Hindawi
Security and Communication Networks
Volume 2022, Article ID 1608689, 16 pages
https://doi.org/10.1155/2022/1608689

Research Article

BSD-Guard: A Collaborative Blockchain-Based Approach for Detection and Mitigation of SDN-Targeted DDoS Attacks

Shanqing Jiang,[1,2,3] **Lin Yang**,[2] **Xianming Gao**,[2] **Yuyang Zhou**,[1,3,4] **Tao Feng**,[2] **Yanbo Song**,[5] **Kexian Liu**,[6] **and Guang Cheng**[1,3,4]

[1]*School of Cyber Science and Engineering, Southeast University, Nanjing, China*
[2]*National Key Laboratory of Science and Technology on Information System Security, Institute of System Engineering, PLA Academy of Military Science, Beijing, China*
[3]*Purple Mountain Laboratories, Nanjing, China*
[4]*Jiangsu Province Engineering Research Center of Security for Ubiquitous Network, Nanjing, China*
[5]*State Key Laboratory on Integrated Services Networks, Xidian University, Xi'an, China*
[6]*School of Computing Science, Beijing University of Posts and Telecommunications, Beijing, China*

Correspondence should be addressed to Guang Cheng; gcheng@njnet.edu.cn

Received 3 January 2022; Revised 16 February 2022; Accepted 24 February 2022; Published 12 April 2022

Academic Editor: Yuling Chen

Software-Defined Networking (SDN) enhances the flexibility and programmability of networks by separating control plane and data plane. The logically centralized control mechanism makes the control plane vulnerable in both single and multiple controller scenarios. Malicious third parties can exploit vulnerabilities of reactive forwarding mode to launch distributed denial-of-service (DDoS) attacks against SDN controllers. Unfortunately, existing DoS/DDoS solutions under single controller can not afford effective performance under multiple controllers due to the absence of cooperative detection and mitigation. To solve the above problem, we propose a blockchain-based SDN-targeted DDoS defense framework (BSD-Guard) that can provide cooperative detection and mitigation mechanism to protect SDN controllers. BSD-Guard introduces a blockchain-based secure middle plane between control plane and data plane. The secure middle plane calculates the suspect rate of new flows based on the collected packets' information and reports suspect lists to blockchain for immutably storing and sharing. Besides, the smart contract deployed on blockchain in advance constitutes collaborative defense strategies based on the suspect lists reported from multiple SDN domains. When receiving defense strategies, the secure middle plane converts them to specific flow table actions and installs actions into relevant switches. The experimental results indicate that BSD-Guard can efficiently detect DoS/DDoS attacks in multiple controllers scenario and issue precise defensive strategies near the source of attack by identifying the attack path.

1. Introduction

Software-Defined Network (SDN) is a novel network architecture designed to help network operators better manage infrastructures. The separation of control and data planes and logical centralized control bring network with high availability and programmability [1]. The logical centralized control plane conducts the behaviors of data plane via southbound protocols, in which the OpenFlow has developed as a typical and widely used southbound protocol. OpenFlow allows reactive mode for installing forwarding rules, which has greatly simplified the rules configuration and policy deployment. The reactive mode arranges the table-missed packets to be encapsulated into *packet_in* message and reports to control plane for generating new forwarding rules. Because of the limited computational capacity, controller may discard normal requested traffics when the number of table-missed packets exceeds controller's processing capacity. This vulnerability can easily be exploited by attackers to launch resource-exhausting attacks against SDN controllers. Among them, the DoS/DDoS attack against SDN controllers has become a critical problem

[2] in recent years. Since controller determines the computation of end-to-end transmission path, when controllers suffer DoS/DDoS attack, it will disturb the normal message forwarding in the control domain, which may further cause the whole network to be disrupted. In summary, DoS/DDoS attack has become a serious security risk, affecting SDN architecture for the following reasons. First, the interaction mechanism based on OpenFlow protocol makes the controller a target of malicious attackers [3]. Second, the attacks are inexpensive and convenient to be implemented through launching forged requesting messages at hijacked hosts. Third, the attacker cannot be accurately traced and legitimate messages may be misdiscarded during defense process.

Traditional DDoS detecting and mitigating methods are often divided into four levels: attack detection, load balancing, traffic filtering, and traffic analysis [4]. (1) Attack detection is to identify DDoS traffic from normal traffic. Common detection methods are mainly based on message statistics and machine learning, which need to be ensured in real-time and accuracy of detection. (2) Load balancing relieves the storage and computing pressure of the victimized target by rerouting or traffic migrating and provides a brief resistance to sudden abnormal traffic within the tolerable range of the load balancing module. (3) Traffic filtering discards DDoS attack traffic by identifying the abnormal traffic characteristics, with the goal of improving the accuracy of identification and ensuring that normal traffic can be forwarded normally by network devices. (4) Traffic analysis aims to identify the attacker's intended behavior and trace the source of the attack by analyzing the collected attack traffic data.

There are two main shortcomings in current research on DoS/DDoS defense for SDN controllers. First, the misclassification of detection may cause the first packet of legitimate traffic to be discarded. Unlike the packet retransmission mechanism in traditional networks, in the SDN environment, the first packet being dropped will lead to more serious consequences. The subsequent messages of the normal traffic will not be processed, and the first packet request needs to be initiated again until controller issues correct forwarding rules. This blocking process affects the communication of normal service in data plane, increasing the burden of controllers and southbound channel. Second, the traditional DDoS detection and mitigation solutions for single controller can not be directly applied to multiple controllers scenario. In multiple controllers scenario, it is difficult to detect large-scale and distributed DDoS attacks and implement effective defense measures. Although the centralized control of SDN provides convenience for the monitoring of network status, for large-scale DDoS attacks, the centralized detection method is redundant and expensive, and the threat situation of the whole network has not been utilized reasonably. The east-west interface is used to maintain communication between controllers, which has not been standardized and not enough to support collaborative awareness and defensive decision. The collaborative protection mechanisms against DDoS in multicontroller scenario have also been introduced in recent years. In [5], a collaborative DDoS defense system can reroute crashing traffic to other domains for filtering. In [6], the Redis Simple Message Queue (RSMQ) approach was used to collaboratively share detection and mitigation rules among multiple controllers. The latest research begins to seek cooperation with blockchain, and its decentralized features bring convenience to collaborative detection and mitigation. Researches [7–9] proposed a blockchain-based SDN framework to share threat information between multiple controllers. However, the smart contract was only used to share risky IP address and the time consumption of generating new block in Ethereum reaches flagrant 14 seconds [10].

In this paper, we propose BSD-Guard, a collaborative and elastic blockchain-based detection and defense system to protect SDN against controller targeted DDoS attacks. BSD-Guard stands between control plane and data plane, consisting of blockchain-based secure middle plane. In the detection stage, the secure middle plane collects statistics information about packets and ports from edge switches. Then the suspect lists calculated by detection algorithm are shared on the blockchain that can not be tampered by malicious attackers. And a global threat situation can be generated by cooperating smart contracts among multiple SDN domains. In the mitigation stage, the defensive strategies generated on blockchain can be installed into the edge switches by secure middle plane. Finally, the attacking packets can be discarded at source switches and benign packets can be forwarded correctly. And the SDN controllers can maintain a low level of CPU utilization when DDoS attack occurs. Our main technical contributions are as follows:

(i) *Novel Framework.* We propose a novel detection and defense framework for protecting SDN controllers from DDoS attacks. The secure middle plane can perform as a proxy for a controller to detect and discard abnormal traffics. The blockchain becomes a platform for information sharing and defense policies scheduling between multiple controllers.

(ii) *Fine-Grained Detection.* We present an entropy based suspect rate calculation method for fine-grained DDoS detection. The blacklist and graylist are generated by the type of forged addresses and its suspect rate. The fine-grained suspect list is beneficial for the subsequent development of precise defense strategies.

(iii) *Collaborative Mitigation.* The detection and mitigation smart contracts deployed on the blockchain can collaborate with threat information reported by multiple secure middle planes. It can accurately identify the scale and the path of DDoS attacks and develop targeted defense strategies.

The rest of this paper will be organized as follows: Section 2 introduces the related works of existing detection and defense of DDoS attacks in the SDN environment. In Section 3, we present the problem statement about the adversary model and attack scenario. In Section 4, we introduce the detailed designs of the BSD-Guard system. Section 5 is the implementation and experimental evaluation of BSD-Guard. Finally, we make the summary of this paper in Section 6.

2. Related Works

2.1. Detection and Mitigation Methods under Single Controller. The DDoS attack on SDN controller has become a serious problem that can affect cloud environments and industrial production platforms that run over SDN networks, which will cause severe network security incidents. To solve this problem, researchers have proposed a large number of detection and defense solutions under single controller scenario. The DDoS detection solutions can be categorized into statistics-based schemes and machine learning based schemes under single controller.

Firstly, the statistics-based detection and mitigation scheme identifies DDoS attack by extracting statistical features of data plane traffic. In [11, 12], researchers identified DDoS attack traffic by detecting the rate and characteristic value of *packet_in* messages. You et al. [11] deployed the traffic collection module on the controller to collect, parse, and extract feature information of *packet_in* and calculate the rate of *packet_in*, entropy value of destination IP address, and port number. Huang et al. [12] predicted the number of *packet_in* in the next cycle by Taylor's formula; the detection module will be activated when the number exceeds the threshold. Then the characteristic values of *packet_in* were extracted for entropy calculation and determined whether there is a DDoS attack according to the entropy value. In [13–16], researchers have also advanced statistical analysis methods of flow tables to detect DDoS attacks. Fouladi et al. [13] detected DDoS by time series analysis of flow tables and determined the aggregation of traffic in network using feature information of destination IP address. Through the extraction of flow table features, the source of attack can also be traced back. Hassan et al. [16] used a lightweight approach to detect and defend against DDoS in SDN based on *Tsallis* entropy, which is able to detect DDoS at early stages, and the proposed dynamic threshold mechanism allows the detection method to adapt to dynamically changing network conditions. There also exist some studies that extract statistical feature from *sFlow* (Sampled Flow) to identify DDoS traffics [17–19]. Lawal et al. [17] obtained *CounterSample* and *FlowSample* messages by *sFlow* sampling, extracting traffic features, calculating traffic rate, and determining the presence of DDoS attack in real time by setting thresholds. Kumar et al. [18] obtained the feature values of traffic by *sFlow* and used machine learning for DDoS detection. Lu et al. [19] employed *sFlow* to obtain packets rate and aggregation of destination IP address in SDN network and jointly determined whether suspicious traffic occurred in SDN. In [20], Chen et al. proposed SDNShield, a three-stage overload control scheme for mitigating DDoS in SDN based on NFV technologies. The simulation results showed that SDNShield can achieve resilient performance against brute-force DDoS attacks and maintain excellent flow service quality at the same time.

Secondly, the machine learning based detection and mitigation scheme identifies and classifies DDoS traffics by various machine learning methods. Mehr and Ramamurthy [21] used the support vector machine to detect DDoS attacks and install defense flow table entries to the switch, which reduced the impact of DDoS attacks on Ryu controllers by 36%. Considering the imbalance of traffic distribution, Cui et al. [22] introduced clustering algorithms such as the k-means to detect malicious traffics. In addition, the authors used *packet_in* message register to filter malicious traffic and evaluate the scheme in terms of detection accuracy, defense effectiveness, and communication latency. Some other researchers proposed hybrid machine learning approaches. Deepa et al. [23] proposed a model of hybrid machine learning with support vector machines and self-organizing mappings, which can effectively protect the SDN controllers to work properly when DDoS attacks occur. Nugraha and Murthy [24] proposed a hybrid Convolutional Neural Network-Long-Short Term Memory (CNN-LSTM) model to detect slow DDoS attacks in SDN networks, and experiments showed that the method achieved 99% accuracy in the considered performance metrics. Xu et al. [25] proposed an efficient and accurate DDoS detection method based on SDN cloud edge collaboration. The method used an entropy approach to select ideal SOM mappings and classify SOM neurons, and then KD-trees were used to identify traffics at a finer granularity, which improved the accuracy of DDoS detection. Ujjan et al. [26] proposed a DDoS detection method based on adaptive polling sampling of *sFlow* and deep learning models. Adaptive polling sampling of *sFlow* was used in the data plane to reduce the switch's overhead. Snort IDS and SAE deep learning models were deployed in the control plane to improve the accuracy of detection. The authors quantitatively investigated the trade-off between the accuracy of attack detection and resource overhead. Luong et al. [27] proposed a DDoS detection model in SDN based on machine learning and deep neural networks, and authors compared the model with decision trees and random forest models. The results showed that complex DDoS detection systems do not necessarily produce more accurate results than simple ones.

2.2. Cooperative Defense under Multicontrollers. DDoS attacks are complex and varied in the actual network environment. Attackers often launch DDoS traffics from remote locations to one target by hijacking a large number of puppet hosts or exploiting vulnerabilities in existing communication protocols. The DDoS attack under multiple controllers network has become one of the most difficult threats in SDN environment. This is because traffics within disparate controller domains often exhibit different characteristics, and a more concentrated aggregation of abnormal traffic usually emerges in the victim's domain. The domain is defined as the partial network managed by one controller. Usually, the DDoS attack has already been carried out in the source domain when detection mechanism was triggered in the targeted domain, which will leave the defender quite limited time to respond and defense. Therefore, it is necessary to share threat information among multiple domains to identify and intercept abnormal traffic during the initiation and dissemination phase, which will save more time for protecting the target controller.

There exist several studies on cooperative defending against DDoS attacks. The IETF is proposing an ongoing protocol called DOTS (DDoS Open Threat Signal) [28], which will mitigate DDoS attacks by an intradomain and cross-domain collaborative solution. The servers and clients of DOTS are required to broadcast blacklists or whitelists addresses. When detecting attacks, the client requests mitigation services from the server responsible for cross-domain communication and coordination. However, the DOTS is still faced with implementation complexity to support different types of communication in distributed and centralized architectures. A similar approach is presented in [29]. The authors employ an advertising protocol based on FLEX (Flow-based Event eXchange) format to simplify the deployment and collaboration between domains. This protocol supports realizing the situational awareness of the current threat posture, pooling expertise and resources, and facilitating automated defense against persistent cyberattacks. However, the deployments of above solutions are complex since they need to create or modify protocols for distributed network architectures. Instead, these collaboration requirements can be met by the natural characteristics of SDN, blockchain, and smart contract, thus avoiding the complexity of deployment and adoption of new protocols.

Blockchain and smart contracts have shown their unique advantages in the area of collaborative threat detection and defense for SDN and IoT. Javaid et al. [30] introduced a smart contracts-enabled IoT device communication framework using Ethereum, a blockchain variant to replace the traditional centralized IoT infrastructure. Smart contracts are required for IoT devices accessing the network. And trusted or untrusted devices can be distinguished by the proposed system. Shao et al. [31] proposed a blockchain-based SDN security system model and a consensus algorithm SPBFT to improve the security and consensus efficiency of the SDN control plane. The smart contracts periodically check the status of controller to detect DDoS attack. Abou et al. [7, 8] designed a collaborative distributed DDoS mitigation framework based on blockchain. The framework utilized smart contracts to transfer attack information between SDN multiple domains to reduce the huge cost of forwarding useless packets across multiple domains. Extensive experiments on both private and public networks (Ganache simulator, Ropsten test network) show that Cochain-SC achieves versatility, security, efficiency, and cost-effectiveness. In [9], a blockchain-based SDN architecture was proposed to advertise whitelisted or blacklisted IP addresses to defend against DDoS attacks, enabling the execution of defense rules across multiple domains. However, the advantages of blockchain and smart contracts have not been fully exploited in existing research. As an excellent distributed collaborative platform, blockchain should not be limited to sharing blacklisted and whitelisted IP addresses, but also sharing the data plane traffic characteristics that are originally opaque between multiple controllers. This method will allow the characteristics of DDoS attack to be jointly discovered at an earlier stage. And smart contracts can also be used as triggers for issuing defense policies automatically.

3. Problem Statement

In this section, we first introduce the workflow of handling normal traffic in SDN networks. Then we present the adversary model of SDN-targeted DoS and DDoS attacks. Finally, we state the challenges of detecting and mitigating the DDoS attacks in multiple SDN controllers networks and the basic principles that should be kept in the process.

3.1. SDN Workflow.

OpenFlow has become a widely used standard southbound interface protocol that specifies the pipeline for switches to handle packets and the types of messages between the data plane and the control plane. OpenFlow supports both proactive and reactive approaches to install flow forwarding rules. In the proactive mode, the controller preregisters forwarding rules on switches to handle incoming packets. In the reactive mode, when an OpenFlow switch receives several new incoming packets, it will process each packet by following steps with the FIFO (first input first output) manner [32], as shown in Figure 1.

(1) The OpenFlow Agent (OFA) traverses its flow table to find if there exist flow table entries that match the header of the new-coming packet. If a match occurs, the switch will process the packet according to the action field of the flow table entry, such as forwarding. Otherwise, the switch treats the packet as *table-miss* by caching it into the buffer area, encapsulating its header into *packet_in* message, and sending to the controller. If the buffer is full, the entire packet will be encapsulated into a *packet_in* message (Steps A, B, and C).

(2) The SDN controller receives the *packet_in* message and calculates forwarding policy based on the global networking view and applications' intention. The forwarding action will be encapsulated into a *packet_out* message and sent back to the switch (Steps D, E).

(3) The OFA receives the *packet_out* message and installs the entries into the flow table and then handles the buffered table-miss packet based on the instruction in *packet_out* (Steps F, G).

(4) When the further packets with the same header arrive within the survival time of flow table entries, the OFA can deal with these packets according to the "match" and "action" instruction with linear rate.

This reactive flow table installation method enables a flexible way to control network traffic, which is the core principle of SDN's control and forwarding decoupling. It is widely used in most OpenFlow scenarios. However, due to the limited processing capacity of hardware and software, this method has also become a source of resource-consuming threats in sSDN networks.

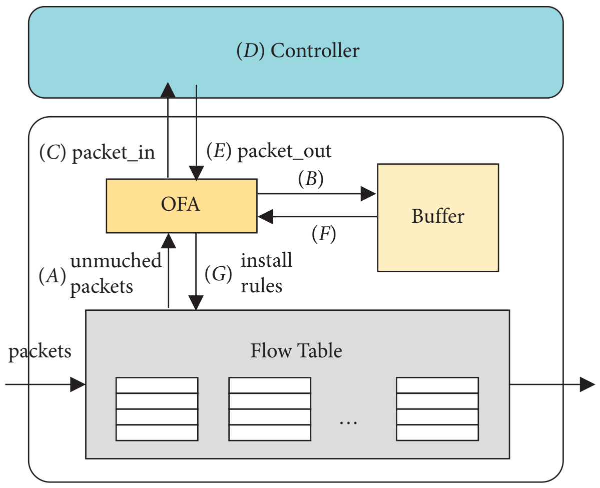


Figure 1: The workflow of OpenFlow reactive method.

3.2. Adversary Model. The SDN-targeted DoS/DDoS attacks are different from DoS/DDoS attacks in traditional networks. The reactive method of SDN's workflow can be exploited by tricky attackers to launch DoS/DDoS attacks. When encountering table-missing packets, the OpenFlow switch must initiate a *packet_in* message to controller for requiring forwarding actions. By sending a large number of forged address packets, attackers can stimulate switch with abundant meaningless *packet_in* messages to controller, which will result in excessive consumption of CPU and storage resources, meanwhile causing switch's buffer overflowing and control channel blocking. When an attacker injects a large number of forged new packets into multiple switches in the data plane at the same time, the controller will suffer a more serious DDoS attack, as shown in Figure 2.

We demonstrate the damage of SDN-targeted DDoS attacks with a group of experiments. We set up an experimental environment consisting of one ONOS controller and 10 OpenVSwitch in mininet (with linear topology). In the first DoS group, forged address packets are injected through one switch with 20 pps to 200 pps. In the second DDoS group, forged address packets are injected through 10 switches with 20 pps to 200 pps concurrently. The result in Figure 3 shows that the CPU utilization rate of the ONOS controller in the DDoS group increases higher than DoS. From the attacker's view, the DoS attack with 200 pps and DDoS attack with 20 pps $*$ 10 will stimulate the same number of new *packet_in* message theoretically. However, we can find in Figure 3 that when the total forged packets rate is 200 pps, the CPU utilization rate in the DDoS group (attack intensity = 20 pps $*$ 10, CPU = 44.41%) is much higher than that in the DoS group (attack intensity = 200 pps $*$ 1, CPU = 11.6%). Therefore, we can conclude that a DDoS attack launched from multiple switches has more serious harm to the control plane than a DoS attack when attackers equip limited attack resources. Besides, we also make another interesting comparison. We disconnect the links between 10 switches and perform DDoS attack again, and the result shows that the CPU consumption of controller decreases by 20% averagely. By capturing packets and analyzing, the truth is that when switches are linked with each other, the forged new packets of DDoS can be broadcasted among switches, which makes the number of *packet_in* messages reported to controller be amplified. Above all, the control plane will be more vulnerable to DDoS attacks under a distributed network scenario.

3.3. Scenario and Challenges. The multiple SDN controllers environment is displayed in Figure 4. In the multiple controllers' scenario, the controller targeted DDoS attacks could be launched from remote data plane managed by other controllers. To implement DDoS attacks more stealthily, the tricky attackers often initiate attacking packets from the neighboring domains of the victim controller, which greatly increases the difficulty of detection and mitigation. And the defensive actions performed in the victim's domain will not be effective to mitigate such attacks. There have been a lot of previous researches on how to detect DDoS attacks between multiple controllers [7, 33]. However, the fine-grained threat information can not be shared collaboratively across multiple controllers, and a complete set of defense schemes has not been developed.

Although there has been a lot of valuable researches on DDoS detection and defense for protecting controllers, two key issues remain unresolved. First, existing detection processes require data plane traffic and network state information to be reported to the centralized controller, which will greatly increase the burden of controller and southbound channel. Second, in the multiple controllers' scenario, threat information within a domain can only be mastered by the internal controller. However, the threat state perceived in one domain is only local information, which cannot form the most effective defense plan. Meanwhile, the interaction among east-west interface will consume the resources for synchronizing state information. Although the author in [7] proposed a blockchain-based framework Cochain-SC to facilitate the collaboration for smart contract-based intradomain DDoS mitigation, the sharing information between intradomains is limited to blacklisted IPs.

Therefore, in the multiple controllers' DDoS defense scenario, collaborative integration of threat information and network resources between north-south and east-west needs to be considered simultaneously. From the north-south view, the threat situation of DDoS traffic in the data plane should not be completely reported to the control plane, which can reserve the valuable computing resources of controller and avoid single-point failure. From the east-west view, the more fine-grained network threat information from multiple control domains should be shared for identifying attack scenarios and tracing attacker more precisely and preventing DDoS traffic from spreading among multiple controllers.

4. System Overview

We design a system named BSD-Guard, which can detect and mitigate SDN-targeted DDoS attacks among multiple controllers. This system can calculate suspect lists according to traffic statistical information from multiple controllers. The

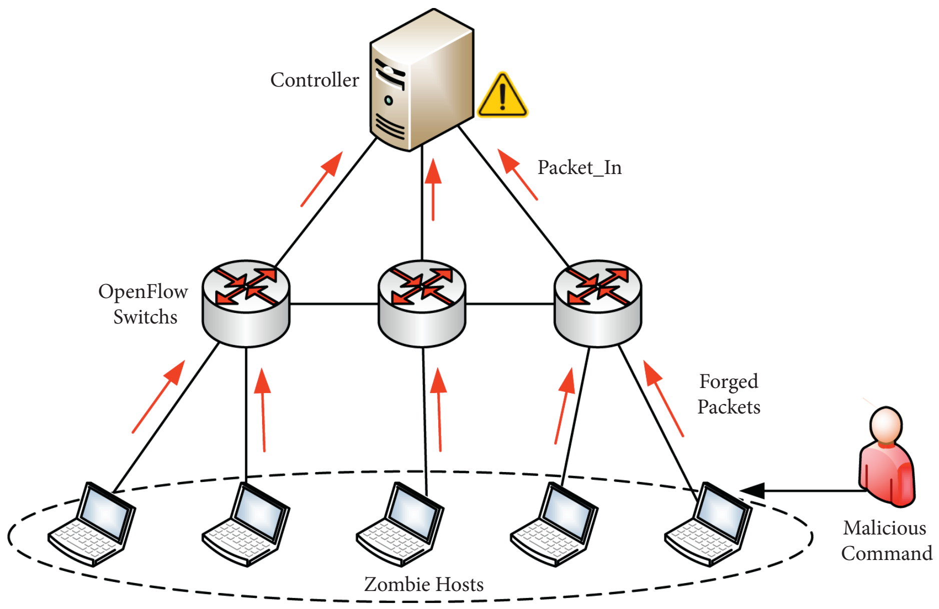


FIGURE 2: The controller targeted DDoS attack in SDN.

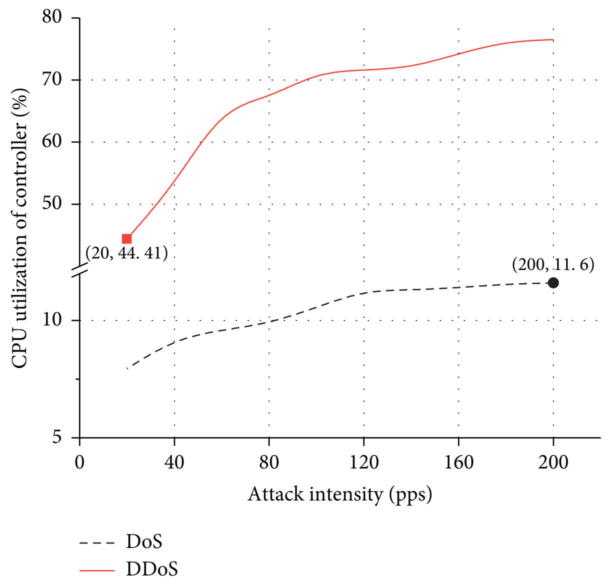


FIGURE 3: The CPU utilization rate of ONOS controller under DoS and DDoS attacks (the upper limit of the Y-axis in DoS group is 12% and is 80% in DDoS group).

threat information can be shared on blockchain via smart contracts. The detection module can collaboratively detect DDoS attacks among multiple controllers and trace the source of attack. The mitigation module can issue defense policies near the source of attack. And the defense strength can be adjusted in conjunction with the controller's real-time load to reduce the misdiscarded rate of normal traffics.

4.1. Architectural Components Overview. BSD-Guard consists of two main modules: secure middle plane and blockchain, as displayed in Figure 5. The secure middle plane stands between the control plane and the data plane, which contains threat detecting and policy issuing functions and smart contract APIs interacting with the blockchain. The threat detecting function collects *sFlow* and intercepts *packet_in* messages sent from data plane to control plane. The policy issuing function receives DDoS mitigating policies from blockchain and registers flow rules into the intrusive switches. The smart contact APIs are responsible for reporting threat information to the blockchain and receiving cocalculated defense strategies. The blockchain plays the role of storage and collaborative sharing of threat state information for multiple SDN domains. It contains blockchain nodes and smart contracts. The DDoS threat information of multiple controllers can be aggregated in blockchain to identify the cross-domain DDoS attack behavior. And the information stored on blockchain can not be tampered by malicious attackers.

In terms of workflow, the system is divided into detection stage, collaboration stage, and mitigation stage. The complete processes are introduced as the following seven steps, and the interaction flow is shown as Figure 6.

(1) In each SDN domain, the threat collecting module collects *sFlow* countersample messages periodically by *sFlow* agents deployed on each OpenFlow switch. Once detecting the velocity of flows exceeds the specified threshold, the detection program records the corresponding switch's *IP* and *port.*

(2) The secure middle plane resolves the *packet_in* messages collected from OpenFlow switches whose port is overspeeding. The *data* field will be extracted for inspecting the original message that triggers table-missing on switch.

(3) The fields extracted from *packet_in* are used to periodically calculate the suspect rate of new flow. And the black/graylists (including *SwitchIP*, *Port*, *IP*, *Mac*,

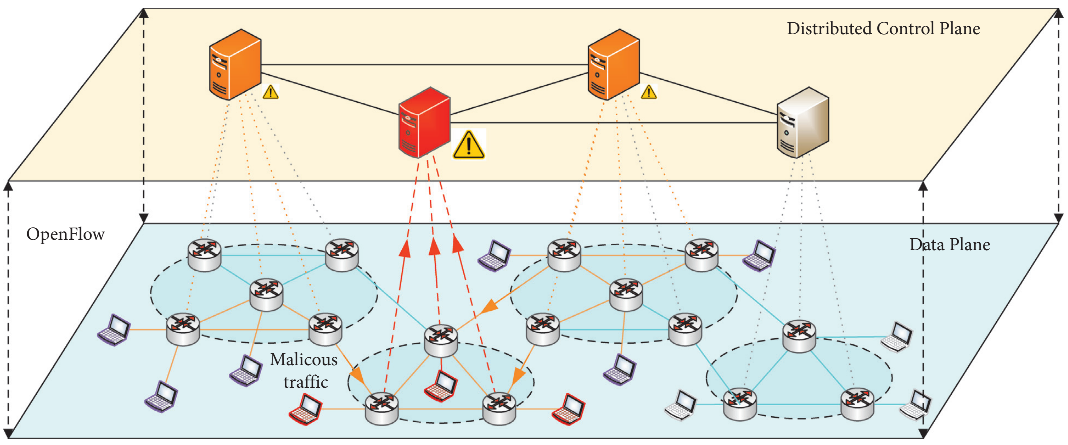


Figure 4: The DDoS scenario in SDN with multiple controllers.

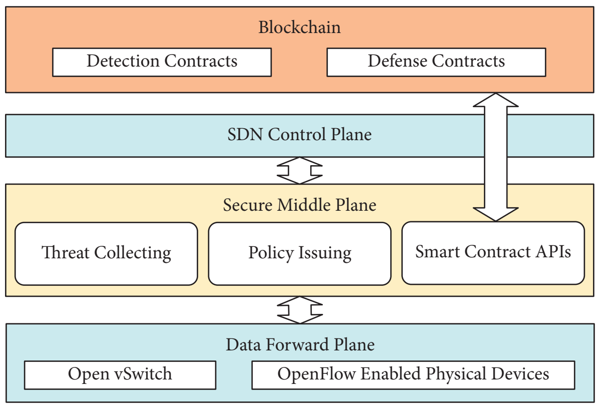


Figure 5: The overview architecture of BSD-Guard.

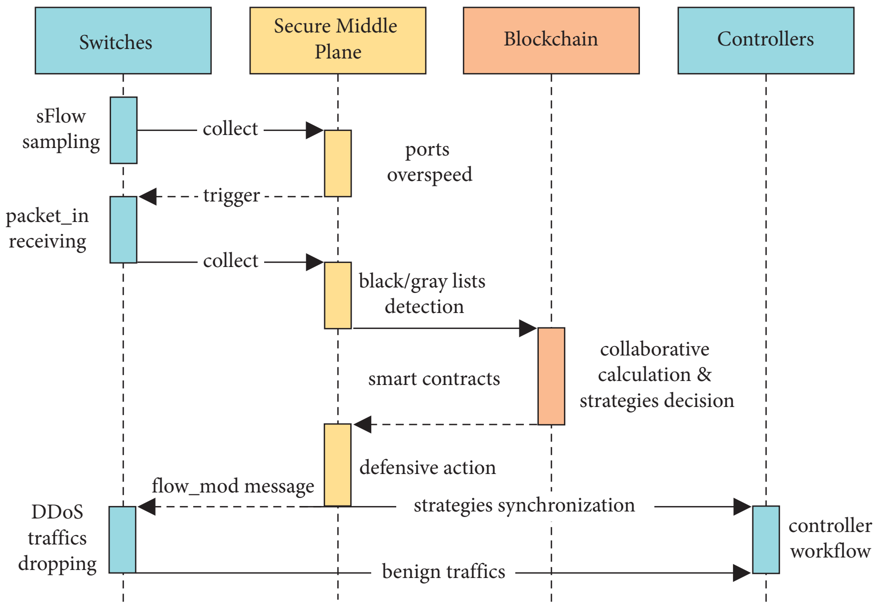


Figure 6: The interaction flows of BSD-Guard.

and suspect rate) will be reported upon blockchain through the smart contract.

(4) The data plane network topologies of multiple controllers are also recorded on the blockchain. Once changed, the real-time topology will be updated by the smart contract.

(5) The collaborative detection algorithm combines the suspect lists and multiple controllers' topology in the previous steps and identifies a complete attack path on the blockchain.

(6) The mitigation algorithm establishes the defense strategies according to the detection results and issues the corresponding instructions to the secure middle plane of the victim controller.

(7) Each secure middle plane executes the defense actions based on the received strategies and controller's real-time load elastically. The *flow_remove* and *flow_mod* message are installed into switches and synchronized to the controller to clear meaningless flow table entries and issue new defense flow table entries.

4.2. Fine-Grained Detection Based on Suspect Rate. Based on the analysis of adversary model, the controller targeted DDoS attack is launched from data plane. Large amounts of forged packets trigger switches sending many meaningless *packet_in* to the controller. In order to detect DDoS attack launched from data plane in a timely manner, we choose the sFlow protocol, an efficient and flexible approach that does not consume the computing capacity and bandwidth of SDN controller and OpenFlow switches. The sFlow agent deployed on switches can generate *FlowSample* and *CounterSample* messages and send them to the sFlow collector with a fixed period. We can calculate the packet rate of switches' inport and outport by analyzing *CounterSameple* messages. When the inport packet rate of a switch is detected to exceed the normal threshold, it is considered that a DDoS attack may occur. Then the *packet_in* parsing module is activated to extract the original packets that crash from the overspeed port. These information will be recorded into Elasticsearch (ES) database with six elements' tuple: <*SwitchID, InPort, SrcMac, SrcIP, DstMac, DstIP*>. These tuples will be counted with several attributes in each fixed period, which is the same as the sampling period of *CounterSample*. These attributes describe the forged level of *Mac* and *IP*, as the SDN-targeted DDoS attack is mainly launched by forged packets that do not match the existing flow table on the OpenFlow switch. The meanings of these attributes are listed in Table 1. The suspect lists and suspect rate will be calculated based on these attributes in the following description.

Inspired by [20], we present an entropy based calculation method of packet's suspect rate. For each element in Table 1, we use the frequency of each element to approximately estimate its probability in each statistical period.

$$p_i = \frac{X_i}{\sum_{i=1}^{N} X_i}. \qquad (1)$$

Table 1: The characteristics of six elements' tuple in each period.

Abbreviation	Explanation
Switch_IP	The *IP* address of switch
Inport	The overspeed port on switch
SrcMac_num	The arising number of *SrcMac*
SrcIP_num	The arising number of *SrcIP*
DstMac_num	The arising number of *ScrMac*
DstIP_num	The arising number of *DstIP*

X represents the element in each tuple (X will be replaced by *SwitchID, InPort, SrcMac, SrcIP, DstMac, DstIP* in the calculation process), and the lower corner i represents the item number of the *packet_in* during the statistical period. Then we can get the information entropy value of each attribute in the tuple by

$$H(X) = -\sum_{i=1}^{N} p_i \log\, p_i. \qquad (2)$$

The information entropy in (2) has been widely used in detecting DDoS traffics. It shows good performance in demonstrating the discrete degree of statistical features. However, it can only represent the overall dispersion of an attribute during the statistical period and cannot pinpoint which specific item affects the entropy value. Therefore, we introduce entropy based suspect rate calculation method, which combines the entropy of each attribute with the frequency of occurrence of the corresponding item.f is a new flow, representing a *packet_in* item in ES database. f_i represents one of the attributes in six elements' tuple. For example, $H(f_{srcmac})/p(f_{srcmac})$ reflects the suspicious level of *SrcMac* in this *packet_in f*. If the *SrcMac* is forged by random generating, $H(f_{srcmac})$ is higher than normal level and the $p(f_{srcmac})$ value is less than normal level. Therefore, the calculated suspect_rate$_f$ will be large when address forged DDoS attacks occur. The normal level of $H(f_j)/p(f_j)$ can be obtained in normal traffic scenario, recorded as θ_{normal}. For the attribute of *SrcMac, SrcIP, DstMac, DstIP*, if the value of $H(f_j)/p(f_j)$ is greater than θ_{normal}, the corresponding attribute can be judged as randomly forged. The malicious or hijacked host intends to stimulate the switch to generate a large number of *packet_in* to send to controller for consuming its computational load and storage. Therefore, we can detect the *packet_in* categories with different combinations of forged addresses. The real-time suspect lists can be figured out by periodically accessing the ES database updated in real-time. For the types of *packet_in* whose partial addresses are forged, the real address can be recognized into the blacklist and the suspect rate can be calculated based on (3). For the type of *packet_in* whose addresses are all forged, since the real source or destination address cannot be identified, only the corresponding overspeed switch's port can be recorded into graylist. Therefore, the graylist contains victim controller ID, switch IP and port, and suspect rate. The examples of graylist and blacklist are listed in Table 2.

Table 2: The example of graylist and blacklist stored on blockchain.

Type	*SwitchIP*	*SwitchID*	*Port*	*DstIP*	*DstMAC*	*SuspectRate*
GrayList	192.168.188.121	1c48cc37ab254bc1	ge-1/1/19	Null	Null	0.764 3
BlackList	192.168.188.199	45ac29bc3714dbc1	ge-1/1/3	192.168.188.201	9A-26-F7-08-0B-2 F	0.866 1

$$\text{suspect\_rate}_f = \sum_{j\in tuples} \frac{H(f_j)}{p(f_j)}. \tag{3}$$

4.3. Suspect Lists Sharing Based on Smart Contract. Several detection and mitigation schemes have considered collaboration among multiple switches and controllers to tackle widely launched DDoS attacks. However, the complexity of deployment and the limitation of information sharing restrict the practical effectiveness of collaborative defense. Abou El Houda et al. [8] reported the suspect *IP* addresses from the victim domain to the collaborative domains by means of smart contract, which can block the illegal traffics in the source and intermediate domains. However, this approach can only deal with traditional DDoS attack against hosts. The sharing information only includes suspect *IP* addresses that cannot cope with more complex attack scenarios in which the *IP* addresses of malicious packets are forged. Considering this situation, we focus on SDN-targeted DDoS attacks in multiple controllers scenario and share the fine-grained DDoS threat information between multiple SDN domains. More specifically, through the collaborative sharing of the suspicious source or destination addresses (*IP* or *Mac*), the edge switch and port, and the suspect rate, a more precise detection and mitigation mechanism can be established.

We design two detection strategies in this collaboratively sharing mechanism with smart contracts. In the first strategy, we focus on the intradomain DDoS attack. Under the intradomain scenario, the destination *Mac* and *IP* of attack packets are randomly forged, which results in the forged packets not being forwarded to the neighboring domain. Therefore, only the intracontroller can suffer from a large number of meaningless *pacekt_in* request messages. In this case, if the source *Mac* or *IP* in the original packets is genuine, the corresponding packets will be precisely dropped at the edge switch by the blacklist strategy mentioned in Section 4.2. However, if the tricky attacker forges all the source *Mac* and *IP*, it is impossible to locate the specific puppet host, and only the abnormal switch's port can be determined. This situation makes the defender very embarrassed. If discarding all messages coming from that switch's port, the normal service traffic will be affected innocently. And if multiple switches are injected with low-intensity forged packets, it will escape the threshold of single-point detection. To solve this problem, we deploy the smart contract to query the graylist related to the same controller on blockchain during the period. Multiple suspect lists from multiple switches are jointly calculated to derive the DDoS attack strength under the global view. This method avoids the failure of missing forged packets below the overspeed threshold on an individual switch.

In the second strategy, we focus on the controller targeted DDoS attack across domains. In the cross-domain scenario, the attacker can construct a large number of packets with forged source *IP* or *Mac* and real destination *IP* and *Mac*, which will stimulate the generation of *packet_in* of all switches on the path from attack source to destination. The controller issues forwarding rules based on the real destination address, so that the forged packets can be forwarded to the destination host hop by hop. Finally, the last-hop switch will be forced to generate abundant *packet_in* messages due to aggregation effect. As shown in Figure 4, the controller in destination domain suffers a serious DDoS attack from the neighbor domains. We collect *packet_in* messages of each overspeed switch port to calculate the blacklist (contains the controller ID, six elements' tuple, and suspect rate) and then store them on blockchain via smart contract. At the same time, the global topology and cross-domain links collated from each controller will also be uploaded to blockchain in real time. Once multiple blacklists with the same destination *Mac* or *IP* exist on the blockchain and the link formed by the suspect switches' ports conforms to the global topology, the link can be confirmed as the attack path of cross-domain DDoS. And the first-hop switch is the edge switch that brings in DDoS attack traffics. The cooperative detection algorithm of cross-domain DDoS attack is shown in Algorithm 1. The smart contract of blacklist is shown in Table 3, and the functions of smart contract consist of storing, searching, updating, and deleting blacklist. Once deployed on the blockchain, these functions can be executed automatically to share the blacklist of multiple control domain on the blockchain. Similarly, the smart contract of graylist equips the same functions to operate graylist on the blockchain.

4.4. Elastic DDoS Mitigation Based on Controller Load. We also design an elastic DDoS mitigation mechanism for different attack scenarios. In terms of the graylist scenario, a large number of meaningless *packet_in* come from switches within the controller domain and no valid blacklist features can be extracted from the raw data of messages. We develop a defense strategy to install *flow_mod* message to disable the graylisted switches' port that generated forged packets. And the disable time depends on the suspect rate and the real-time load of controller. Specifically, the field of *hard_time* in the *flow_mod* message can be calculated as

$$\text{hard\_time} = SR_{ij} * \frac{\text{Num}GL_k}{\text{Num}PktIn_k} * 30\,(s). \tag{4}$$

SR_{ij} represents the suspect rate of port j on switch i, $\text{Num}GL_k$ represents the number of graylists of controller k, and $\text{Num}PktIn_k$ represents the total received *packet_in* of controller k during the continuous period. The benchmark

```
Input: Blacklists, Topology, $i \longleftarrow 0$, $j \longleftarrow 0$.
Output: Attack_path
(1) for blacklist[$i$] in Blacklists do
(2)     if blacklist[$i$].inport is connected with a host then
(3)         Attack_path[0] $\longleftarrow$ blacklist[$i$]
(4)         Blacklists.remove(blacklist[$i$])
(5)         break
(6)     else
(7)         $i \longleftarrow i + 1$
(8)     end if
(9) end for
(10) while Blacklists is not empty do
(11)     for blacklist[$j$] in Blacklists do
(12)         if $\langle$blacklist[$i$], blacklist[$j$].inport$\rangle$ in Topology then
(13)             Attack_path.append(blacklist[$j$])
(14)             Blacklists.remove(blacklist[$j$])
(15)             $i \longleftarrow j$
(16)         else
(17)             $j \longleftarrow j + 1$
(18)         end if
(19)     end for
(20) end while
```

Algorithm 1: Cooperative detection of DDoS attack path.

Table 3: Details of blacklist based collaborative DDoS detection smart contract.

Contract address	0x30f60167c7fb71444d6b90c9b53e93fb4e4eaae1
contractName	DDoSBlackListManager
abi	[{“constant”:false, “inputs”: [{“name”:“switchIp”, “type”:“string”}, {“name”:“srcIp”, “type”: “string”}], “name”:“ deletOneBlackList”, “outputs”:[{“name”:”“, “type”:“int256”}], ...
bytecodeBin	608060405260043610610083576000357c0100000000000000000000000000000000 00000000000000000000000900463ffffffff16806353eb3d90146100885780636bde...
Functions	storeBlackList(), searchBlackList(), updateBlackList(), deletBlackList()

of 30 seconds is based on our experimental test. If too long, the forwarding of normal traffic on this port will be affected. Instead, too short *hard_time* will reduce the effectiveness of defense. The cooperative mitigation algorithm based on graylist is shown in Algorithm 2.

In the cross-domain scenario, the policy issuing module on the secure middle plane will install *flow_mod* message to the corresponding switches on the detected attack path based on blacklist. Specifically, when the blacklists and attack path are detected, the *flow_mod* commands take effect in all switches on the attack path immediately to quickly eliminate the harm of DDoS attack. When the flow table entry exceeds the *hard_timeout*, only the first-hop switch is under-monitored. If the *CounterSample* overspeeds again and the incoming messages still match the blacklist, the same defensive *flow*_mod will be directly reissued on the first-hop switch. And the remaining switches on the path do not need to install *flow*_mod again because the forged packets have already been dropped on the first-hop switch. This mechanism can save switch's flow table space and prevent the spread of this type of DDoS attack at the source of attack. Meanwhile, this mechanism reduces the possibility of misdiscarding of normal traffics on the subsequent switches. The identification of attack path can also provide a priori knowledge for later defense.

These two mechanisms mentioned above will achieve real-time detection and accurate defense against cross-domain DDoS attacks. The benefits are as follows: (1) Timely discarding forged packets at the source of attack will occur. (2) The attack path crossing domains can be identified. (3) Large-scale and low-intensity DDoS attacks can also be detected. (4) The blacklists or graylists stored on blockchain and automated execution of smart contracts can prevent being tampered by malicious attackers.

5. Experiment and Evaluation

We first introduce our implementation of the BSD-Guard system and describe the experimental setups in both software and hardware environment. Finally, we discuss the detection and mitigation results and analyze the characteristics of the BSD-Guard system.

5.1. Implementation. We implement the BSD-Guard system, including the blockchain and secure middle plane. The secure middle plane consists of network state collection

```
Input: Graylists on blockchain
Output: flow_mod message
(1) Group the graylist by victim controller ID
(2) for Each victim controller $C_k$ do
(3)     Calculate the Num$GL_k$ and Num$PktIn_k$
(4)     for Each suspect switch do
(5)         Calculate the hard_time of each suspect port
(6)         Construct the flow_mod message according to graylist and hard_time
(7)         Issue the flow_mod message to switches
(8)     end for
(9) end for
```

Algorithm 2: Elastic mitigation based on graylist.

module, DDoS detection module, defense policy issuing module, and smart contract APIs. All of them are deployed in *Docker* containers, which are convenient for management and migration. Meanwhile, we install the ONOS controller on the Huawei 2288H V5 server equipped with Intel(R) Xeon(R) Gold 6130 CPU and 64 GB memory. In terms of forwarding devices, we employ commercial OpenFlow switch Pica8 AS4610-54T to establish the data plane. We develop and install an application called *midonos* on ONOS to keep the communication between controller and secure middle plane. The blockchain is deployed among the secure middle planes with distributed blockchain nodes. We also use the Elasticsearch (ES) database to store *sFlow* and *packet_in* messages in a high-performance server. We employ four Ubuntu hosts as attacker, victim, and normal users, respectively, in our environment. The experimental network topology is displayed in Figure 7. There is no east-west interface between two ONOS controllers, and switches in the data plane have cross-domain links. We employ FISCO and WeBASE platform [34] to provide blockchain service. WeBASE is a set of common components built between blockchain applications and FISCO-BCOS nodes. It standardizes blockchain application development into five steps: deployment, configuration, development of smart contracts, development of application layer, and online management, which simplifies the process of deploying smart contracts. The interface of the node console (v2.8.0) is shown in Figure 8, which includes the management of blockchain nodes and smart contracts. Administrators can directly edit the contract's ".sol" file and then compile and deploy them on the blockchain.

5.2. Experimental Setup. We construct four experiments under two different attack scenarios to verify that our proposed system can detect and mitigate DDoS attacks. The effectiveness of the proposed system will be evaluated compared with the OpenFlow process without defense measures. First, we construct the DDoS attack scenario in one control domain. The $host_1$ and $host_2$ are selected as attackers to launch UDP flooding packets. We design two different address random methods of forged packets to verify the defense strategies specifically. The "UDP0000" represents the UDP flooding packets with randomly forged <*SrcMac, SrcIP, DstMac, DstIP*>. The "UDP1100" represents the UDP flooding packets with randomly forged <*DstMac, DstIP*> and genuine <*SrcMac, SrcIP*>. The *Scapy* will be used to generate flooding UDP packets under 250 pps attack rate on two hosts. We measure the CPU utilization rate of ONOS controller and the rate of *packet_in* messages the controller received. Second, based on the previous scenario, we adjust the attack intensity of "UDP0000," ranging from 100 pps to 1000 pps. We also compare the CPU utilization of ONOS controller under the OpenFlow process and our BSD-Guard process. Third, we construct the DDoS attack scenario across two control domains. In Figure 7, the two controllers have no east-west interface, and the threat information of the two control domains is shared and synchronized through blockchain by the smart contract APIs on secure middle plane. We launch the TCP SYN flooding attack on $host_1$ with forged packets "TCP0011," which consists of randomly forged <*SrcMac, SrcIP*> and genuine <*DstMac, DstIP*> of $host_3$. Finally, we set up a comparative experiment to verify whether the proposed defense method interferes with normal traffic. We set the $host_2$ as a normal user and test whether the $host_2$ and $host_3$ can keep communication normally.

5.3. Experimental Result. In the first experiment, we compare our proposed system BSD-Guard with OpenFlow (no defense) mechanism under two types of UDP DDoS flooding (UDP0000 and UDP1100). The CPU utilization and received *packet_in* rate of ONOS controller are shown in Figure 9. In Figure 9(a), we launch UDP0000 flooding attack from two seconds to the end. It can be clearly seen that the *packet_in* rate and CPU utilization keep a continuously high level after the attack under OpenFlow scenario. Differently, under the BSD-Guard scenario, these two metrics rapidly decrease after five seconds because the defending *flow*_mod has been installed on the switch at the peak position of the curve. It can be validated by entering "ovs-ofctl dump-flows bridge" command on Pica8 switch, and the defending flow table entry contains suspicious inport generated by graylist. A similar phenomenon can be observed in Figure 9(b); the BSD-Guard takes only four seconds to detect and mitigate UDP1100 flooding attack. The response performance is better than the 13 seconds of Cochain-SC [7]. The *packet_in*

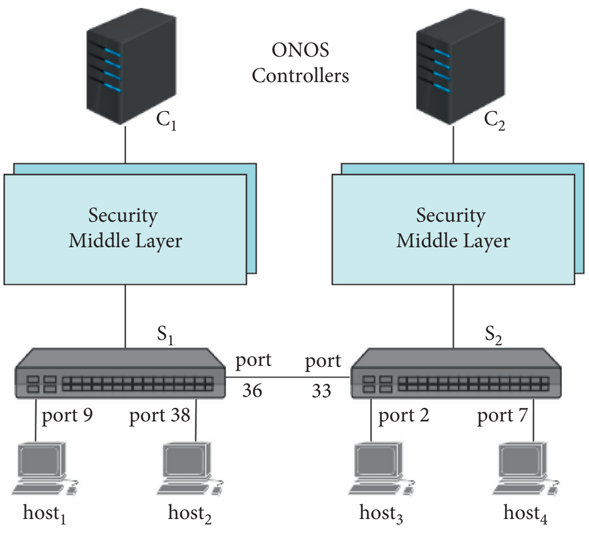
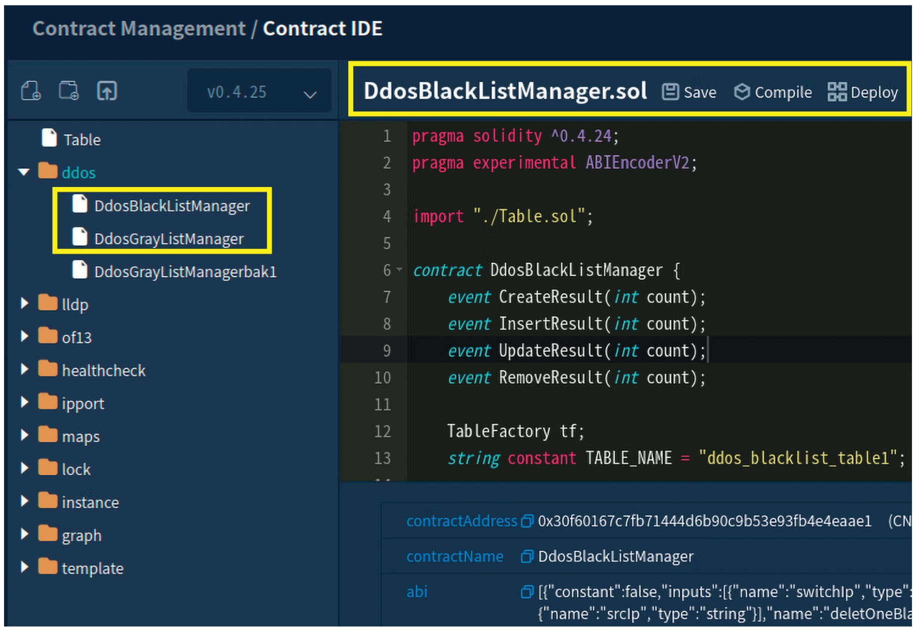


Figure 7: The experimental network topology.



```
pragma solidity ^0.4.24;
pragma experimental ABIEncoderV2;

import "./Table.sol";

contract DdosBlackListManager {
    event CreateResult(int count);
    event InsertResult(int count);
    event UpdateResult(int count);
    event RemoveResult(int count);

    TableFactory tf;
    string constant TABLE_NAME = "ddos_blacklist_table1";
```

```
contractAddress 0x30f60167c7fb71444d6b90c9b53e93fb4e4eaae1   (CN
contractName    DdosBlackListManager
abi             [{"constant":false,"inputs":[{"name":"switchIp","type":
                {"name":"srcIp","type":"string"}],"name":"deletOneBla
```

Figure 8: The management interface of smart contracts on WeBASE platform.

rate decreases from 779pps to 4pps within two seconds, and the CPU utilization of controller decreases from 22.5% to 5% within 2.5 seconds. Meanwhile, we can check the flow table on switch, which contains the entry with suspicious inport and <*SrcMac*, *SrcIP*> generated by blacklist, and the forwarding action is *drop*. It is worth noting that the CPU utilization of UDP0000 in Figure 9(a) is higher than UDP1100 in Figure 9(b) under OpenFlow mechanism with the same attack intensity. The randomly forged *SrcMACs* will be regarded as large amounts of new hosts in the controller's view. The creation and maintenance of new hosts' identity will consume a lot of CPU on the controller. In contrast, just forging the destination address will only make the controller consume CPU for calculating forwarding rules, without creating new forged hosts. We also construct other types of packet forged methods, including UDP/TCP-1000/0100/1101. The results verify that our proposed system can report corresponding blacklists and install the special defense flow table according to the forged packet characteristics. This precise defense pattern can effectively avoid the incorrect discarding of normal packets.

In the second experiment, we adjust the intensity of DDoS attacks from 100pps to 1000pps and record the average CPU utilization of ONOS controller under OpenFlow and BSD-Guard mechanisms. As is shown in Figure 10, with the increasing of attack intensity, the CPU utilization of controller keeps rapid growth under OpenFlow scenario. The CPU utilization reaches amazing 85.6% when the attack

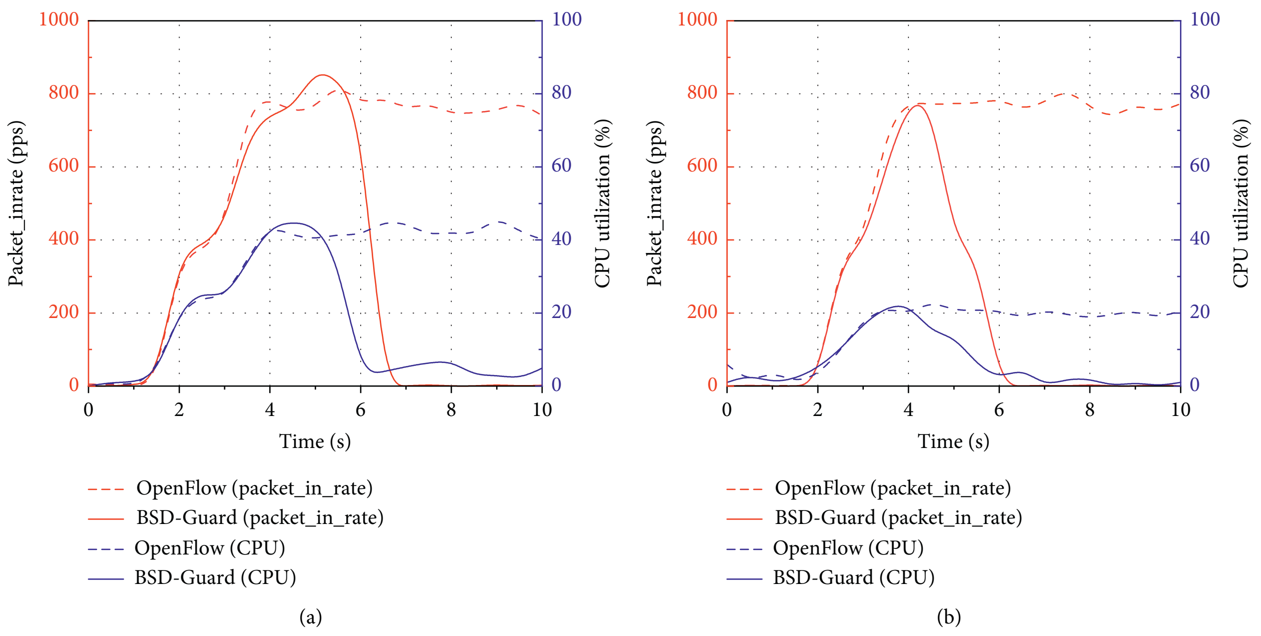


Figure 9: The *packet_in* rate and controller CPU utilization under udp0000 and udp1100 DDoS attack scenarios. (a) udp0000 attack; (b) udp1100 attack.

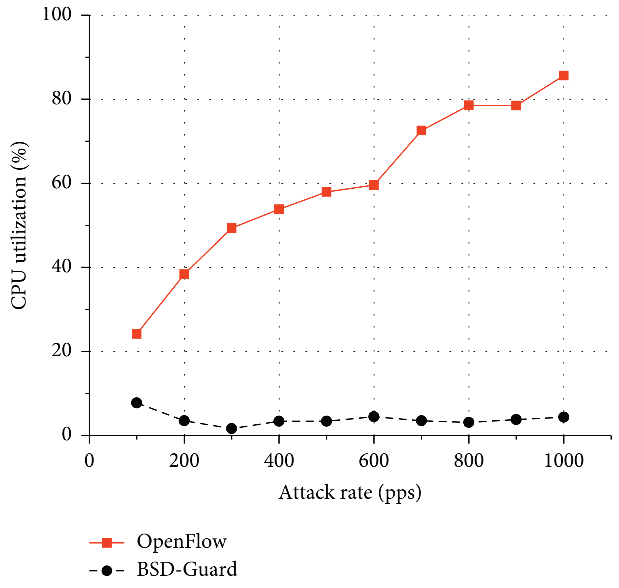


Figure 10: The CPU utilization of controller under different attack intensities.

rate is 1000 pps. In contrast, the values under BSD-Guard maintain a stable low level around 5%, which indicates that our proposed defense mechanism can resist high-intensity SDN-targeted DDoS attacks well.

In the third experiment, we construct a TCP SYN attack between two neighboring domains under the management of two ONOS controllers. The forged packet TCP0011 has randomly forged source addresses and genuine destination address of host_3, which allows the forged new packet to be forwarded to the target controller's domain and reach the target $host_3$. In this process, the two controllers are successively involved in the calculation of forwarding policies, as shown in Figures 11(a) and 11(b). The received *packet_in* rate and CPU utilization of the controller in source domain are earlier than the controller in targeted domain. The time difference between them is equal to the sum of switch forwarding delay and the delay caused by the source controller to make forwarding policies and install them to switch. Meanwhile, an attack path "$\text{host}_1 \longrightarrow S_1\,(port_9) \longrightarrow S_1\,(\text{port}_{36}) \longrightarrow S_2\,(\text{port}_{33}) \longrightarrow S_2\,(\text{port}_2) \longrightarrow \text{host}_3$" can be identified on blockchain, which is generated by the detection algorithm in Algorithm 1 based on the corresponding blacklists. We also inspect the flow table entries on switches S_1 and S_2, and the result shows that only S_1 in source domain installs defense flow table entry "$\text{flow\_}i\ d = 65542, \text{priority} = 200, \text{in\_port} = 9, ds\ tmac = 00\!:0c\!:29\!:cf\!:76\!:ca, ds\ tip = 192.168.188.123, \text{actions} = \text{drop}$" (the *dstmac* and *dstip* are the addresses of host_3). The above results prove that our proposed collaborative blockchain-based defense policy is implemented. The DDoS traffics have been intercepted in the source domain.

We also append a comparison experiment to verify the effectiveness of our proposed method; that is, only the attack source switch continuously updates the defense flow table, and the normal traffics aimed to targeted domain will not be discarded. We partially modify the proposed mechanism called Isolated BSD-Guard, in which the detection module is reserved, but the smart contract-based (SC-based) collaborative defense module is removed. The attack is performed again in the third experiment, in which the forged TCPSYN0011 packets are launched from host_1 to host_3 with the same intensity. After launching an attack, the same detection process is ongoing under BSD-Guard and Isolated BSD-Guard. And the rate of *packet_in* decreases after a few seconds. However, in the Isolated BSD-Guard group, both

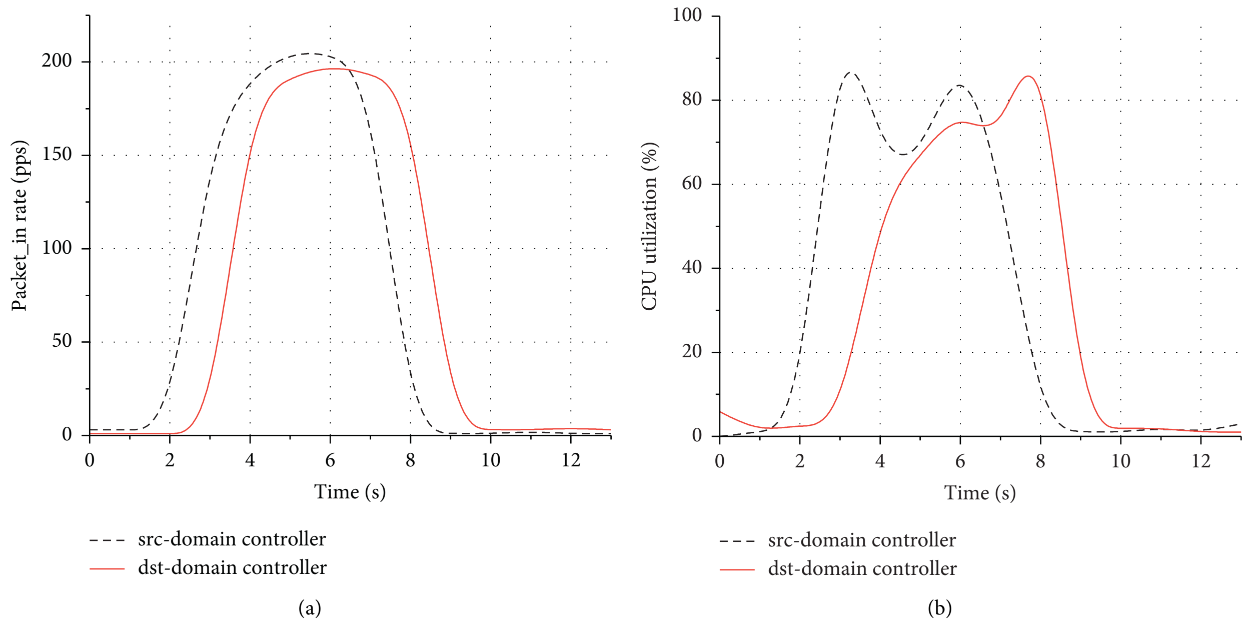


FIGURE 11: The *packet_in* rate and CPU utilization of source and destination domains controller under tcpsyn0011 DDoS attack with BSD-Guard. The change in the attack source domain is ahead of the change in the target domain. (a) *packet_in* rate of source and destination domains controller; (b) CPU utilization of source and destination domains controller.

the switches S_1 and S_2 issue and update defense flow table entries to discard the forged packets targeted to host$_3$. At the same time, we launch the normal TCP SYN request from host$_2$ to host$_3$. The result shows that the request packets cannot reach host$_3$ because of the defense flow table entry on switch S_2. In contrast, BSD-Guard did not discard request packets from normal user *host*$_2$ while defending against DoS attacks from *host*$_1$. The compared results in Table 4 prove that the BSD-Guard can collaborate the suspect traffic information of multidomain SDN, identify the attack path with global view, and mitigate controller targeted DDoS from the source of attack.

We also count the time overhead of our detection and defense process. The detection module consists of state collection and detection algorithm. The collected information is stored to ES database in real time. We divided the total time overhead into four stages, including T1 (searching ES database and computing), T2 (forming black/graylists and uploading to blockchain), T3 (cooperative detecting by smart contract), and T4 (issuing defense strategies). Table 5 demonstrates the time overhead of five groups of experiments, and the average of total time is 675.02 ms, which is mainly occupied by T1 and T2. The millisecond-level block generation speed can meet the requirement of defense and is much faster than 14s in Ethereum [10]. Since what is stored on the blockchain is not the original data of *sFlow* and *packet_in*, but the blacklist and graylist formed by statistical analyzing, the amount of data uploaded on the blockchain is not very large. After the successful defense, the smart contract will also delete the expired data to save space on the blockchain.

TABLE 4: Comparison of BSD-Guard and Isolated BSD-Guard under attack.

Indicators	BSD-Guard	Isolated BSD-Guard
Detect DDoS	Yes	Yes
SC-based defense	Yes	No
Identify attack path	Yes	No
Permit normal flow	Yes	No
Flow table space	Saved	Wasted

TABLE 5: The time overhead during detection and mitigation stages (ms).

Group	1	2	3	4	5
T1	483.37	444.21	307.73	331.55	362.81
T2	276.24	258.21	311.02	206.94	262.71
T3	25.21	23.66	21.55	25.93	27.13
T4	1.50	1.21	0.97	1.63	1.54
Total	786.32	727.29	641.27	566.05	654.19

5.4. Characteristics Analysis. The main objective of BSD-Guard is to provide a collaborative, elastic, lightweight, easy-to-deploy controller targeted DDoS attacks detection and mitigation scheme based on blockchain and smart contract. In this section, we will discuss how our proposed BSD-Guard achieves these characteristics.

(1) *Easy to Deploy.* The implemented functional modules in the BSD-Guard system are deployed in Docker containers, allowing for rapid deployment and cluster scaling. The FISCO platform [34] is employed to provide blockchain, and the official

WeBASE platform provides convenient nodes and contracts management function.

(2) *Collaborative Detecting and Defending.* Multiple controllers can share topology and threat information on the trusted blockchain, on which information can not be tampered by malicious attackers. Collaborative defense makes forged attack traffic discarded at the source switch, which reduces the defense overhead of subsequent switches. And the formation of attack path helps the adoption of more precise defense strategies.

(3) *Precise and Elastic Defending.* The defense policies are established based on the blacklists and graylists stored on blockchain. The generation of defense flow table entries is determined by the characteristics of detected attack traffic, which can avoid dropping of normal traffic. Meanwhile, the duration of defense flow table depends on the real-time load of controller, which makes defense more elastic.

(4) *Lightweight.* The system employs a private blockchain that does not consume additional *gas* and does not affect the performance of ONOS controller. The collaborative defense policies save flow table space on hardware switches. Compared with machine learning based detection algorithms, in which the features extracting and data training increase the complexity, our proposed BSD-Guard system is more lightweight.

6. Conclusion

In this paper, we proposed BSD-Guard, a collaborative and elastic blockchain-based detection and mitigation framework to protect SDN against controller targeted DDoS attack. BSD-Guard consists of the secure middle plane and blockchain. The secure middle plane can collect traffic information from data planes, including *sFlow* and *OpenFlow*. The blockchain stores and shares the blacklists and graylists via smart contracts and makes global defense strategies. We design two types of detection and mitigation mechanisms under the intradomain and cross-domain scenarios. We deploy BSM-Guard on the physical environment to verify the effectiveness of our proposed framework. Three groups of experiments have been conducted to verify the system's defense abilities against various types of DDoS attacks. The experimental results indicate that BSD-Guard can detect DDoS attacks with global view, identify the attack path, and install precise defending flow table entries on the near-attack switches. The SDN controller can be well protected and normal service traffic will not be affected by defense policy. Compared with controller clusters, the introduction of blockchain solves the problem of threat sharing among multiple controllers and achieves rapid response and mitigation of DDoS attacks against controllers within an acceptable time and space range.

Data Availability

The experiment data of BSD-Guard are uploaded at https://github.com/SeuSQ/BSD-Guard/issues/1#issue-1092568880.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFB1804604 and in part by the General Program of the National Natural Science Foundation of China under Grant 62172093.

References

[1] A. Abdou, P. C. Van Oorschot, and T. Wan, "Comparative analysis of control plane security of sdn and conventional networks," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3542–3559, 2018.

[2] L. F. Eliyan and R. Di Pietro, "Dos and ddos attacks in software defined networks: a survey of existing solutions and research challenges," *Future Generation Computer Systems*, vol. 122, pp. 149–171, 2021.

[3] M. Imran, M. H. Durad, F. A. Khan, and A. Derhab, "Toward an optimal solution against denial of service attacks in software defined networks," *Future Generation Computer Systems*, vol. 92, pp. 444–453, 2019.

[4] M. Essaid, D. Kim, S. H. Maeng, S. Park, and H. T. Ju, "A collaborative ddos mitigation solution based on ethereum smart contract and rnn-lstm," in *Proceedings of the 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 1–6, IEEE, September 2019.

[5] M. S. Elsayed, N.-A. Le-Khac, and A. D. Jurcut, "Insdn: a novel sdn intrusion dataset," *IEEE Access*, vol. 8, Article ID 165263, 2020.

[6] O. E. Tayfour and M. N. Marsono, "Collaborative detection and mitigation of ddos in software-defined networks," *The Journal of Supercomputing*, vol. 77, no. 11, Article ID 13166, 2021.

[7] Z. Abou El Houda, A. S. Hafid, and L. Khoukhi, "Cochain-SC: an intra- and inter-domain ddos mitigation scheme based on blockchain using SDN and smart contract," *IEEE Access*, vol. 7, Article ID 98893, 2019.

[8] Z. Abou El Houda, A. Hafid, and L. Khoukhi, "Co-iot: a collaborative ddos mitigation scheme in iot environment based on blockchain using sdn," in *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, Waikoloa, HI, USA, December 2019.

[9] B. Rodrigues, T. Bocek, A. Lareida, D. Hausheer, S. Rafati, and B. Stiller, "A blockchain-based architecture for collaborative ddos mitigation with smart contracts," in *Proceedings of the IFIP International Conference on Autonomous Infrastructure, Management and Security*, pp. 16–29, Springer, Zurich, Switzerland, July 2017.

[10] J. Dheeraj and S. Gurubharan, "Ddos mitigation using blockchain," *International Journal of Research in Engineering, Science and Management*, vol. 1, no. 10, pp. 622–626, 2018.

[11] X. You, Y. Feng, and K. Sakurai, "Packet in message based ddos attack detection in sdn network using openflow," in *Proceedings of the 15th International Symposium on Computing and Networking (CANDAR)*, pp. 522–528, IEEE, Aomori, Japan, November 2017.

[12] X. Huang, X. Du, and B. Song, "An effective ddos defense scheme for sdn," in *Proceedings of the IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, Paris, France, May 2017.

[13] R. F. Fouladi, O. Ermiş, and E. Anarim, "A ddos attack detection and defense scheme using time-series analysis for sdn," *Journal of Information Security and Applications*, vol. 54, Article ID 102587, 2020.

[14] W. Chen, S. Xiao, L. Liu, X. Jiang, and Z. Tang, "A ddos attacks traceback scheme for sdn-based smart city," *Computers & Electrical Engineering*, vol. 81, Article ID 106503, 2020.

[15] K. Bhushan and B. B. Gupta, "Distributed denial of service (ddos) attack mitigation in software defined network (sdn)-based cloud computing environment," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 5, pp. 1985–1997, 2019.

[16] M. Hassan, D. Mahmood, Q. Shaheen, R. Akhtar, W. Changda, and S-dps, "An sdn-based ddos protection system for smart grids," *Security and Communication Networks*, vol. 2021, Article ID 6629098, 19 pages, 2021.

[17] B. H. Lawal and A. Nuray, "Real-time detection and mitigation of distributed denial of service (ddos) attacks in software defined networking (sdn)," in *Proceedings of the 26th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, IEEE, Izmir, Turkey, May 2018.

[18] C. Kumar, B. P. Kumar, A. Chaudhary et al., "Intelligent ddos detection system in software-defined networking (sdn)," in *Proceedings of the IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pp. 1–6, IEEE, Bangalore, India, July 2020.

[19] Y. Lu and M. Wang, "An easy defense mechanism against botnet-based ddos flooding attack originated in sdn environment using sflow," in *Proceedings of the 11th International Conference on Future Internet Technologies*, pp. 14–20, ACM, Nanjing, China, June 2016.

[20] K.-Y. Chen, S. Liu, Y. Xu et al., "Sdnshield: nfv-based defense framework against ddos attacks on sdn control plane," *IEEE/ACM Transactions on Networking*, vol. 30, 2021.

[21] S. Y. Mehr and B. Ramamurthy, "An svm based ddos attack detection method for ryu sdn controller," in *Proceedings of the 15th international conference on emerging networking experiments and technologies*, pp. 72-73, ACM, Orlando, FL, USA, December 2019.

[22] J. Cui, J. Zhang, J. He, H. Zhong, and Y. Lu, "Ddos detection and defense mechanism for sdn controllers with k-means," in *Proceedings of the IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pp. 394–401, IEEE, Leicester, UK, December 2020.

[23] V. Deepa, K. M. Sudar, and P. Deepalakshmi, "Detection of ddos attack on sdn control plane using hybrid machine learning techniques," in *Proceedings of the International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pp. 299–303, IEEE, Tirunelveli, India, December 2018.

[24] B. Nugraha and R. N. Murthy, "Deep learning-based slow ddos attack detection in sdn-based networks," in *Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 51–56, IEEE, Leganes, Spain, November 2020.

[25] Y. Xu, Y. Yu, H. Hong, and Z. Sun, "Ddos detection using a cloud-edge collaboration method based on entropy-measuring som and kd-tree in sdn," *Security and Communication Networks*, vol. 2021, Article ID 5594468, 16 pages, 2021.

[26] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R. Mumtaz, and J. González, "Towards sflow and adaptive polling sampling for deep learning based ddos detection in sdn," *Future Generation Computer Systems*, vol. 111, pp. 763–779, 2020.

[27] T.-K. Luong, T.-D. Tran, and G.-T. Le, "Ddos attack detection and defense in sdn based on machine learning," in *Proceedings of the 7th NAFOSTED Conference on Information and Computer Science (NICS)*, pp. 31–35, IEEE, Ho Chi Minh City, Vietnam, November 2020.

[28] K. Nishizuka, L. Xia, J. Xia, D. Zhang, L. Fang, and C. Gray, *Interorganization Cooperative Ddos protection Mechanism*, Internet-Draft, 2016.

[29] J. Steinberger, B. Kuhnert, A. Sperotto, H. Baier, and A. Pras, "Collaborative ddos defense using flow-based security event information," in *Proceedings of the NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 516–522, IEEE, Istanbul, Turkey, April 2016.

[30] U. Javaid, A. K. Siang, M. N. Aman, and B. Sikdar, "Mitigating lot device based ddos attacks using blockchain," in *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*, pp. 71–76, New York, NY, USA, 2018.

[31] Z. Shao, X. Zhu, A. M. M. Chikuvanyanga, and H. Zhu, "Blockchain-based sdn security guaranteeing algorithm and analysis model," in *Proceedings of the International Conference on Wireless and Satellite Systems*, pp. 348–362, Springer, Harbin, China, January 2019.

[32] S. Gao, Z. Peng, B. Xiao, A. Hu, Y. Song, and K. Ren, "Detection and mitigation of dos attacks in software defined networks," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1419–1433, 2020.

[33] K. Giotis, M. Apostolaki, and V. Maglaris, "A reputation-based collaborative schema for the mitigation of distributed attacks in sdn domains," in *Proceedings of the NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 495–501, IEEE, Istanbul, Turkey, April 2016.

[34] Fisco-bcos, "Fisco-bcos-documentation," 2021, https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/.