}{(TP + FP)} \tag{3}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{4}$$

$$F1\ Score = \frac{2 * Precision * Recall}{(Precision + Recall)} \tag{5}$$

Where the TP stands for True Positive and represents the number of packets that were correctly detected by the algorithm as normal

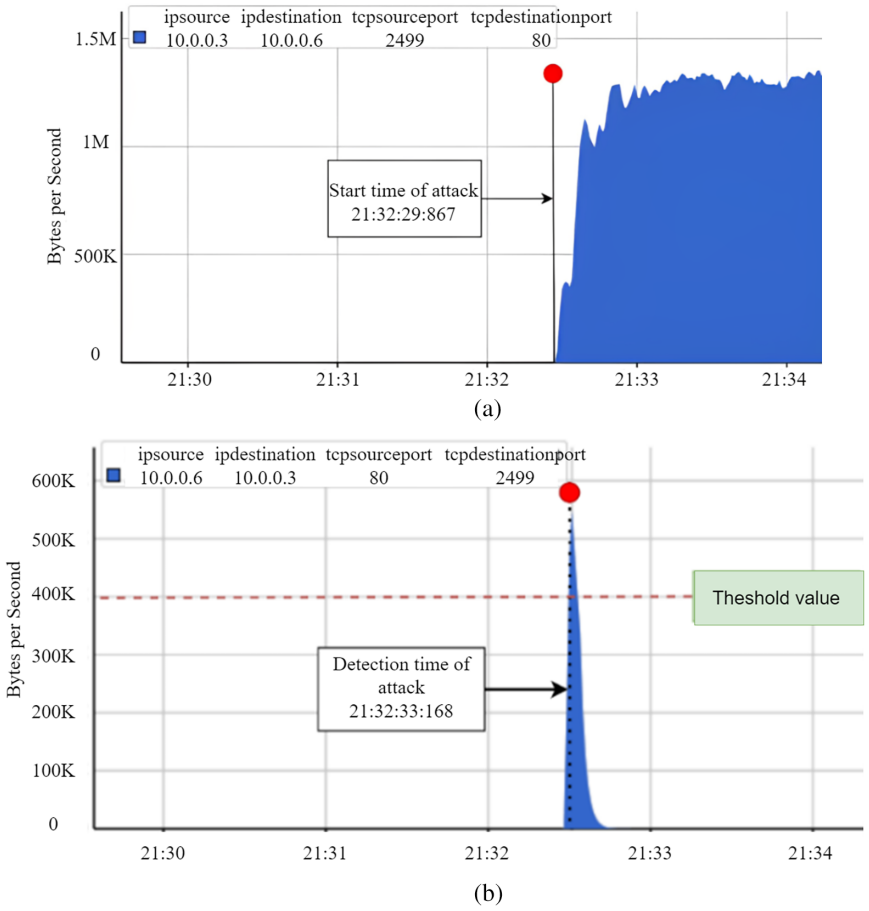

FIGURE 5 | (a) TCP traffic during flood attack at (a) the attacker (b) the victim.

traffic. FP stands for False Positive and represents the number of normal packets that were incorrectly classified as an attack. TN stands for True Negative and represents the number of normal packets correctly classified as normal, and FN stands for False Negative and represents the number of attack packets missed by the algorithm.

Table 6 shows the evaluating results of TFAD algorithm at different threshold values measured in byte per time window. The experiment is executed three times for each threshold value and the maximum, minimum, and average detection time value is recorded. The table provides a comparison of some performance parameters of the algorithm at different threshold values. As the threshold value increases, the average detection time measured over three runs of the experiment increases too. It is also observed that when the TCP traffic volume is less than the threshold value, the TFAD algorithm successfully detects all attacks as expected. When the threshold value is 1500 bytes per second, only one attack is detected because the peak of the normal network traffic value ranged between 1500 and 1700 bytes per second as mentioned earlier. There are no detected attacks for threshold values greater than or equal to 1800 bytes per second because the size of the network traffic did not exceed the threshold value. All attacks detected by the TFAD algorithm are successfully mitigated by implementing a rule flow in the ONOS controller.

Table 7 focuses on the performance of the STFAD algorithm in detecting and mitigating TCP-SYN flood attacks. The table provides the values for detection time and the number of generated and detected attacks at different threshold values. The threshold value is the difference between TCP-SYN packet and TCP-SYN-ACK packet. As the threshold value increases, the average detection time increases too. It is observed that when the differences are less than the threshold value, the STFAD algorithm successfully detects all attacks. When the threshold value is between 170 and 180, some of the packets are considered an attack because the number of TCP-SYN requests generated by normal traffic ranges from 170 to 180 requests as explained earlier. Additionally, there are no detected attacks when the threshold values greater than or equal to 190 because the number of differences between [SYN] TCP packets and [SYN, ACK] TCP did not exceed the threshold value. All attacks detected by the STFAD algorithm were successfully mitigated by implementing a rule flow in the ONOS controller.

Based on the data presented in Tables 6 and 7, it can be observed that TFAD and STFAD have a low false positive rate because the normal traffic features are vastly different from the attack traffic features. However, when the threshold value exceeds the network's traffic for both algorithms, there is a notable decrease in the true positive rate. This suggests that the larger threshold values are not suitable for detecting generated attacks. To achieve optimal performance, an effective algorithm would exhibit a high count of true positives and true negatives, while minimizing false positives and false negatives. TFAD successfully detects all attacks with threshold values up to 1700 bytes per second,

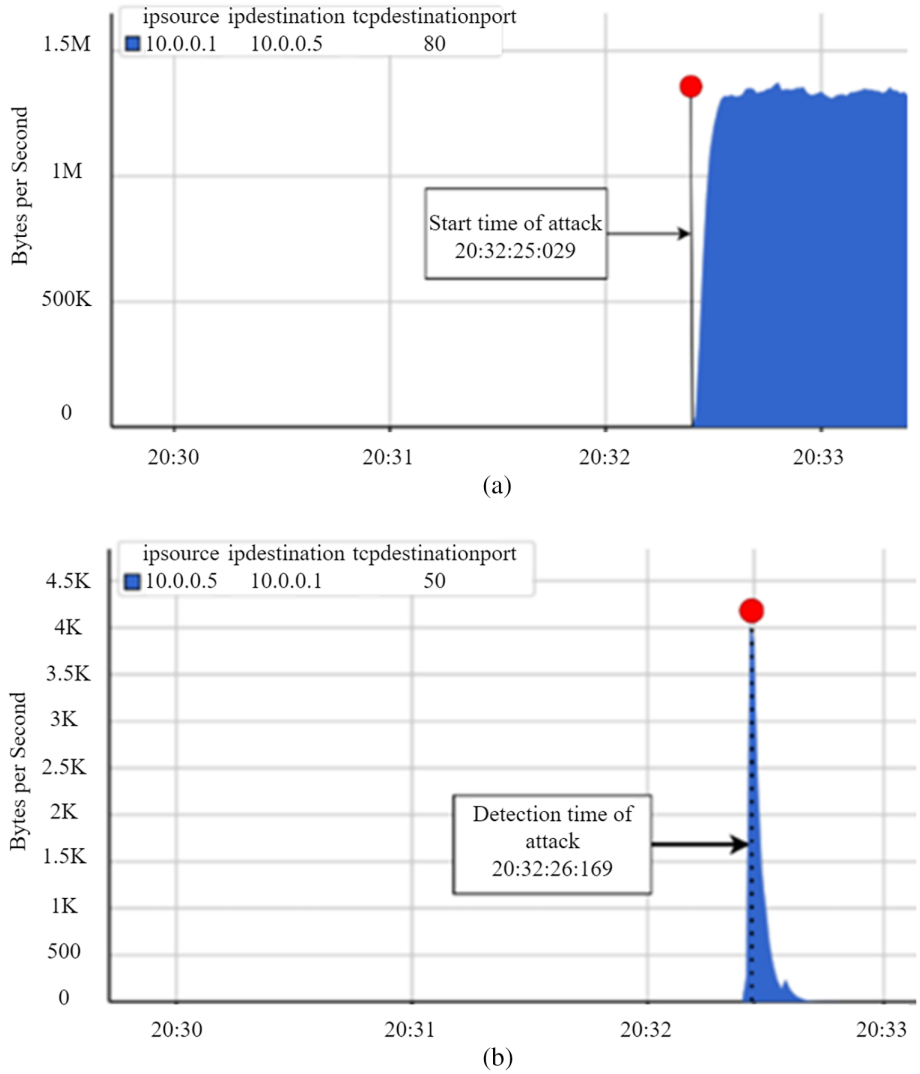

FIGURE 6 | TCP traffic during a SYN flood attack at (a) the attacker (b) the victim.

resulting in a high number of true positives and negatives and a small number of false positives and false negatives. However, when the threshold value exceeds 1700 bytes per second, the algorithm fails to detect any attacks accurately. This leads to an increase in FN and TN counts but decreases TP and FP counts. However, STFAD demonstrates excellent performance with an accuracy score of 1.0 for the first seven threshold values. It correctly classifies all instances as either TP or TN without generating any FP or FN errors. As we raise the threshold value to 170, there are five missed attack instances classified as FN leading to slightly reduced accuracy at 0.95. Beyond this point, increasing thresholds result in more occurrences of false negatives causing lower $F1$ scores. Therefore, it is important to analyze the network traffic and adjust the threshold value based on an algorithm like load shedding algorithm to achieve a balance between detection accuracy and false positives.

Table 8 provides a detailed explanation of the average detection time for each proposed algorithm. The average detection time indicates the speed at which the algorithm can detect SDN attacks. As the threshold value increases, the average detection time also tends to increase. A higher threshold value, often corresponding to a more stringent criterion for detecting attacks, may lead to a longer average detection time. On the other hand, a lower threshold value may result in a shorter average detection time but could potentially lead to a higher false positive rate. Striking a balance between the threshold value and the average detection time is crucial for effective and efficient attack detection.

The performance of detecting any of the attacks under study depends on the values of the thresholds. The threshold value is based on the difference between [SYN] TCP requests and [SYN, ACK] TCP responses per time window. It can help identify the optimal threshold value that balances detection accuracy and network performance. The values of all thresholds should be calculated statistically based on the average value and standard deviation of their corresponding values in normal traffic. The threshold value is the main design parameter of all these algorithms to successfully detect and block the attacks at an early stage. The presented data demonstrates the efficient performance of the proposed algorithms in detecting and mitigating DDoS while minimizing any adverse impact on the system functionality.

TABLE 6 | Evaluation parameters of TFAD algorithm.

#	Threshold value (bytes per time window)	Minimum detection time (s)	Maximum detection time (s)	Average detection time (s)	Number of generated attacks	Number of detected attacks	TP	FP	TN	FN	Accuracy	*F*1 score
1	1000	0.946	1.956	1.3432	50	50	50	0	50	0	1	1
2	1100	1.794	2.019	1.884	50	50	50	0	50	0	1	1
3	1200	1.83	2.06	1.9174	50	50	50	0	50	0	1	1
4	1300	1.897	4.006	2.557	50	50	50	0	50	0	1	1
5	1400	1.902	4.162	3.850	50	50	50	0	50	0	1	1
6	1500	5.959	5.959	5.959	50	1	1	0	50	49	0.5102	0.6766
7	1600	1.933	9.227	5.58	50	2	2	0	50	48	0.5306	0.6939
8	1700	9.164	9.164	9.164	50	1	1	0	50	49	0.5102	0.6766
9	1800	—	—	—	50	0	0	0	50	50	0.5	0
10	1900	—	—	—	50	0	0	0	50	50	0.5	0
11	2000	—	—	—	50	0	0	0	50	50	0.5	0
12	2100	—	—	—	50	0	0	0	50	50	0.5	0
13	2200	—	—	—	50	0	0	0	50	50	0.5	0

TABLE 7 | Evaluation parameters of STFAD algorithm.

#	Threshold value (SYN and ACK differences per time window)	Minimum detection time (s)	Maximum detection time (s)	Average detection time (s)	Number of generated attacks	Number of detected attacks	TP	FP	TN	FN	Accuracy	*F*1 score
1	100	1.425	1.694	1.5304	50	50	50	0	50	0	1	1
2	110	1.58	1.862	1.692	50	50	50	0	50	0	1	1
3	120	1.55	1.813	1.658	50	50	50	0	50	0	1	1
4	130	1.679	1.891	1.807	50	50	50	0	50	0	1	1
5	140	1.799	3.927	2.989	50	50	50	0	50	0	1	1
6	150	1.889	3.951	3.4504	50	50	50	0	50	0	1	1
7	160	3.7199	4.008	3.8741	50	50	50	0	50	0	1	1
8	170	1.971	8.048	5.4635	50	45	45	0	50	5	0.95	0.8182
9	180	8.181	9.718	8.9495	50	2	2	0	50	48	0.52	0.0769
10	190	—	—	—	50	0	0	0	50	50	0.5	0
11	200	—	—	—	50	0	0	0	50	50	0.5	0
12	210	—	—	—	50	0	0	0	50	50	0.5	0
13	220	—	—	—	50	0	0	0	50	50	0.5	0

TABLE 8 | Average detection time for each proposed algorithm.

#	Algorithm	Average detection time (s)
1	TFAD	4.032 s
2	STFAD	3.430 s

Accurately assessing the utilization of hardware resources such as CPU, network traffic, and memory is essential to analyze the efficiency of software and quantify the impact of application execution on the resources. Line 1 in Figure 7 represents the start of a TCP flood attack commenced precisely at 7:50:12.878, while line 2 depicts the end of the attack mitigation. The detection commenced precisely at 7:50:13.169 where the CPU utilization at the controller reached a maximum value of 37%, the sending network traffic reached around 140 KiB/s, and the receiving traffic reached 90 KiB/s. However, after mitigating the attack, both CPU usage and network traffic decreased significantly. Memory usage is almost negligible since its usage percentage increases from 67.4% to 69.5% during the detection and mitigation phases. These results demonstrate that the proposed algorithms do not consume

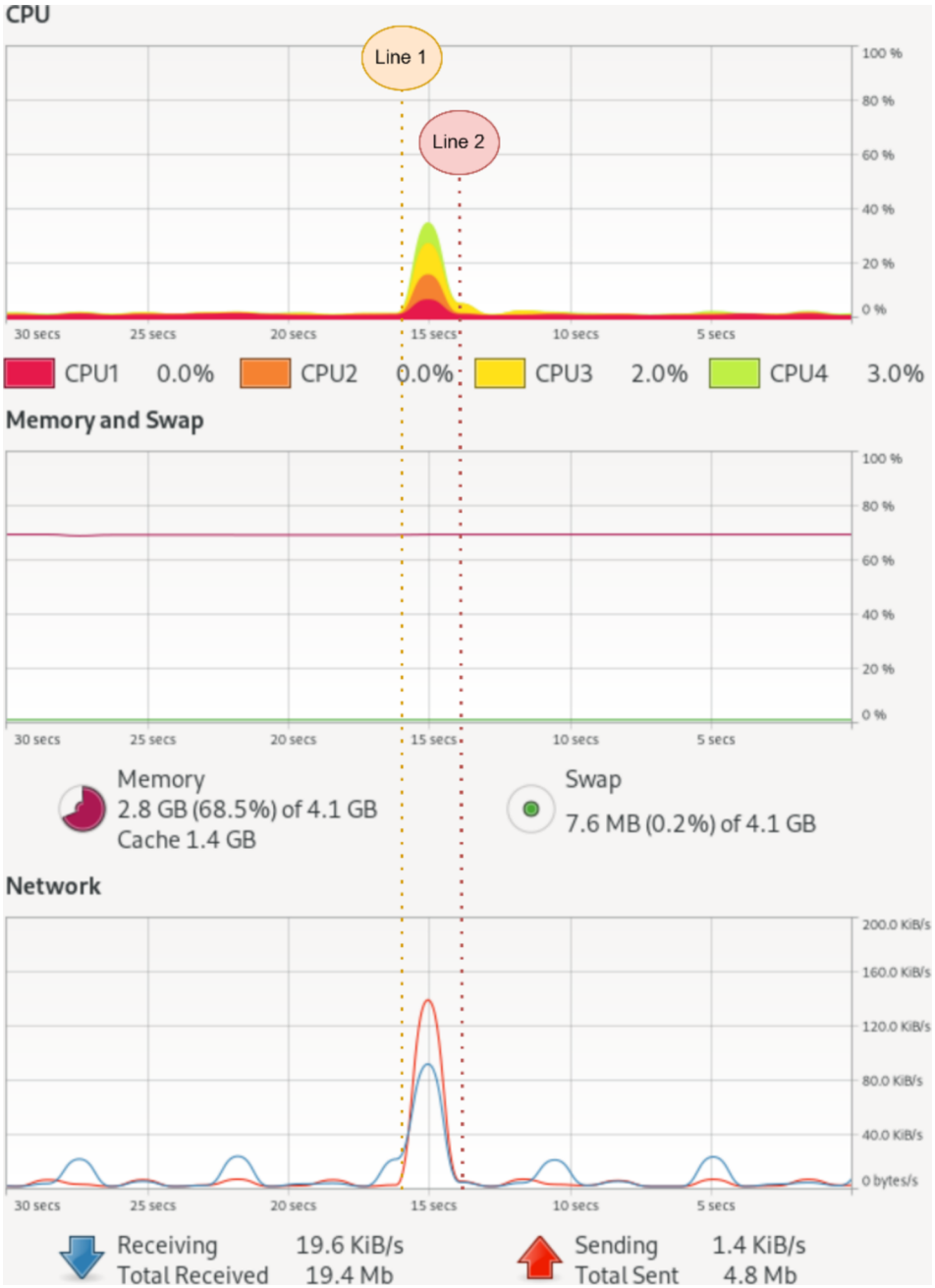

FIGURE 7 | Resources utilization during the TCP flood attack at the victim host h5.

the available resources extensively. This is an important consideration for real-time intrusion detection and mitigation systems, as it ensures that the system remains responsive and available to legitimate users even during an attack.

6 | Comparison With Previous Work

It is important to note that direct comparisons with previous works can be unfair sometimes and challenging due to variations in the methodologies employed and the data used for testing across different studies. However, Table 9 highlights the performance parameters of the proposed system in detecting and mitigating DDoS attacks in comparison with other algorithms and systems in the literature. Some of these studies use the statistical approach (S), another one uses the machine learning approach (ML), and two others use statistical approach to calculate a few parameters that become an input to a machine learning approach (S, ML). It is also noted that only our proposal and the one by Aslam, Srivastava, and Gore [22] developed a specific monitoring application as part of the research, while all others only develop detection and/or mitigation algorithms. The worst mitigation time for each study is recorded and it reflects the longest duration taken to mitigate an attack, it is the time from the detection of attack to the time of mitigating the effect of the attack, measured in seconds. The worst detection time indicates the longest time taken to detect an attack, it is the time from the start of the attack to the time of detection, also measured in seconds. Even though these parameters are very important, some studies

TABLE 9 | ONOS flood defender in comparison with other proposals in the literature.

References	Year of publication	SDN controller	Approach	Application developed	The worst mitigation time	Maximum accuracy achieved	The worst detection time	Maximum CPU overhead
Chouikik et al. [20]	2024	Open Daylight	S	✗	—	—	—	—
Linhares et al. [21]	2023	Floodlight	S, ML	✗	—	97.4	—	—
Aslam, Srivastava, and Gore [22]	2022	ONOS	ML	✓	7 s	99.3	8	100%
Girdler and Vassilakis [23]	2021	POX	S	✗	3.93	—	2.33	—
Oo et al. [24]	2020	ONOS	S	✗	2 but traffic is not shown	99.6	—	—
Tuan et al. [11]	2020	POX	S, ML	✗	—	98.8	At window size =3 3 s	—
Birkinshaw, Rouka, and Vassilakis [25]	2019	POX	S	✗	—	—	Hammer 0.25 CB-TRW ∞	97.6%
Our proposal	2024	ONOS	S	✓	0.291	100	1.5304	35%

did not report them or report the values without showing the traffic and how they measure these values. Maximum accuracy is also recorded for each study, but the comparison is challenging for this parameter because it is a function of the threshold in the statistical approaches. Finally, we also compare the maximum CPU overhead which represents the maximum percentage of CPU resources consumed during the process.

Despite these discrepancies, our proposed algorithm demonstrates robust performance metrics, achieving the highest accuracy of 100% with a remarkably low worst mitigation time of just 0.291 s. In contrast, several previous works exhibit longer mitigation times and varying accuracy levels. For instance, Aslam, Srivastava, and Gore [22] achieved a maximum accuracy of 99.3% but with the worst mitigation time of 7 s. Similarly, while Girdler and Vassilakis [23] reported a worst detection time of 2.33 s, their maximum accuracy was not specified. Other studies, such as those by Linhares et al. [21] and Tuan et al. [11], show that while they utilize machine learning approaches, their performance does not match the efficiency of our proposed solution.

Overall, while we assert that the performance of our proposed algorithm is competitive and often superior to most existing statistical and machine learning models in the context of DDoS attack detection and mitigation, the variability in testing data and approaches necessitates caution in drawing definitive conclusions.

7 | Conclusion and Future Work

The paper introduces two statistical detection and mitigation algorithms of TCP and TCP-SYN DDoS flood attacks in SDN networks integrated with the ONOS SDN external controller, Mininet, and sFlow-RT in real-time. Detection of attacks is done at an early stage with a minimum detection time for TFAD and STFAD algorithms of 1.3432 and 1.5304 s, respectively. The proposed algorithms have shown their ability to adapt to a variety of network conditions by utilizing tuned threshold values based on network traffic volume and IAT of packets. Detection accuracy also remains consistently high when the traffic is less than the threshold value. Future work should focus on using a statistical algorithm to calculate the best threshold value according to the current network traffic. Additionally, expanding the algorithms to cover other SDN flood attacks like those using http and DNS protocols, would further enhance the comprehensiveness of the detection and mitigation of the proposed system. We should also modify the proposed algorithms to work in a network with multiple SDN controllers.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

1. M. B. Jimenez, D. Fernandez, J. E. Rivadeneira, L. Bellido, and A. Cardenas, "A Survey of the Main Security Issues and Solutions for the SDN Architecture," *IEEE Access* 9 (2021): 122016–122038, https://doi.org/10.1109/ACCESS.2021.3109564.

2. T. Han, S. R. U. Jan, Z. Tan, et al., "A Comprehensive Survey of Security Threats and Their Mitigation Techniques for Next-Generation SDN Controllers," *Concurrency and Computation* 32, no. 16 (2020): e5300, https://doi.org/10.1002/CPE.5300.

3. N. McKeown, T. Anderson, H. Balakrishnan, et al., "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review* 38, no. 2 (2008): 69–74.

4. D. Carrascal, E. Rojas, J. M. Arco, D. Lopez-Pajares, J. Alvarez-Horcajo, and J. A. Carral, "A Comprehensive Survey of In-Band Control in SDN: Challenges and Opportunities," *Electronics* 12, no. 6 (2023): 1265, https://doi.org/10.3390/electronics12061265.

5. D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE* 103, no. 1 (2015): 14–76, https://doi.org/10.1109/JPROC.2014.2371999.

6. S. Bhardwaj and S. N. Panda, "Performance Evaluation Using RYU SDN Controller in Software-Defined Networking Environment," *Wireless Personal Communications* 122, no. 1 (2022): 701–723, https://doi.org/10.1007/s11277-021-08920-3.

7. T. Koponen, M. Casado, N. Gude, et al., "Onix: A Distributed Control Platform for Large-Scale Production Networks," in *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, vol. 2010 (Berkeley, CA: OSDI, 2019).

8. Y. Maleh, Y. Qasmaoui, K. El Gholami, Y. Sadqi, and S. Mounir, "A Comprehensive Survey on SDN Security: Threats, Mitigations, and Future Directions," *Journal of Reliable Intelligent Environments* 9, no. 2 (2023): 201–239, https://doi.org/10.1007/s40860-022-00171-8.

9. Cloudflare, "DDoS Threat Report for 2024 Q2" (2024), https://radar.cloudflare.com/reports/ddos-2024-q2.

10. D. Bastos, A. Guelfi, M. de Azevedo, A. Silva, and S. Kofuji, "Formal Modelling and Analysis of Man in the Middle Prevention Control in Software Defined Network," *Revista Eletrônica de Sistemas de Informação* 11 (2022): 16–40.

11. N. N. Tuan, P. H. Hung, N. D. Nghia, N. Van Tho, T. Van Phan, and N. H. Thanh, "A DDoS Attack Mitigation Scheme in ISP Networks Using Machine Learning Based on SDN," *Electronics (Switzerland)* 9, no. 3 (2020): 413, https://doi.org/10.3390/electronics9030413.

12. S. Jiang, L. Yang, X. Gao, et al., "BSD-Guard: A Collaborative Blockchain-Based Approach for Detection and Mitigation of SDN-Targeted DDoS Attacks," *Security and Communication Networks* 2022 (2022): 1–16, https://doi.org/10.1155/2022/1608689.

13. A. A. Wabi, I. Idris, O. M. Olaniyi, and J. A. Ojeniyi, "DDOS Attack Detection in SDN: Method of Attacks, Detection Techniques, Challenges and Research Gaps," *Computers & Security* 139 (2024): 103652, https://doi.org/10.1016/j.cose.2023.103652.

14. A. Khurum, "Removing SYN Flooding in TCP/IP Network," *Engineering Engrxiv Archive* (2019), https://doi.org/10.31224/osf.io/nwrj7.

15. M. Dimolianis, A. Pavlidis, and V. Maglaris, "SYN Flood Attack Detection and Mitigation Using Machine Learning Traffic Classification and Programmable Data Plane Filtering," in *2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops, ICIN*, vol. 2021 (Paris, France: IEEE, 2021), 126–133, https://doi.org/10.1109/ICIN51074.2021.9385540.

16. W. G. Negera, F. Schwenker, T. G. Debelee, H. M. Melaku, and Y. M. Ayano, "Review of Botnet Attack Detection in SDN-Enabled IoT Using Machine Learning," *Sensors* 22, no. 24 (2022): 9837, https://doi.org/10.3390/s22249837.

17. M. A. Kumar, E. M. Onyema, B. Sundaravadivazhagan, et al., "Detection and Mitigation of Few Control Plane Attacks in Software Defined Network Environments Using Deep Learning Algorithm," *Concurrency and Computation* 36, no. 26 (2024): e8256, https://doi.org/10.1002/cpe.8256.

18. P. Rajesh Kanna and P. Santhi, "Unified Deep Learning Approach for Efficient Intrusion Detection System Using Integrated Spatial–Temporal Features," *Knowledge-Based Systems* 226 (2021): 107132, https://doi.org/10.1016/j.knosys.2021.107132.

19. S. Wang, J. F. Balarezo, K. G. Chavez, et al., "Detecting Flooding DDoS Attacks in Software Defined Networks Using Supervised Learning Techniques," *Engineering Science and Technology, an International Journal* 35 (2022): 101176, https://doi.org/10.1016/j.jestch.2022.101176.

20. M. Chouikik, M. Ouaissa, M. Ouaissa, Z. Boulouard, and M. Kissi, "Detection and Mitigation of DDoS Attacks in SDN Based Intrusion Detection System," *Bulletin of Electrical Engineering and Informatics* 13, no. 4 (2024): 2750–2757, https://doi.org/10.11591/eei.v13i4.7570.

21. T. Linhares, A. Patel, A. L. Barros, and M. Fernandez, "SDNTruth: Innovative DDoS Detection Scheme for Software-Defined Networks (SDN)," *Journal of Network and Systems Management* 31, no. 3 (2023): 55, https://doi.org/10.1007/s10922-023-09741-4.

22. N. Aslam, S. Srivastava, and M. M. Gore, "ONOS Flood Defender: An Intelligent Approach to Mitigate DDoS Attack in SDN," *Transactions on Emerging Telecommunications Technologies* 33, no. 9 (2022): e4534, https://doi.org/10.1002/ett.4534.

23. T. Girdler and V. G. Vassilakis, "Implementing an Intrusion Detection and Prevention System Using Software-Defined Networking: Defending Against ARP Spoofing Attacks and Blacklisted MAC Addresses," *Computers and Electrical Engineering* 90 (2021): 106990, https://doi.org/10.1016/j.compeleceng.2021.106990.

24. N. H. Oo, A. C. Risdianto, T. C. Ling, and A. H. Maw, "Flooding Attack Detection and Mitigation in SDN With Modified Adaptive Threshold Algorithm," *International Journal of Computer Networks and Communications* 12, no. 3 (2020): 75–95, https://doi.org/10.5121/ijcnc.2020.12305.

25. C. Birkinshaw, E. Rouka, and V. G. Vassilakis, "Implementing an Intrusion Detection and Prevention System Using Software-Defined Networking: Defending Against Port-Scanning and Denial-Of-Service Attacks," *Journal of Network and Computer Applications* 136 (2019): 71–85, https://doi.org/10.1016/j.jnca.2019.03.005.

26. D. K. Sharma, T. Dhankhar, G. Agrawal, et al., "Anomaly Detection Framework to Prevent DDoS Attack in Fog Empowered IoT Networks," *Ad Hoc Networks* 121 (2021): 102603, https://doi.org/10.1016/j.adhoc.2021.102603.

27. S. Krishnan and E. O. J. Joel, "Mitigating DDoS Attacks in Software Defined Networks," in *2019 3rd International Conference on Trends in Electronics and Informatics* (New York: IEEE, 2019), https://doi.org/10.1109/icoei.2019.8862589.

28. B. Babayiğit and S. Karakaya, "Cost-Effective Logging Using SDN Architecture," in *International Conference on Advanced Technologies* (Karabük: Computer Engineering and Science, 2018).

29. "sFlow-RT Defining Flows," September 16, 2023, https://sflow-rt.com/define_flow.php#keys.

30. D. Sanvito, D. Moro, M. Gulli, I. Filippini, A. Capone, and A. Campanella, "ONOS Intent Monitor and Reroute Service: Enabling Plugplay Routing Logic," in *2018 4th IEEE Conference on Network Softwarization and Workshops* (New York: NetSoft, 2018), 272–276, https://doi.org/10.1109/NETSOFT.2018.8460064.

31. O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "SDN controllers: A comparative study," in *Proceedings of the 18th Mediterranean Electrotechnical Conference: Intelligent and Efficient Technologies and Services for the Citizen* (New York: IEEE, 2016), 1–6, https://doi.org/10.1109/MELCON.2016.7495430.

32. ONF, "MININET — Open Networking Foundation" (2018).

33. B. Lantz and B. O'Connor, "A Mininet-Based Virtual Testbed for Distributed SDN Development," *Computer Communication Review* 45, no. 4 (2015): 365–366, https://doi.org/10.1145/2785956.2790030.

34. P. Berde, M. Gerola, J. Hart, et al., "ONOS: Towards an Open, Distributed SDN OS," in *HotSDN 2014—Proceedings of the ACM SIGCOMM 2014 Workshop on Hot Topics in Software Defined Networking* (New York: ACM, 2014), 1–6, https://doi.org/10.1145/2620728.2620744.

35. O. E. Tayfour and M. N. Marsono, "Collaborative Detection and Mitigation of Distributed Denial-Of-Service Attacks on Software-Defined Network," *Mobile Networks and Applications* 25, no. 4 (2020): 1338–1347, https://doi.org/10.1007/s11036-020-01552-0.

36. "Open vSwitch," June 4, 2023, https://www.openvswitch.org/.

37. "Understand Queue Limits and Output Drops on IOS Software Platforms, Cisco Systems," June 4, 2023, https://www.cisco.com/c/en/us/support/docs/routers/7200-series-routers/110850-queue-limit-output-drops-ios.html.

38. B. Lanz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proceedings of the 9th ACM Workshop on Hot Topics in Networks, Hotnets-9* (New York: ACM, 2010), https://doi.org/10.1145/1868447.1868466.

39. "Introduction to Mininet," June 4, 2023, https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#limits.