Security in Software-Defined Networks Against Denial-of-Service Attacks Based on Increased Load Balancing Efficiency

Ying ZHANG, Hongwei DING*
Hebei Software Institute, Hebei, Baoding 071000, China

***Abstract*—The goal of software-oriented networks (SDNs), which enable centralized control by separating the control layer from the data layer, is to increase manageability and network compatibility. However, this form of network is vulnerable to the control layer going down in the face of a denial-of-service assault because of the centralized control policy. The considerable increase in events brought on by the introduction of fresh currents into the network puts a lot of strain on the control surface when the system is in reaction mode. Additionally, the existence of recurring events that seriously impair the control surface's ability to function, such as the gathering of statistical data from the entire network, might have a negative impact. This article introduces a new approach that uses a control box comprising a coordinating controller, a main controller that establishes the flow rules, and one or more sub-controllers that establish the rules to fend off the attack and avoid network paralysis. It makes use of current (when needed). The controllers who currently set the regulations are relieved of some work by giving the coordinating controller management and supervision responsibilities. Additionally, the coordinator controller distributes the load at the control level by splitting up incoming traffic among the controllers of the flow rules. Thus, a proposed method can avoid performance disruption of the flow rule setter's main controller and withstand denial-of-service attacks by distributing the traffic load brought on by the denial-of-service attack to one or more sub-controllers of the flow rule setter. The results of the experiments conducted indicate that, when compared to the existing solutions, the proposed solution performs better in the face of a denial-of-service assault.**

Keywords*—Security; open balance; denial-of-service attacks; software-oriented networks

I. INTRODUCTION

The next-generation network approach, known as a software-oriented network (SDN), allows for programmable control of the network and makes network management easier by separating network transmission from control operations [1]. The Open_ Flow protocol has been utilized the most out of all the tools that are currently available to actualize the software-oriented network [2]. The flow rules that the controller installs in the flow tables of the switches direct the network traffic in an Open_ Flow network. The controller can generate flow rules using either the proactive method or the reactive way, respectively [3]. In the pre-active method, before launching the software-based network, the controller installs the flow rules in the switches based on the predetermined strategy [4]. No special flow rules are put in the switches beforehand when using the reaction technique [5]. A table loss event occurs for each new flow that does not match the installed flow rules [6]. In response, an Open_ Flow request is delivered to the controller, and the controller chooses a new flow rule based on this request. The reactive technique, which is flexible, is typically employed in software-based networks [7].

Even though Open_ Flow has many benefits for streamlining network management and expanding its adoption, its reactive approach to installing flow rules makes it simple for DoS attacks on the controller to succeed because the controller must deal with all the packets generated by the absence of the table in [8]. The Open_ Flow network may receive a high number of transient phony flows that were created by a hacked host. Many flow requests (packet_in packets) are made to the controller when the Open_ Flow switch receives these malicious flows because they cause events linked to the loss of the table. Consequently, these high-frequency current requests deplete the controller's resources, interfering with regular functioning [9].

A denial-of-service attack on the Open_ Flow network often has the following effects: a) overloading of switches; b) congestion in the data plane and control plane communication channel; c) overloading of the controller; and d) overflowing of switch flow tables [10]. While the anomaly connected to the controller's service might cause disruption and failure of the entire network, overloading the switches and overflowing the switch flow tables only endanger the victim switch. As a result, the majority of threats from a denial-of-service attack in the Open_ Flow network are brought on by the emergence of congestion in the communication channel of the data level with the control level and overloading of the controller [11].

The suggested approach for the control level of the software-based network has been utilized in this article to strengthen the security of the network against denial-of-service attacks and boost its availability [12]. In order to increase the availability of the software-based network against significant changes in the events brought on by the arrival of new flows (due to the temporal and spatial characteristics of the network traffic) and the existence of repeated events, the proposed method consists of a control box [13]. It is intended to gather statistical data from the entire network, which overburdens the controller. A coordinating controller, a primary controller of the current setter, and one or more (as required) sub-controllers of the current setter are all included in this control box [14].

The current installation controllers' job is essentially decreased by giving the coordinating controller responsibility for managing and monitoring the software-based network. It is the responsibility of flow controllers to install flow rules in line with the network applications that are operating on them in order to configure the data level of the software-oriented network [15].

Additionally, the coordinator controller will categorize the incoming traffic in a statistical manner after activating one or more (as necessary) sub-controllers in response to the network's increased traffic load, which can be brought on by a denial-of-service assault [16]. Each of them receives a portion of the incoming traffic that the flow controller divides into known categories. This results in the flow controller's controllers sharing the load of incoming traffic, which lessens the effort of all of the flow controller's controllers. In the suggested method, the redundancy of the controllers and the division of labor among them are used to distribute the traffic caused by the denial-of-service attack to one or more sub-controllers, taking into consideration the multi-controller capability offered in Open_ Flow Specification 1.2. The software-based network controller will experience less of an impact from the attack thanks to the flow installer, and the software-based network will be more resilient to unexpected and severe fluctuations in network traffic. In summary, the article presents a solution to improve the security and resilience of SDNs against DoS attacks. The proposed approach, utilizing a control box with coordinating and sub-controllers, is shown to be effective in managing traffic during attacks and reducing response times. However, the article acknowledges the need to address resource consumption and scalability in future work.

In short, the contribution of the authors in this research is as follows:

- Introducing a new approach: A new approach has been introduced to increase the security and resilience of Software Defined Networks (SDN) against Denial of Service (DoS) attacks. It lies in the creation of a "control box" consisting of coordinating, main and sub-controllers.
- They also contribute to this field by comparing the proposed solution with four other existing methods for mitigating DoS attacks in SDN, including Ryu controllers with different mechanisms. Through this comparison, they show that their approach consistently outperforms these alternatives and provides a more effective way to counter DoS attacks.

The article's structure is described in the paragraphs that follow. The work that has been done to fortify the software-based network's control level against denial-of-service assaults has been mentioned in Section II. The software-based network controller's input load has been looked into in Section III. Section IV describes the suggested technique for enhancing the security of the software-based network control level. In Sections IV and V, it was examined, respectively, how denial-of-service assaults would affect the proposed approach and how a saturation attack on the control surface by the data surface would affect it. The effectiveness of the suggested approach against a denial-of-service attack is contrasted with that of many other solutions in Section VI, and conclusions and recommendations for further work are provided in Section VII and Section VIII respectively.

II. Related Works

Studies on data-level protection and control-level protection that aim to lessen the effects of denial-of-service attacks on software-based networks fall into these two groups. In order to fight against assault, data layer protection focuses on enhancing data layer functionality or adding new features [17]. In order to keep the controller from becoming overloaded with requests, OF-Guard uses a data-level cache. The strategy, however, lacks flexibility because it is uncertain whether such a cache will be established at the data level. The implementation of SYN Proxy is proposed in AVANT-GUARD [18] as a unit that performs the TCP Handshake in the switches before sending the incoming TCP stream to the network. The pressure on the switch buffer is increased using this unit, and there is also a cap on the number of proxy ports. Because LineSwitch can arbitrarily proxy flows from the same IP source that it has already established a TCP handshake with, it is suggested that LineSwitch will enhance AVANT-GUARD [19]. Another AVANT-GUARD-based solution is SDN-Shield, which employs a number of NFV-based attack mitigation units to counter distributed denial-of-service attacks at the software-oriented network data level. However, none of these SYN Proxy-based techniques work with other network protocols.

In order to sustain network policy enforcement, Flood-Guard offers a pre-active flow rule analyzer that can examine the source code of an Open_Flow-based application and generate several pre-active rules via running time monitoring of each application's global state. To produce based on the protocol, Flood-Guard stores the remaining packets caused by table loss in a cache at the data level. The source code analyzer, however, is extremely complicated and unable to be deployed across a network. Additionally, the same attack flow protocol was utilized by additional benign flows, so the data level cache cannot ensure fair behavior against them.

Enhancing the security of the controller policies and implementing detection and filter techniques to lessen denial-of-service attacks are the main goals of control level protection [20]. In Flow Ranger, the controller employs a rating algorithm to recognize typical users and pre-process packets with a higher priority. In order to distinguish malicious communication, SGuard introduces an access control system that leverages six items as feature vectors in the classification unit [21]. Researchers in [22] have suggested a scheduling policy for the proposal controller that divides the flows into various queues. The controller manages each queue based on round-robin scheduling, and each queue corresponds to a switch that is situated on the denial-of-service attack path. MLFQ creates an equitable sharing of control resources between the server and hosts in the network by utilizing a number of expandable and collapsible request queues [23]. Potentially hostile traffic is diverted to the intrusion detection system by SDN-Shield, and the impact of a denial-of-service attack is lessened by appropriately building the flow rules.

In order to lessen channel congestion between the victim switch and the controller, Flood Defender suggests a technique for distributing packets related to table loss from the victim switch to its surrounding switches. Additionally, for the purpose of detecting sporadic attacks, Flood Defender employs a two-stage filter and briefly archives packet_ in packets. To handle new benign flows, the packet_ in packet buffer introduces a significant delay. Additionally, the surrounding switches that are being flooded may suffer damage from the packet distribution caused by the absence of the table. In order to counter the denial-of-service attack, FMD employs a technique based on flow migration at the software-oriented network control level slow. The master controller, to which fraudulent requests with a high volume are delivered, is replaced in this technique by a slave controller. The master controller handles all typical Open_ Flow requests. In order to safeguard the channel of communication between the victim switch and the master controller, FMD sends the threatening Open_ Flow requests to the slave controller after identifying a denial-of-service attack. The migrated requests are briefly kept in the slave controller and transferred to the master controller for additional processing at a limited rate [24] to prevent the master controller from becoming overloaded. SGS is offered to defend the control plane from denial-of-service assaults, and its key component is the clustered deployment of several controllers in the control plane. The abnormal traffic detection and controller dynamic defense units make up the SGS procedures. Anomalous traffic detection uses quadratic feature vectors to separate phony flows from real ones by focusing on switches that already exist at the data level. The impact of the denial-of-service assault at the control level is minimized by the controller's dynamic protection, which involves remapping the controller and sending the access control message to the switch. As it was already indicated, the majority of the techniques for data layer protection include modifying Open_ Flow switches or incorporating unique features into the data layer. However, all techniques in the second category (control level protection) work to reduce the number of resources that a denial-of-service assault uses up. In reality, the controller cannot manage the rate of receiving Open_ Flow packets. Additionally, the main cause of the denial-of-service attack in the control level is the congestion of the communication channel connecting the data level with the control level, which is of the TCP or TLS type. Source in [42] has demonstrated the capacity to enhance networks and withstand DDoS attacks. Thus, the purpose of this survey is to review 65 articles about DDoS attack detection in SDN. As a result, each reviewed article's systematic reviews of the suggested methodology are examined. This work additionally analyzes the performance metrics and their best assault detection accomplishments from each research publication. Furthermore, this work reviews the reduction technique applied in each paper.

In this study, an architecture to handle service source attacks is developed in order to address the restrictions indicated above. A suggestion is to have flow rule setter sub-controllers that are multiples of the main flow rule setter controller and assign incoming traffic from denial-of-service attacks to them regardless of the protocol used, as well as from the channel connecting the data level with the level. Control is comprehensive since it guards against congestion and shields the main flow rule controller from denial-of-service attacks to minimize their negative effects on good new flows. Additionally, the concept is transparent because it does not require any adjustments to the network's infrastructure or application programs to be used.

III. Examining the Software-based Network Controller's Input Load

As depicted in Fig. 1, within a software-oriented network made up of Open_Flow switches, the Open_Flow controller periodically sends multi-part messages [25] to the switches under its control. These messages serve to collect statistical data from the switches, contributing to the construction of the network's information base (NIB). The controller can have a comprehensive view of the entire network under his control based on the information in this network's information base. The initial packet of each flow that reaches the input switch is examined to determine whether it complies with the rules set up in the switch's flow table. Suppose the switch's flow table does not contain a match for the packet. In that case, a packet-in message is provided to the controller containing the lowest priority flow entry that is compatible with any incoming flow (flow entry absent from the table). The request in this message is to begin planning a path for the incoming stream. The Open_Flow controller determines the appropriate route for the incoming flow based on information from the network information base. The route is established by installing the appropriate flow rules in the flow tables of the switches along the route, and as a result, all flow packets are directed to their intended destination. Eq. (1) determines the input load to the controller resulting from the input currents with the input speed of the current per second in a data center with N switches, assuming the average number of switches located in the path is [N/2] and for each flow in the flow table of switches located in the path of two flow rules (to forward in two directions) installed.

$$L_{arrival-flows} = \lambda \times (\lceil m_{pi} \rceil + \lceil m_{po} \rceil + \left\lceil \frac{N}{2} \right\rceil \times 2 \times \lceil m_{fm} \rceil) \quad (1)$$

where, m_{fm} stands for the size of the flow_mod packet, which is the packet delivered from the controller to the Open_Flow switches to install a new flow rule. Additionally, m_{pi} and m_{po} stand for the sizes of the packets in and out, respectively. As a result, there is a chance that the controller will become overloaded by increasing the speed at which fresh currents enter the input switch in the reaction mechanism chosen for each incoming current to be passed into the controller.

The input load brought on by gathering statistical data at the network level is another element that overburdens the software-based network controller. Important statistical data that can be gathered from existing Open_ Flow switches in a data center includes data about the ports connected to servers and data about the flows set up in the switches' flow tables. Eq. (2) can be used to determine the input load to the control level in a data center as a result of gathering statistical data from Open_ Flow switches.

$$L_{statistics} = N \times (([m_{portreq}] + H \times [m_{portrep}] + [m_{flowreq}] + F \times [m_{portrep}])) \times [\frac{1}{T}] \quad (2)$$

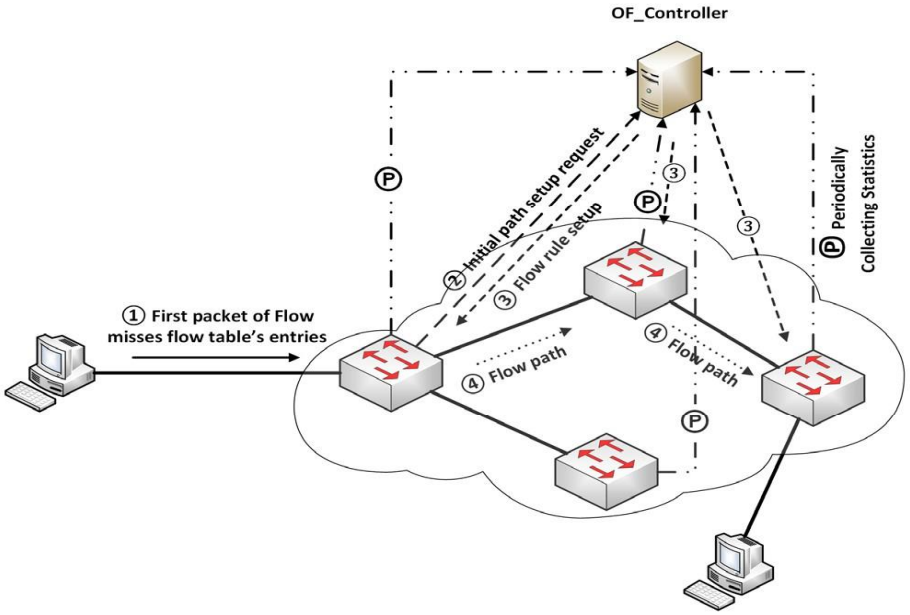


Fig. 1. The software-oriented network's controller operation mechanism creates a path for a new flow.

TABLE I. THE VARIABLES THAT ARE USED TO CREATE THE INCOMING TRAFFIC TO THE DATA CENTER

Name	Model	Parameters
New flow arrivals	Poisson	$\lambda(t) = 192$
Internal rack flow ratio	Bernoulli	$R_{int} = 0.8$
TCP flow ratio	Bernoulli	$R_{tcp} = 0.85$
Flow duration, Dn	Pareto	$a_p = 1.504$ $M_p = 1.0001$
Flow transmission rate, Yn	Gaussian	$E[Y_n] = 4303.69$ $Var[Y_n] = 69936.37$
Flow size, Sn	$Y_n \times D_n$	$E[S_n] = 8517.97$

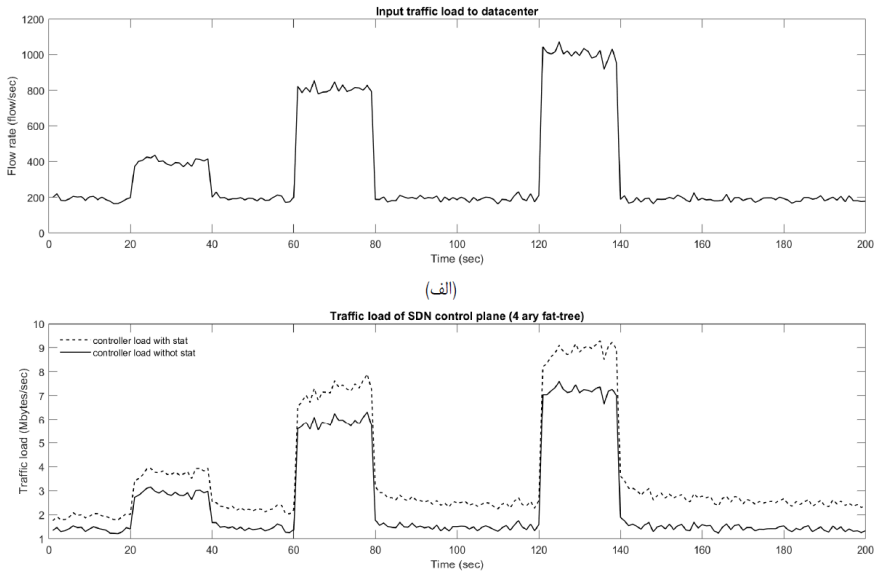


Fig. 2. (a) Incoming traffic to the data center and (b) Incoming load to the controller in a data center with _Ary_Fat_tree4 structure ((-- -) with the load resulting from the collection of statistical information and (-) without it.

where, H is the total number of switch ports linked to servers, and N is the total number of Open_ Flow switches in the data center. F is the number of installed currents in the switches, and T is the controller's periodic request for statistics data. A simulation has been run in MATLAB 2022b to show the impact of the load brought on by the gathering of statistical data (Relation 2) and the admission of fresh flows into the data center (Relation 1) on the software-based network controller.

The data center in this simulation has an Ary Fat_tree 4 structure. The data center's incoming traffic is generated based on the findings of Yoonseon Han and his associates [26], who used the values in Table I. Fig. 2 displays the simulation outcome based on (1) and (2), as well as the relationships offered by Bong-yeol Yu and his associates [27]. As seen in Fig. 2(a), the data center receives 192 flows per second of incoming traffic in the usual state. This amount is in the intervals to observe the impact of speeding up this flow rate on the amount of incoming load to the software-based network controller. There has been an increase in the current per second from 20 to 40 seconds, 60 to 80 seconds, and 120 to 140 seconds, respectively. The input load to the software-based network controller of a data center with an _Ary_Fat_tree4 structure is shown in Fig. 2(b) in two states: (-) with the load brought on by gathering statistical data and (-) without it. As might be predicted, as the pace of incoming traffic to the data center grows, so does the input load on the controller. The important feature in Fig. 2's lower portion is the sizeable input load brought on by the controller's statistical data collection, which has a significant impact on the occurrence of interference with the transmission of basic controller messages (such as current installation), delays the arrival of information to the controller, and ultimately lowers the controller's efficiency. Therefore, packet_in can improve the controller's efficiency and availability by separating the load arising from the gathering of statistical data from the load resulting from the arrival of packages.

IV. Suggestion for Enhancing the Security of SDN Control Levels

In software-based networks, the controller is not just in charge of performing the functions of a straightforward switch to transfer flows throughout the network in order to take advantage of its central management. It is required to gather statistical data from the network level in order to implement efficient programs, such as balancing the load on communication lines and servers in the network or quality control of service provided to flows system by system. For successful and integrated management, software-oriented networks need the controller to be highly available.

There are two ways to provide high availability for Open_ Flow controllers in software-based networks [28]. Reduce the load on controllers as the initial step in the approach. The Open_ Flow controller communicates with the Open_ Flow switches frequently, particularly while running in reactive mode, as was covered in the preceding section. The controller has become overwhelmed. As a result, making it is unable to process incoming messages. The second option is to replace one controller with many controllers to add redundancy.

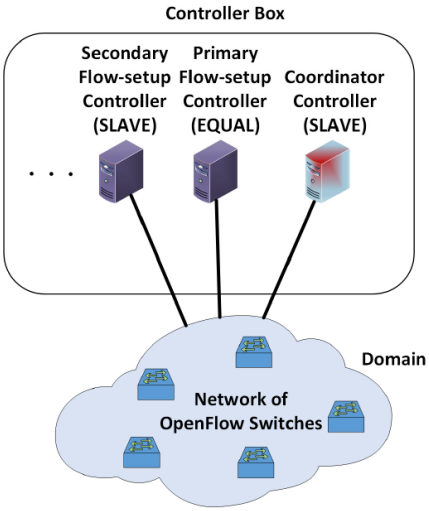


Fig. 3. The recommended approach for software-based network controllers' structure.

Both of the aforementioned methods are taken into account in the proposed control box, which is employed in the proposed method to increase the availability of Open_ Flow controllers in software-based networks (see Fig. 3). One of the controllers in this control box serves as a coordinator, connecting to Open_ Flow switches only to gather statistical data from them. Additionally, there is a main controller and one or more (depending on the demand) sub-controllers in this control box that are in charge of adding flow rules to the flow tables of Open_ Flow switches. So, in addition to lessening the strain on the controller, more controllers have been deployed by splitting the work between the coordinating controller and the controllers that establish the flow rules. It is necessary to employ several controllers to control Open_ Flow switches, which are accessible in Open_ Flow version 1.2 and later, in order to implement the control box concept in software-based networks. Since Open_ Flow version 1.2, it has been possible to have numerous controllers, each of which can control Open_ Flow switches in one of the three modes of SLAVE, MASTER, or EQUAL. The MASTER and EQUAL modes have complete access to the switch and can receive all asynchronous communications (such as packet-in) from the switch, among the other two modes [29].

Each Open_ Flow switch is permitted to communicate with one MASTER controller but numerous EQUAL controllers. In SLAVE mode, the controller can only read from the switches through its access to them; it is, therefore, unable to receive any other asynchronous messages than the answer of the multi-part message containing the switch's status. Each controller can modify its state by sending the switches the OFPT_ROLE_REQUEST message [30]. The switch transmits the OFPT_ROLE_REPLY message to the controller after receiving this message.

The other controllers of that switch will convert to the SLAVE state if the switch gets a message from it instructing it to change the controller state to MASTER. Due to the switch's ability to have several Open_ Flow channels, it is not necessary to re-establish the channel in the event that one of the

controllers connected to it fails. Open_ Flow switches are able to transmit packet-in messages solely to the controller whose ID is in the matching table entry by giving each of the flow rule set controllers in the control box a distinct ID. Their direction of flow is known. Utilizing the Nicira features integrated within the RYU network operating system; this capability is possible [31]. The NXTSetControllerId function is used to accomplish this by giving each flow rule controller a special identification number. The number of flow rule set controllers that can be used in the control box will thus no longer be constrained. As a result, the control box is expandable, allowing the number of active controllers that establish the flow rules to grow as necessary. This boosts the availability of the control level of the software-oriented network by dividing the load associated with packet-in messages among the controllers of the flow rules and reducing their individual loads.

A. *Coordinating Controller*

All Open_ Flow switches in the domain are connected to the coordinator controller that is operating in SLAVE mode. As seen in Fig. 4, the coordinating controller has a variety of components for regulating the operation of the controllers that establish the flow rules and for keeping an eye on the network. Allocating incoming traffic to the flow controller's sub-controller is the primary responsibility of the coordinating controller. When the coordinating controller notices that the main controller is on the verge of overflowing, this is what happens. By doing this, the existing installation's main controller's traffic load is lessened, and it is once again able to function.

In order to prevent the overload of the controller controllers as the amount of incoming traffic to the network increases, the coordinating controller repeats this process by turning on new flow controller sub-controllers and allocating the overload of incoming traffic to them (see Fig. 5). On the other hand, by lowering the load of traffic entering the network, the coordinating controller transfers all of the traffic load entering the network to the main controller of the flow setter while deactivating the sub-controller of the flow setter. The description of the coordinating controller's role as well as information on each of its component parts, are provided below.

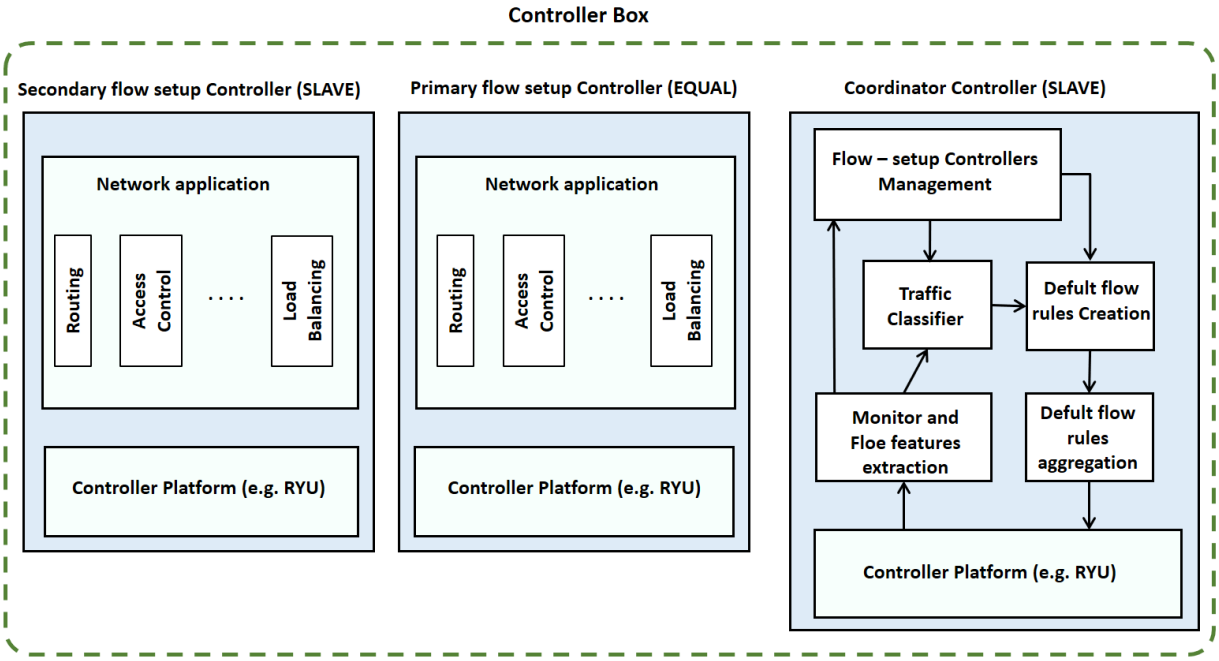


Fig. 4. Block diagram of the components of the proposed method.

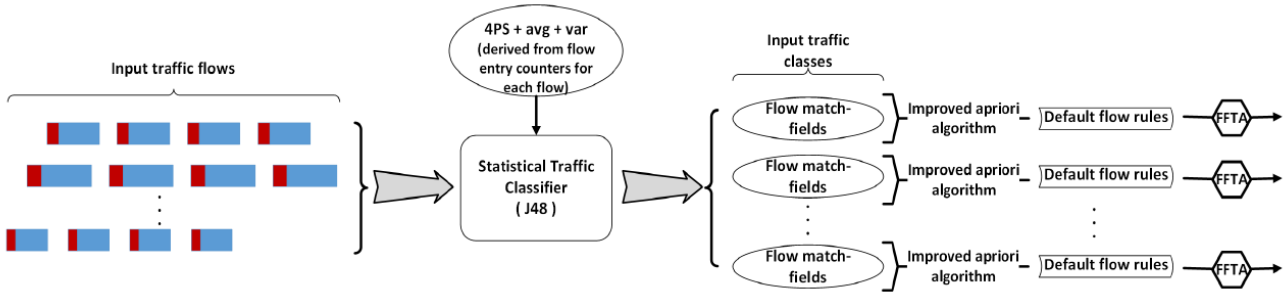


Fig. 5. The process of creating default flow rules and statistically classifying incoming network traffic.

B. How to Switch the Sub-controller On and Off

The main controller of the current installation may become overloaded as a result of changes in the rate at which traffic enters the network. The current setter's sub-controller is activated by the coordinating controller when it notices that the main controller is on the verge of overload as shown in Fig. 6(a). The flow controller's sub-controller receives the be-Active signal from the coordinating controller in order to accomplish this. The sub-controller of the flow installer sends the OFPT_ROLE_REQUEST message to all the Open_ Flow switches attached to it to alert them that its state has changed to EQUAL after receiving the be-Active signal. The sub-controller of the flow installer enters EQUAL mode after receiving the OFPT_ROLE_REPLY signal from the Open_ Flow switches. The coordinating controller changes its state from SLAVE to EQUAL upon receiving the activated signal from the flow controller's sub-controller, and after implementing the default flow rules related to one of the traffic categories entering the network, the output of the controller's classification algorithm. The gate switches activated current controller switches back to SLAVE mode.

The sub-controller of the current setter is deactivated once again by the coordinating controller when it notices that the main controller of the current setter is no longer in overload mode as shown in Fig. 6(b). The coordinating controller delivers the be-passive signal to the current setter sub-controller after switching the mode from SLAVE to EQUAL and removing the default flow rules connected to the active current setter sub-controller from the gateway switches. A remark is sent. The coordinating controller then reverses its state, going back from EQUAL to SLAVE. The flow installer's sub-controller sends Barrier request messages to all of the switches connected to it upon getting the be-passive signal, causing these switches to complete processing all of the messages they had been receiving from the flow installer's sub-controller. The flow controller sub-controller then sends the OFPT_ROLE_REQUEST message to all the switches connected to it, changing their status from EQUAL to SLAVE. The current regulator's sub-controller is rendered inactive as a result.

C. Investigating How the Proposal is Affected by a Control-Level Saturation Attack at the Data Level

A UDP flood attack was applied against three scenarios (the first scenario: using one controller, the second scenario: using a desaturation controller and a number of flow rule set controllers, and the third scenario: using the proposed architecture) in order to demonstrate the impact of redundancy on the availability of the proposed architecture. This experiment was carried out using the structure depicted in Fig. 7 and implemented in the Mininet 2.2.2 emulator environment [32]. According to Table II, this implementation uses OpenVSwitch [33] software switches for the Open_ Flow switches and the RYU platform [34] for the controllers. Additionally, the iPerf [35] tool has been used to produce UDP flood attack traffic in order to test the scenarios. The attacker launches a UDP flood assault on the Open_ Flow switch while a client is corresponding with the load balancer between servers at a consistent pace of 50 flows per second. In this experiment, the round-trip time (RTT) of client requests collected by running the ping command has been used to assess the performance of various scenarios. The round-trip time of client requests has been observed for all three situations, as seen in Fig. 8, despite the fact that the speed of the UDP flood attack rises linearly up to 800 packets per second (PPS). Fig. 8 shows that the proposal approaches saturation when the attack speed hits 800 packets per second, whereas for scenarios 1 and 2, this happened at 450 and 500 packets per second, respectively. In fact, the controllers of the flow rules are out of reach and are unable to install a new route on the Open_ Flow switch in exchange for the arrival of fresh flows once the attack speed surpasses the saturation point of each of the scenarios. The results show that the proposal has a high availability compared to alternative scenarios and is particularly resistant to the assault of saturating the control surface due to the usage of redundancy in the flow rule controllers.

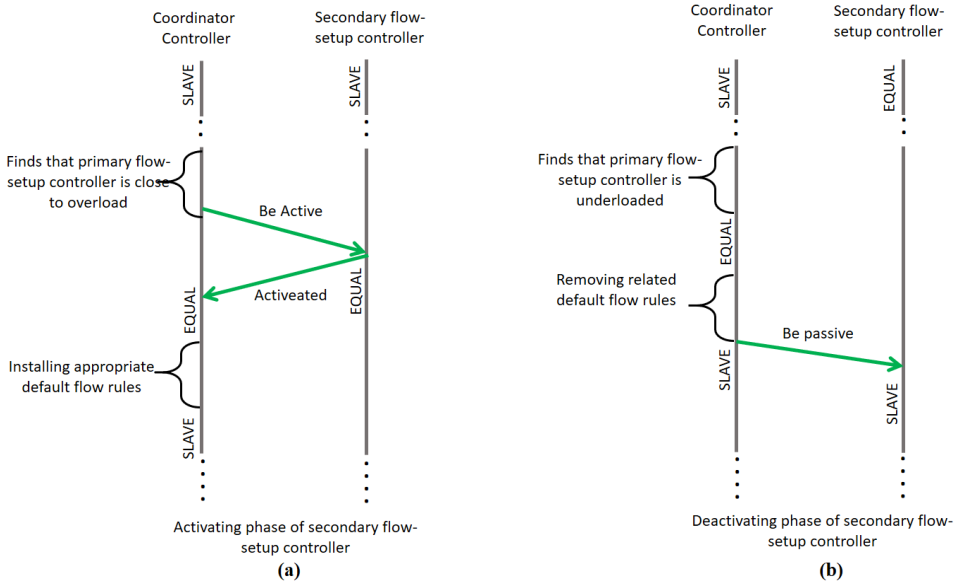


Fig. 6. How to modify the proposal's state of the existing regulator's sub-controller: a) activation and (b) deactivation.

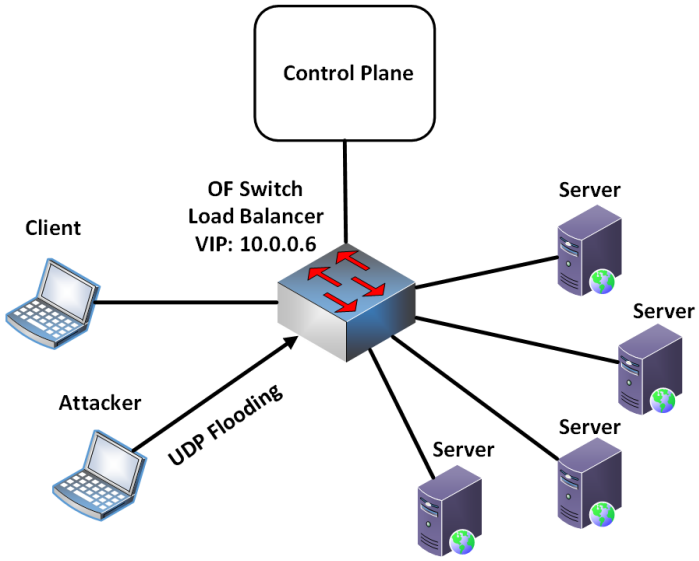


Fig. 7. A structure that was utilized for the UDP flood assault test.

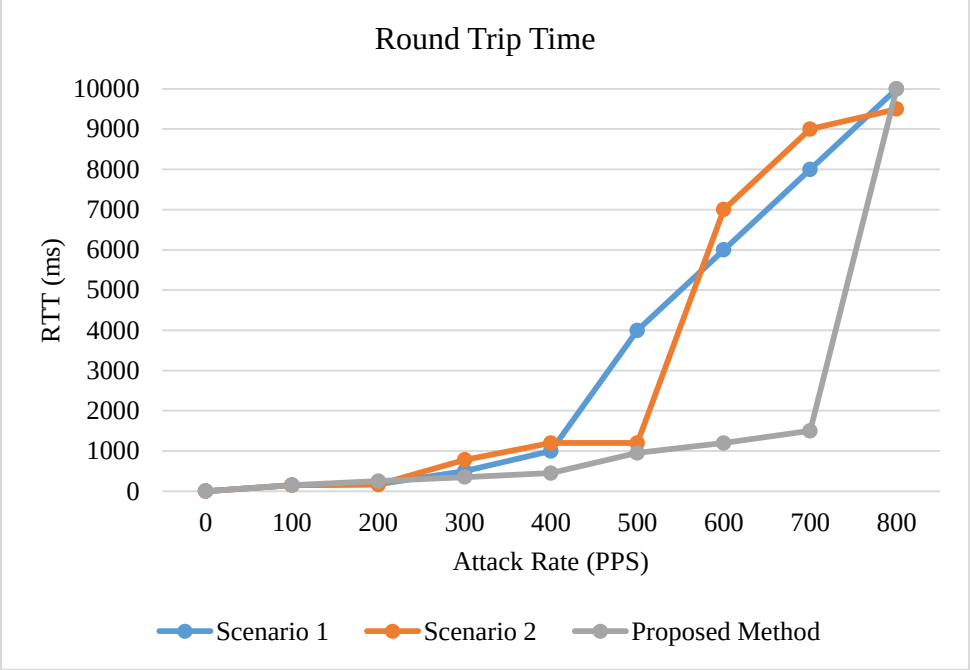


Fig. 8. Determining, during the flood attack, the turnaround time for client requests for proposals and scenarios 1 and 2. a) The first scenario: using one controller, (b) the second scenario: using a coordinating controller and a number of controllers that set the flow rules, and (c) the third scenario: using a proposed method.

TABLE II. LABORATORY PARAMETERS FOR THE PROPOSED EVALUATION

Virtual Machine	Oracle VM VirtualBox (Version 5.1.10 r 112026)
Guest OS	Ubuntu 16.04
CPU	Intel core i7-4720hq cpu@2.60ghz
RAM	16GB
Emulator	Mininet 2.2.2
Network Operating System	RTU 4.15
Open_Flow Switch	OpenVSwitch 2.4.0
Network Traffic Generator	iPerf 2.08b

V. Analyzing How Denial-of-Service Attacks Affect the Proposal

One of the elements affecting the availability of software-based networks is denial-of-service attacks. This section has researched how three different denial-of-service attacks affect the proposal. These attacks include ICMP/Ping flooding, UDP flooding, and TCP.SYN flooding. These attacks, which are a subset of volume denial-of-service attacks, were created using the Hyenae tool version 0.36 [36]. In this section, experiments were carried out utilizing the topology depicted in Fig. 7 that was constructed using the.222 Mininet simulator [37] and [38].

Additionally, the attacker conducts denial-of-service assaults on the Open_ Flow switch while a client is interacting with it (the switch is an Open_ Flow load balancer between servers) at a constant speed of 50 flows per second. The intended parameters in this test are the input load on the controllers in the proposed method, the round-trip time (RTT) of customer requests, and the impact of denial-of-service attacks on the proposed method's availability and performance. The ping command is used in conjunction with the 062bwm-ng v. utility to acquire these parameters. These attacks and the outcomes of their use have been addressed in the paragraphs that follow.

A. Study of the ICMP / Ping Flood Attack's Effects

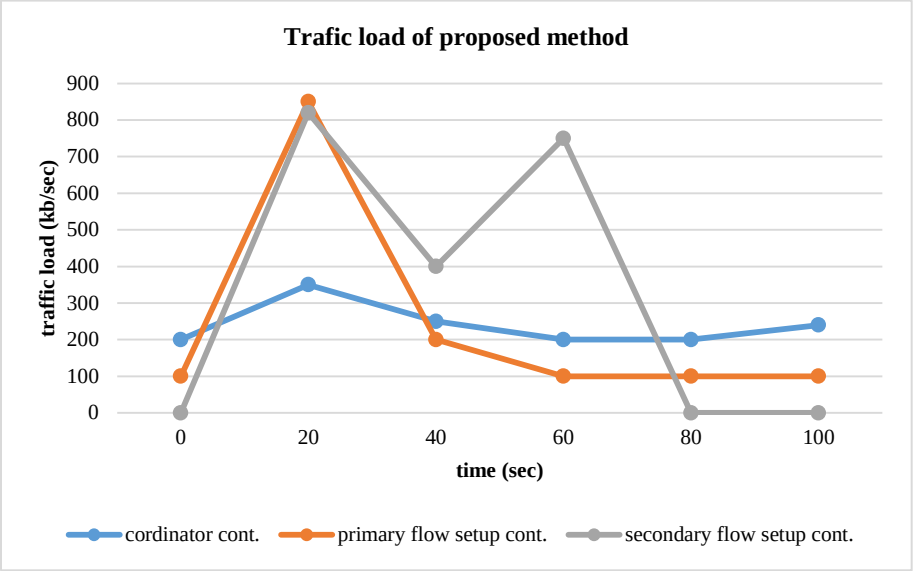
This kind of attack involves the attacker bombarding the target with ICMP echo requests, which interfere with other services that the target's other programs rely on. This kind of attack requires the attacker to rapidly transmit to the victim a large number of echo_request packets that appear to be coming from various sources and have random addresses. By having many flows, the attacker can interfere with the network. Each time a new packet-in is generated when an attacker attacks a victim using many sources with random addresses, the controller will experience a significant increase in traffic. The controller begins to sink when it receives a lot of packet-in packets, and after some time, it begins to disregard fresh incoming packets. The amount of production traffic described in terms of bytes, packets, or streams can be used to identify this kind of assault. Fig. 9 displays the outcomes of an ICMP/Ping flood assault conducted on the software-oriented network using the suggested technique, whose topology is depicted in Fig. 7. The volume of incoming traffic using the suggested method before, during, and after the attack is depicted in Fig. 9(a). Fig. 9(b) also depicts the duration of back-and-forth for client inquiries. The ICMP/Ping flood attack begins at 10 seconds and ends at 40 seconds, as can be shown. As soon as the attack begins, the time it takes for the customer's requests to move back and forth dramatically increases due to the main controller of the flow rules being overloaded and unable to handle requests entered as packet-in packages. The amount of incoming traffic load to the main controller of the flow rules setter is decreased by the activation of the sub-controller of the flow rules setter by the coordinating controller due to the detection of overloading of the main flow rules setter controller and the allocation of additional load resulting from the attack on it. The amount of incoming traffic to the main controller of the flow rules is decreasing, and as a result, this controller is once more able to handle and process requests that are received as packet-in packages. As a result, the amount of time required to go back and forth for customer requests returns to a reasonable value. This test demonstrates that the proposed method's availability is unaffected by an ICMP/Ping flood attack, save for a brief period of time (the time needed for the coordinating controller to notice that the main controller of the flow rules setter is overloaded before activating the sub-controller and adding additional load to it).

B. Investigating the Effect of UDP Flood Attack

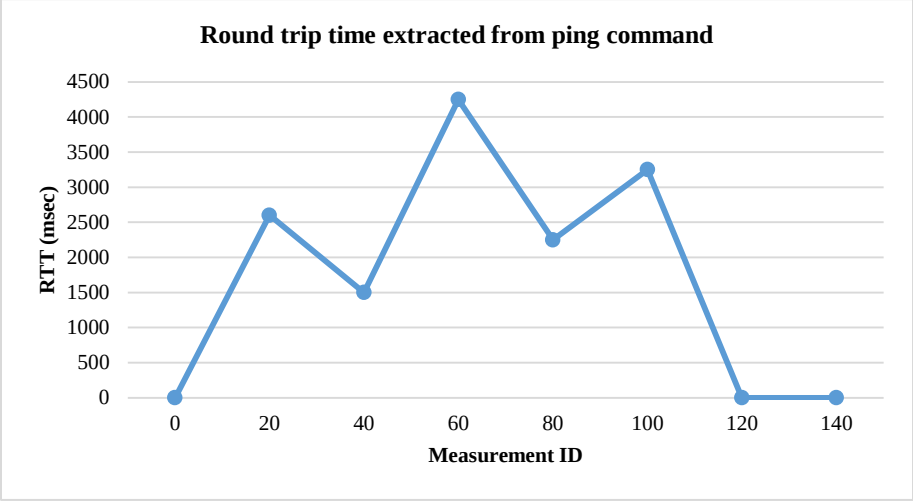
A UDP flood assault aims to bombard the target with a large volume of UDP packets. A host employs a huge number of bogus sources IP addresses when launching a UDP flood attack. The buffer overflow occurs in the victim as a result of a huge quantity of UDP packets arriving at the victim. Since sending a large number of bytes to the victim is the primary characteristic of UDP flood attacks, they produce the most traffic among denial-of-service attacks. In this attack, bogus source IP addresses are used to produce UDP packets that appear to send the target a lot of fresh streams per second. As a result, the software-based network controller starts pouring new flows after a given amount of time. The results of a UDP flood assault on a software-based network using the suggested technique, whose structure is depicted in Fig. 7, are displayed in Fig. 10. The volume of incoming traffic using the suggested method before, during, and after the attack is depicted in Fig. 10(a).

Additionally, Fig. 10(b) displays the amount of time spent responding to client inquiries. The UDP flood attack begins at 10 seconds and stops at 50 seconds, as can be shown. As soon as the attack begins, the time it takes for the customer's requests to move back and forth dramatically increases due to the main controller of the flow rules being overloaded and unable to handle requests entered as packet-in packages. Fig. 10(b) shows that this increase in time is caused by the software-based network controller receiving more incoming traffic than during the ICMP/Ping flood attack.

The amount of incoming traffic load to the main controller of the flow rules setter is decreased by the activation of the sub-controller of the flow rules setter by the coordinating controller due to the detection of overloading of the main flow rules setter controller and the allocation of additional load resulting from the attack on it. The main flow rule controller's ability to handle and process requests received in the form of packet-in packets is restored with the reduction of incoming traffic load, and as a result, the time required for the customer's requests to move back and forth to its reasonable value. Therefore, in this instance, similar to the ICMP/Ping flood attack, the UDP flood attack has no impact on the proposed method's availability other than for a brief time (the time needed for the coordinating controller to detect the main controller of the flow rules setter's load before activating the sub-controller and allocating additional load to it).

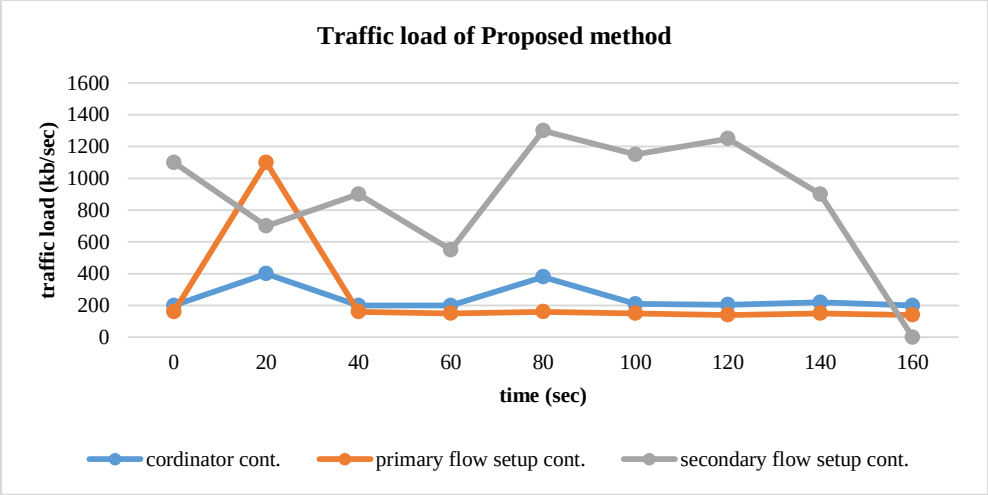


(a)

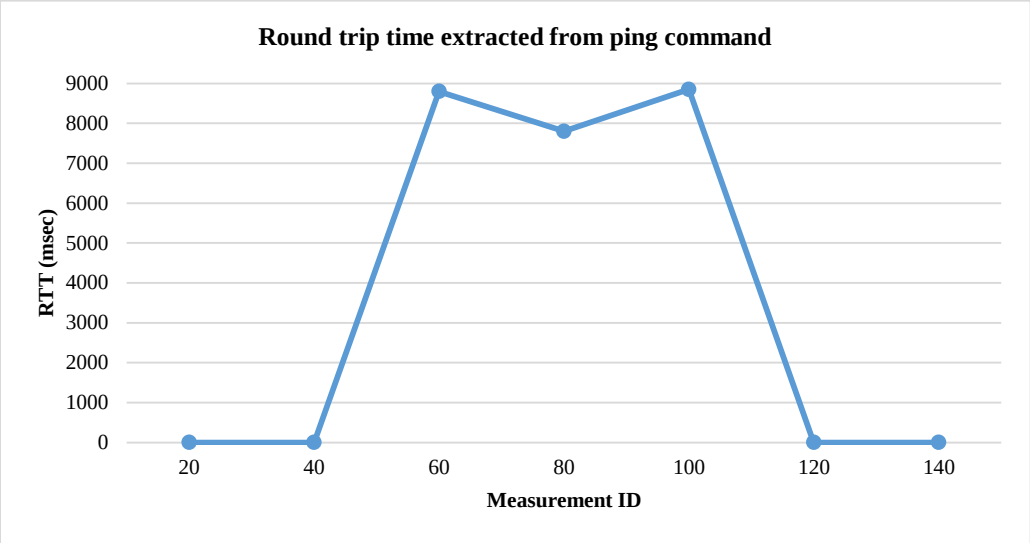


(b)

Fig. 9. (a) The amount of incoming traffic in the proposed method before the ICMP / Ping flood attack, during the attack and after it and (b) the round trip time (RTT) of customer requests.



(a)



(b)

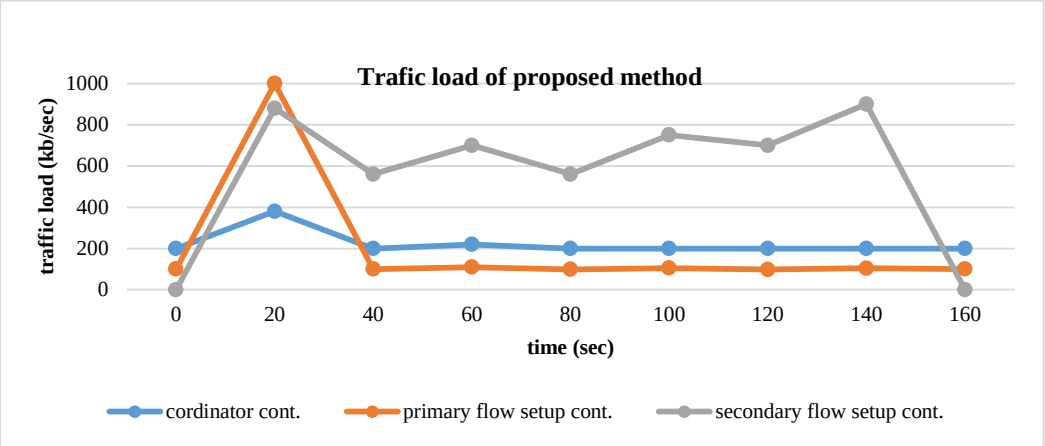
Fig. 10. (a) The amount of incoming traffic in the proposed method before the UDP flood attack, during the attack and after it and (b) the round trip time (RTT) of the client's requests.

C. Investigating the Effect of TCP_SYN Flood Attack

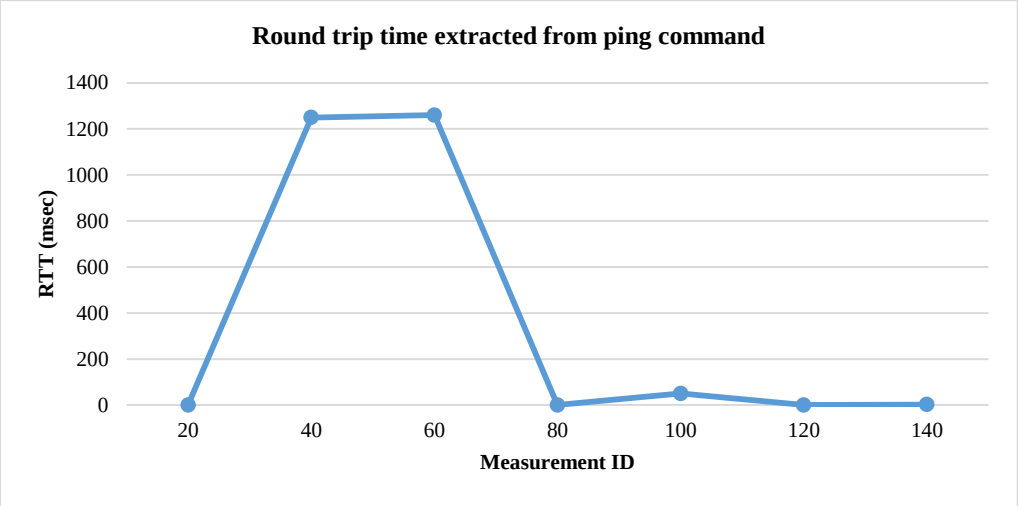
A TCP_SYN flood attack bombards the target with bogus SYN requests that were generated using fictitious source IP addresses. The victim never gets an answer to their SYN/ACK packets because the source IPs are faked. As a result, the attack's port is still open unnecessarily. All of the victim's ports are blocked as a result of several bogus SYN requests, making it impossible for the victim to connect to trustworthy people. It only takes a small amount of bandwidth for this kind of attack to keep the false connections open and render the victim unreachable. Hyenae tool version 0.36 was used [39] to develop a TCP_SYN flood attack, which begins by sending the victim a large number of low-speed streams containing TCP_SYN packets.

Additionally, the Open_Flow load balancing switch only distributes a load of incoming traffic across three servers to analyze the effect of the TCP_SYN flooding attack on the software-based network controller. From the fourth server, the round-trip time is measured. Customer requests are employed in (RTT). The software-based network controller can see a significant number of flows in this assault since random source IPs are being used to produce flow towards the target. This significantly hinders the efficiency of the software-based network controller by adding a lot of traffic to it.

The results of the TCP_SYN flooding assault on the software-based network using the suggested technique, whose structure is depicted in Fig. 7, are displayed in Fig. 11. The volume of incoming traffic using the suggested approach before, during, and after the attack is depicted in Fig. 11(a). Fig. 11(b) also depicts the duration of back-and-forth travel in response to customer demands. The TCP_SYN flood assault begins in three seconds and concludes in 45 seconds, as can be shown. As soon as the attack begins, the time it takes for the customer's requests to move back and forth dramatically increases due to the main controller of the flow rules being overloaded and unable to handle requests entered as packet-in packages. Due to the software-based network controller receiving far less incoming traffic than the UDP flood attack, the time increase seen in Fig. 11(b) is noticeably reduced. The amount of incoming traffic load to the main controller of the flow rules setter is decreased by the activation of the sub-controller of the flow rules setter by the coordinating controller due to the detection of overloading of the main flow rules setter controller and the allocation of additional load resulting from the attack on it. The main controller of the flow rules can once again handle and process requests that are received in the form of packet-in packets by reducing the amount of incoming traffic. As a result, the time it takes for the customer's requests to go back and forth returns to a reasonable value. As a result, in this instance, similar to ICMP/Ping and UDP flood attacks, the TCP_SYN flood attack has no effect on the proposed method's availability other than for a brief period of time (the time needed to detect the coordinator controller loading the main controller of the flow rules and then the activation of the installer) Flow rules and allocation of additional load to it.



(a)



(b)

Fig. 11. (a) The amount of incoming traffic in the proposed method before the TCP_SYN flooding attack, during the attack and after it and (b) the round trip time (RTT) of customer requests.

VI. Comparing the Effectiveness of the Proposed Solution to those Already in Use

This section compares the effectiveness of the proposed strategy with four different ways of preventing denial-of-service attacks. The four options are as follows: Ryu controller [40] without any protection method, Ryu controller [41] with MLFQ mechanism, Ryu controller [18] with Flood Defender mechanism, and Ryu controller [40] with FMD-ARA system. The method used by the researchers in [19] provides the basis for this comparison. Fig. 12 (implemented in the Mininet 2.2.2 emulator) depicts the network structure utilized to conduct this comparison [36], [37].

The control layer network in this experiment was out-of-band (a different network from the data layer network), in contrast to the prior tests where the control layer had an in-band network (using the data layer network to communicate with switches and other controllers).

As seen in Fig. 12, in the scenario taken into consideration for the test, 2h interacts with 3h, and 7h communicates with 8h while attackers 1h and 5h perform a UDP flood attack and steadily speed up this attack. Response Request Time (RRT) is measured at about 2h with 3h and 7h with 8h as a standard to evaluate the effectiveness of each solution. RRT is the average response time from the controller and demonstrates the efficiency of the controller in producing network connections. Yes, that is doable. The response request time in Fig. 13 is in relation to 2h and 3h demonstrates the effectiveness of the approach against the denial-of-service attack. The coordinating controller speeds up the attack until it reaches 1000 packets per second (PPS), at which point it activates the sub-controller of the flow rules setter and sends the traffic produced by the UDP flood attack to it. As demonstrated in Fig. 13, this operation has significantly lowered the average response time to legitimate and safe requests to build the flow by reducing the traffic load on the main controller that establishes the flow rules. Fig. 13's findings, derived from the researchers' article from [20], indicate that while the efficiency of the other four techniques improved as the attack speed increased, so did their average response times. The control level is resistant to denial-of-service attacks and performs better thanks to its unique structure (the presence of sub-controllers that determine the flow rules in the main and coordinating controllers).

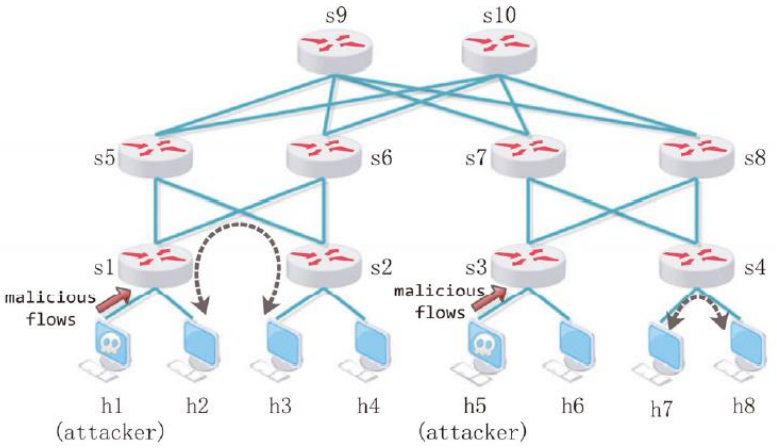


Fig. 12. Network architecture consisting of Open_Flow switches to evaluate the proposed efficiency against denial-of-service attacks.

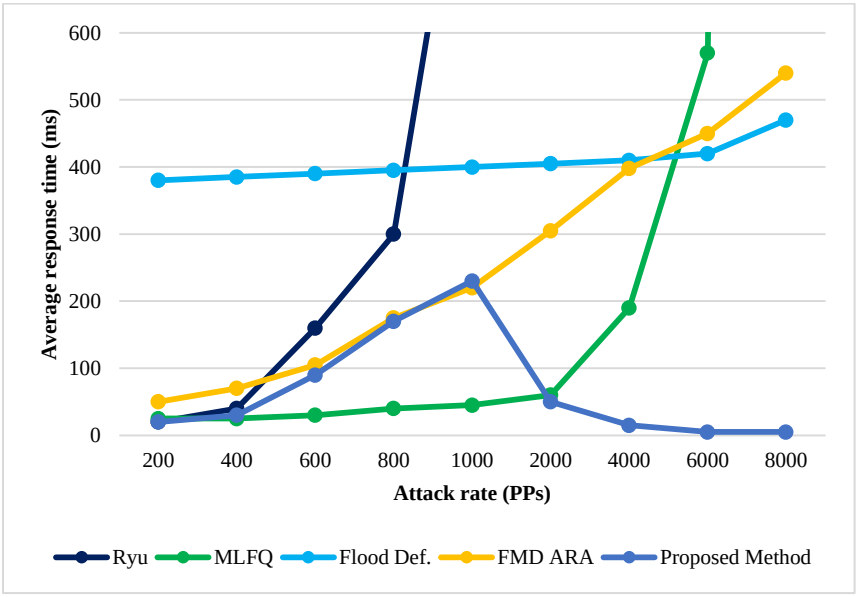


Fig. 13. For current approaches, the typical reaction time is between 2 and 3 hours.

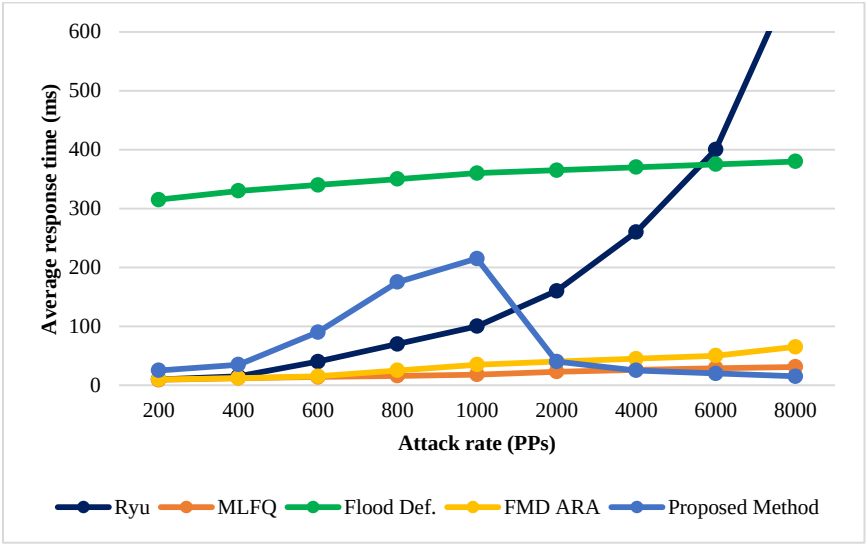


Fig. 14. Based on current techniques, the average reaction time in communication is between 7 and 8.

Fig. 14 illustrates the typical response time for communications between 7 and 8 hours. In contrast to the previous instance, a connection is established between two hosts not directly linked to the switch targeted in the attack. In this instance, the recommended efficiency is not much different from the prior state. Among the four solutions, with the exception of Flood Defender, which relies on a pre-processing system and must temporarily store each new Open_Flow request in a packet buffer before reacting to it, all show better performance than the previous setup. This situation results from the fact that the main factor causing an increase in the reaction time delay in the three solutions, MLFQ, Ryu, and FMD-ARA, is the presence of congestion in the communication channel of the switch with the controller. If, as proposed, the switch's communication channel with the main controller of the flow rules becomes congested and the main controller becomes overloaded, increasing the response time delay, the additional load that caused the congestion is sent through a different communication channel to the main controller. The set of flow rules acts as a guide for a sub-controller. The coordinating controller activates the sub-controller of the flow rules when it notices that the main controller of the flow rules is overloaded. This not only overloads the main controller but also creates a separate communication channel between the sub-controller and the switch. To avoid clogging up the main controller's communication channel with the switch.

The key findings of the study highlight the effectiveness of a novel approach to enhance the security and resilience of software-defined networks (SDNs) against denial-of-service (DoS) attacks. The introduced "control box" solution, consisting of coordinating, main, and sub-controllers, successfully manages and distributes incoming traffic during DoS attacks, reducing response times and maintaining network availability. Experimental validation, using various DoS attack scenarios, demonstrates the method's superiority over existing solutions. Furthermore, comparisons with alternative methods, including Ryu controllers with various mechanisms, consistently show that the proposed approach outperforms them. While the solution offers promising results, the study acknowledges the need for further research to address resource consumption and scalability issues, paving the way for more resource-efficient and scalable network security solutions in the future.

VII. Conclusion

In this article, a new solution for the control layer has been developed with the aim of increasing access to software-based networks by increasing the security of its control layer. The control box is the main part of the proposed method. This control box consists of a main controller that adjusts the flow rules, one or more sub-controllers that adjust the flow rules, and a coordinating controller. The task of the coordinator controller is to monitor the controllers that define the flow rules and monitor the network by collecting statistical data from the Open_Flow switches. Controllers that install flow rules are also responsible for installing flow rules. Based on network information base (NIB) data obtained from the coordinating controller, these flow rules are generated by network programs running on the controllers that install the flow rules. Based on the traffic load entering the network, the coordinating controller also coordinates the operation of the main and sub-controllers of the flow rules. The high tolerance of the proposed method for attacks that lead to the saturation of the control plane by the data plane of SDN networks is due to the redundancy of flow rule set controllers in the method. In summary, the importance of the proposed solution to increase the security and resilience of Software Defined Networks (SDN) against Denial of Service (DoS) attacks is emphasized. They highlight the unique contribution of their "control box" approach, which effectively manages and distributes incoming traffic during DoS attacks, thereby reducing response time and maintaining network availability. The following are the limitations and future works.

The proposed solution, while effective in reducing DoS attacks, requires the deployment of additional resources, which can lead to increased energy consumption and cost. This limitation highlights the need for further optimization to reduce the resource overhead associated with the approach. This article also mentions a limitation on the number of available switch ports for adding sub-controllers to a flow rules installation. This scalability limitation needs to be addressed to ensure that this approach can be applied to larger and more complex SDN environments.

VIII. Future Work

1) Resource efficiency: Future research should focus on optimizing the proposed solution to reduce resource requirements, making it more resource efficient and environmentally sustainable.

2) Scalability solutions: Addressing scalability limitations is a priority. Researchers could explore ways to scale the approach to accommodate larger SDN deployments, possibly through innovations in controller communication and load distribution techniques.

3) Security extensions: Expanding research to cover a wider range of cyber threats and security challenges in SDNs would be valuable. This could include exploring ways to increase security against other types of attacks beyond DoS, ensuring comprehensive network protection.

References

[1] 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, vol. 25, pp. 1338–1347, 2020.

[2] H. Farhady, H. Lee, and A. Nakao, “Software-defined networking: A survey,” Computer Networks, vol. 81, pp. 79–95, 2015.

[3] M. A. Aladaileh, M. Anbar, I. H. Hasbullah, Y.-W. Chong, and Y. K. Sanjalawe, “Detection techniques of distributed denial-of-service attacks on software-defined networking controller–a review,” IEEE Access, vol. 8, pp. 143985–143995, 2020.

[4] E. Khezri and E. Zeinali, “A review on highway routing protocols in vehicular ad hoc networks,” SN Comput Sci, vol. 2, pp. 1–22, 2021.

[5] A. F. Abdullah, F. M. Salem, A. Tammam, and M. H. A. Azeem, “Performance analysis and evaluation of software defined networking controllers against denial-of-service attacks,” in Journal of Physics: Conference Series, IOP Publishing, 2020, p. 012007.

[6] W. Li, W. Meng, and L. F. Kwok, “A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures,”

Journal of Network and Computer Applications, vol. 68, pp. 126–139, 2016.

[7] M. F. Hyder and T. Fatima, "Towards crossfire distributed denial-of-service attack protection using intent-based moving target defense over software-defined networking," IEEE Access, vol. 9, pp. 112792–112804, 2021.

[8] J. Benabbou, K. Elbaamrani, and N. Idboufker, "Security in OpenFlow-based SDN, opportunities and challenges," Photonic Network Communications, vol. 37, pp. 1–23, 2019.

[9] M. Sakthivel, R. Kamalraj, S. Sivanantham, and V. Krishnamoorthy, "An Analysis of Machine Learning Depend on Q-MIND for Defencing The Distributed Denial of Service Attack on Software Defined Network.," International Journal of Early Childhood Special Education, vol. 14, no. 5, 2022.

[10] Cao, Y., Xu, N., Wang, H., Zhao, X., & Ahmad, A. M. (2023). Neural networks-based adaptive tracking control for full-state constrained switched nonlinear systems with periodic disturbances and actuator saturation. International Journal of Systems Science, 54(14), 2689-2704.

[11] B. Mladenov and G. Iliev, "Optimal software-defined network topology for distributed denial-of-service attack mitigation," Bulletin of Electrical Engineering and Informatics, vol. 9, no. 6, pp. 2588–2594, 2020.

[12] E. Khezri, E. Zeinali, and H. Sargolzaey, "SGHRP: Secure Greedy Highway Routing Protocol with authentication and increased privacy in vehicular ad hoc networks," PLoS One, vol. 18, no. 4, p. e0282031, 2023.

[13] Wang, T., Zhang, L., Xu, N., & Alharbi, K. H. (2023). Adaptive critic learning for approximate optimal event-triggered tracking control of nonlinear systems with prescribed performances. International Journal of Control, 1-15.

[14] H. Wang, L. Xu, and G. Gu, "OF-GUARD: A DoS attack prevention extension in software-defined networks," The Open Network Summit (ONS), no. 2014, 2014.

[15] D. Agnew, S. Boamah, R. Mathieu, A. Cooper, J. McNair, and A. Bretas, "Distributed Software-Defined Network Architecture for Smart Grid Resilience to Denial-of-Service Attacks," arXiv preprint arXiv:2212.09990, 2022.

[16] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran, "Lineswitch: Tackling control plane saturation attacks in software-defined networking," IEEE/ACM Transactions on Networking, vol. 25, no. 2, pp. 1206–1219, 2016.

[17] M. Trik, A. M. N. G. Molk, F. Ghasemi, and P. Pouryeganeh, "A hybrid selection strategy based on traffic analysis for improving performance in networks on chip," J Sens, vol. 2022, 2022.

[18] Yue, S., Niu, B., Wang, H., Zhang, L., & Ahmad, A. M. (2023). Hierarchical sliding mode-based adaptive fuzzy control for uncertain switched under-actuated nonlinear systems with input saturation and dead-zone. Robotic Intelligence and Automation, 43(5), 523-536.

[19] K. Chen, A. R. Junuthula, I. K. Siddhrau, Y. Xu, and H. J. Chao, "SDNShield: Towards more comprehensive defense against DDoS attacks on SDN control plane," in 2016 IEEE conference on communications and network security (CNS), IEEE, 2016, pp. 28–36.

[20] T. Semong et al., "Intelligent load balancing techniques in software defined networks: A survey," Electronics (Basel), vol. 9, no. 7, p. 1091, 2020.

[21] Zhao, H., Wang, H., Xu, N., Zhao, X., & Sharaf, S. (2023). Fuzzy approximation-based optimal consensus control for nonlinear multiagent systems via adaptive dynamic programming. Neurocomputing, 553, 126529.

[22] A. K. A. Al-Mashadani and M. Ilyas, "Distributed Denial of Service Attack Alleviated and Detected by Using Mininet and Software Defined Network," Webology, vol. 19, no. 1, pp. 4129–4144, 2022.

[23] Zhang, H., Zou, Q., Ju, Y., Song, C., & Chen, D. (2022). Distance-based support vector machine to predict DNA N6-methyladenine modification. Current Bioinformatics, 17(5), 473-482.

[24] O. Polat and H. Polat, "An intelligent software defined networking controller component to detect and mitigate denial-of-service attacks," Journal of Information and Communication Technology, vol. 20, no. 1, pp. 57–81, 2021.

[25] Cao, C., Wang, J., Kwok, D., Cui, F., Zhang, Z., Zhao, D., ... & Zou, Q. (2022). webTWAS: a resource for disease candidate susceptibility genes identified by transcriptome-wide association study. Nucleic acids research, 50(D1), D1123-D1130.

[26] S. Sharathkumar and N. Sreenath, "Distributed Clustering based Denial of Service Attack Prevention Mechanism using a Fault Tolerant Self Configured Controller in a Software Defined Network," 2023.

[27] Arefanjazi, H., Ataei, M., Ekramian, M., & Montazeri, A. (2023). A robust distributed observer design for Lipschitz nonlinear systems with time-varying switching topology. Journal of the Franklin Institute, 360(14), 10728-10744.

[28] E. Khezri, E. Zeinali, and H. Sargolzaey, "A novel highway routing protocol in vehicular ad hoc networks using VMaSC-LTE and DBA-MAC protocols," Wirel Commun Mob Comput, vol. 2022, 2022.

[29] Wang, Z., Jin, Z., Yang, Z., Zhao, W., & Trik, M. (2023). Increasing efficiency for routing in internet of things using Binary Gray Wolf Optimization and fuzzy logic. Journal of King Saud University-Computer and Information Sciences, 35(9), 101732.

[30] M. Samiei, A. Hassani, S. Sarspy, I. E. Komari, M. Trik, and F. Hassanpour, "Classification of skin cancer stages using a AHP fuzzy technique within the context of big data healthcare," J Cancer Res Clin Oncol, pp. 1–15, 2023.

[31] J. Sun, Y. Zhang, and M. Trik, "PBPHS: a profile-based predictive handover strategy for 5G networks," Cybern Syst, pp. 1–22, 2022.

[32] M. Trik, H. Akhavan, A. M. Bidgoli, A. M. N. G. Molk, H. Vashani, and S. P. Mozaffari, "A new adaptive selection strategy for reducing latency in networks on chip," Integration, vol. 89, pp. 9–24, 2023.

[33] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui, "FloodDefender: Protecting data and control plane resources under SDN-aimed DoS attacks," in IEEE INFOCOM 2017-IEEE Conference on Computer Communications, IEEE, 2017, pp. 1–9.

[34] P. Wu, L. Yao, C. Lin, G. Wu, and M. S. Obaidat, "Fmd: A DoS mitigation scheme based on flow migration in software-defined networking," International Journal of Communication Systems, vol. 31, no. 9, p. e3543, 2018.

[35] Y. Wang, T. Hu, G. Tang, J. Xie, and J. Lu, "SGS: Safe-guard scheme for protecting control plane against DDoS attacks in software-defined networking," IEEE Access, vol. 7, pp. 34699–34710, 2019.

[36] Y. Han, J.-H. Yoo, and J. W.-K. Hong, "Poisson shot-noise process based flow-level traffic matrix generation for data center networks," in 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), IEEE, 2015, pp. 450–457.

[37] D. Mokhlesi Ghanevati, E. Khorami, B. Boukani, and M. Trik, "Improve replica placement in content distribution networks with hybrid technique," Journal of Advances in Computer Research, vol. 11, no. 1, pp. 87–99, 2020.

[38] M. Trik, S. P. Mozaffari, and A. M. Bidgoli, "Providing an adaptive routing along with a hybrid selection strategy to increase efficiency in NoC-based neuromorphic systems," Comput Intell Neurosci, vol. 2021, 2021.

[39] B. Yu, G. Yang, and C. Yoo, "Comprehensive prediction models of control traffic for SDN controllers," in 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), IEEE, 2018, pp. 262–266.

[40] Q. Gao, "Recommended System Optimization in Social Networks based on Cooperative Filter with Deep MVR Algorithm," 2022.

[41] Y. Ashgevari and M. Karami, "Study of Atmospheric Discharge Effects in Distribution Networks with a Novel Residential Buildings Protection Approach," Advances in Engineering and Intelligence Systems, vol. 2, no. 02, 2023.

[42] Karthika, P., & Karmel, A. (2023). Review on distributed denial-of-service attack detection in software defined network. International Journal of Wireless and Mobile Computing, 25(2), 128-146.