Received November 30, 2021, accepted December 18, 2021, date of publication December 30, 2021, date of current version January 7, 2022.

Digital Object Identifier 10.1109/ACCESS.2021.3139511

Source-Based Defense Against DDoS Attacks in SDN Based on sFlow and SOM

MENG WANG, YIQIN LU, AND JIANCHENG QIN
School of Electronic and Information Engineering, South China University of Technology, Guangzhou 510641, China

Corresponding author: Meng Wang (w.m15@mail.scut.edu.cn)

This work was supported by the Key-Area Research and Development Program of Guangdong Province, China, under Grant 2019B010137001 and Grant 2020B0101120002.

ABSTRACT In the traditional distributed control network, due to the difficulty in detection and the ambiguous defense responsibility, it is not efficient and effective to detect Distributed Denial of Service (DDoS) attacks in the network where they are launched, which is so-called source-based defense mechanism. Moreover, with the development of cloud computing, Internet of Things (IoT), and mobile Internet, the number of terminals and the communication bandwidth in a single autonomous domain have increased significantly, providing much more easy conditions for organizing large-scale botnets to launch a threatening DDoS attack. Therefore, there is an urgent need for source-based defense against DDoS attacks. The emerging Software-Defined Networking (SDN) provides some new ideas and advantages to solve this problem, such as centralized control and network programmability. In this paper, we proposed a defense method based on sFlow and improved Self-Organizing Map (SOM) model in SDN. This method consists of an sFlow-based macro-detection, which could cover the entire network to perceive DDoS attacks, a SOM-based micro-detection, which is used to recognize the attack traffic, and a response strategy based on the global view given by the controller. The experimental results under open data and simulated attack scenarios have proved the effectiveness of the proposed method, and it also has better overall detection performance than k-means and k-medoids.

INDEX TERMS DDoS attacks detection, network security, software-defined networking, self-organizing map, source-based defense.

I. INTRODUCTION

Most of today's DDoS attacks achieve the malicious purpose through exhausting the limited resources to disrupt the connection or service of normal users. Attackers usually use Botnets to launch a DDoS attack through remotely controlling many malware-infected machines, which is called zombies, and instructing them to continuously and concurrently send a large number of service requests to the target [1]. DDoS attacks not only hinder normal users to get legitimate service, but also affect the experience of other users on the transmission path. Therefore, an ideal defense is to detect and filter the attack traffic at the network as close to the source as possible. According to the classification of defense mechanisms based on their deployment locations, this kind of defense is called source-based defense mechanism and is deployed at the source to prevent the network from generating DDoS attacks. The other two mechanisms are called network-based and destination-based respectively [2]. However, it is not an efficient and effective way to deploy source-based defense in the traditional distributed control network. Firstly, as the zombies are usually spread in multi-domains, the significance of attack hiding in the huge background traffic in a single domain is too small to be detected, and it is also very hard to trace the source of all zombies. Secondly, traditional defense mechanisms are deployed near the aggregation routers at the edge of the network. The edge-based deployment only aims to detect the attack that passes through the protection perimeter. It usually only defends against external attacks, but can not perceive the inner attack that targets intra-domain systems, which is simply ignored and passed to downstream autonomous domains. Thirdly, the responsibility of defending DDoS attacks at the source is ambiguous. The source-based defense mainly prevents the attack traffic from spreading outside a domain. It is unclear whether the source network or the target network

The associate editor coordinating the review of this manuscript and approving it for publication was Claudio Cusano.

should pay for the CAPEX and OPEX. This is the main reason why network entities usually lack the motivation to deploy source-based defense mechanisms.

With the development of cloud computing, IoT, and mobile Internet in recent years, the situation of defense against DDoS attacks at the source network has changed. The need for source-based defense mechanisms has become increasingly urgent. According to report [3], the cloud host accounts for up to 89% of the masters of Botnets due to the accessibility, reliability, and low cost of the cloud platform. Moreover, the inherent characteristics of cloud computing have objectively led to a continuous increase in the frequency of DDoS attacks happening in the cloud environment [4]. The on-demand self-service of cloud service allows attackers to create a Botnet very conveniently and treat the malware as a service to gain illegal revenue. It is reported that a botnet composed of 10,000 zombie nodes only costs 1,000 US dollars [5]. Besides, in the case of multi-tenant coexistence on the cloud platform, some internal malicious tenants can attack other legitimate tenants, which is inside the cloud network and very hard to be detected on the physical perimeter. On the other hand, more than billions of IoT and mobile devices that lack security protection have become the new breeding ground of zombie nodes after accessing the Internet [6], [7]. These devices are easily hacked by attackers to constitute the most powerful Botnet. The numerous zombies and broad bandwidth mean a Botnet in a single autonomous domain can launch a sufficiently threatening DDoS attack. By the way, with the enforcement of network security laws and the increase of cybersecurity awareness, network entities should avoid becoming the source of various network attacks as much as possible. In summary, the situation that current defense mechanisms do not pay much attention to the source-based can no longer meet the demands in many emerging scenarios. Researchers need to rethink the source-based mechanism, and use new technologies, such as SDN, to explore innovative solutions that could avoid the disadvantages in traditional networks.

SDN is a new network architecture aiming to solve the problem that the traditional network architecture is ill-suited to meet today's requirements. In this architecture, the control plane is separated from the data plane, and it is divided into three layers: infrastructure layer, control layer, and application layer. SDN has characteristics such as logically centralized control, global view of the network, programmability of network, etc. More details about SDN could see the introduction in [8]–[10]. The main advantages provided by SDN in designing source-based defense mechanism are summarized as follows [11]–[13]. Firstly, logically centralized control provides the global view of the dominated network, and the controller can obtain traffic statistics from any node of the entire network, which is conducive to constructing a holistic defense to avoid the incapacity of global sense in traditional edge-based defense. Meanwhile, centralized control also provides the possibility for organizing intra- or inter-domain collaborative defense. Secondly, benefiting from the programmability of network and the abstraction of underlying details, SDN enables the source-based defense mechanism to be developed efficiently and run as a lightweight software on the common platform, which greatly reduces the cost and improves the willingness of deployment. Thirdly, flow-based programming and forwarding enable unprecedented precision and flexibility of the response strategy, which could minimize the impact on the normal users when filtering the attack traffic. Based on these advantages, it is more feasible to implement the source-based DDoS attack defense mechanism in SDN than traditional network.

In order to monitor the traffic and collect the customized flow information from the entire network, the sFlow technology is used instead of directly collecting traffic statistics from the controller. The sFlow means sample flow, and it was formally proposed in 2001. It is a packet sampling technology embedded in routers and switches with a centralized control architecture. It can concurrently monitor the traffic in wire-speed mode on all interfaces across the entire network. The distributed agents deployed in switches monitor the traffic passed through them to get the sampled statistical data and then send them to a central collector. It is also an industry-standard supported by almost all SDN switches, so it is feasible to combine with SDN. More details about sFlow could be seen in RFC3176 [14]. In summary, sFlow can not only provide customized collection covering the entire network, but also has some features, such as high scalability and low deployment overhead, that can meet the requirements of deploying the source-based defense mechanism.

This work proposes a novel source-based defense method based on SDN and sFlow. The main contributions of this work can be summarized as:

1). A novel source-based detection method based on sFlow for detecting DDoS attacks in SDNs. This method defines a metric to characterize the nature of DDoS attacks and makes decisions based on statistical theory.

2). An improved SOM model to distinguish between normal and attack traffic in real time. The model strives to reduce the false positive errors to separate the real normal flow from the macro attack traffic as much as possible.

3). A simple response strategy for filtering the attack traffic as close to the source as possible. Dropping or rate-limiting rules are issued to near the attack source according to the global view provided by the controller.

4). Validation of the proposed method under open data and simulated attack scenarios. The results show that the proposed method works well and performs better than k-means and k-medoids algorithms.

The rest of this paper is organized as follows. Section 2 reviews the related works in traditional and SDN networks. Section 3 describes the details of our proposed method. Section 4 describes the design of experiments and the analysis of experimental results. At last, we conclude this paper in section 5.

II. RELATED WORKS

In traditional networks, source-based defense mechanisms are usually deployed at the edge routers of the local network or the access routers of adjacent autonomous systems [2]. A typical defense method is to filter the packets with the forged IP address based on the valid IP address range at the edge routers, such as the IP source address spoofing filtering mechanism proposed in RFC2827 [15]. Protocols specified in RFC4301 [16] and RFC4302 [17] also proposed authentication mechanisms to identify forged IP source address packets, but they are not widely used due to the excessive overhead of the authentication process. Mirkovic and Reiher [18] proposed a source-based defense system to monitor inbound or outbound traffic and detect DDoS flooding attack traffic through filtering the flows that do not match the predefined normal flow models. Paper [19] proposed a method for detecting and filtering DoS bandwidth attacks based on recognizing the abnormal flow imbalances of the rate of traffic between one direction and the opposite direction. Besides, the reverse firewall [20] is also an effective source-based defense method. Compared with traditional firewalls, the reverse firewall is mainly used to prevent illegal packets sent from inside to outside the network. In summary, source-based defense mechanisms are usually based on hard decisions to filter the packet that violates specific rules. Their development level is lagging behind other types of defense mechanisms as they are not very efficient and effective in traditional networks. The main reasons also include source network lacking deployment motivation and the difficulty in recognizing small attack flows from the huge background traffic. However, the source-based defense is the most ideal countermeasure against DDoS attacks, because it can prevent them from the beginning, which not only mitigates the attack but also avoids wasting the bandwidth.

With the development and deployment of SDN in recent years, researchers have proposed many SDN-based defense methods against DDoS attacks. Braga *et al.* [21] proposed a lightweight detection method based on SOM and SDN. It periodically collects the flow entry information on OpenFlow switches to construct six new features as inputs of the detection model, and then detects the DDoS attack according to the clustering result. Different from the edge-based single-point detection in traditional networks, this method uses SDN to obtain global traffic information. But its defects include frequent calls to get flow table statistics through the controller introducing new vulnerability to the network, and only detecting the occurrence of DDoS attacks without recognizing the specific attack flows. Xu and Liu [22] and Nam *et al.* [23] also proposed defense methods based on SOM and SDN. The former used flow intensity and flow ration asymmetry as input features, and the latter used entropy-based flow features. Both of them aimed at destination-based defense and can only detect the occurrence of DDoS attacks. Sahoo *et al.* [24] proposed a method to detect DDoS attacks in SDN based on an improved support vector machine (SVM) model using flow features obtained from OpenFlow switches in a certain time interval. The innovation of this work mainly includes the combination of genetic algorithm and an improved kernel function with SVM to improve detection accuracy, as well as the application implementation and response strategy based on SDN. Haider *et al.* [25] proposed a deep convolutional neural network (CNN) ensemble framework based on SDN to detect DDoS attacks. The framework can be deployed on any commercial SDN controller as a northbound application. Paper [26] proposed a modular architecture based on SDN to integrate intrusion detection system (IDS) with different machine learning models, including decision tree, random tree, random forest, reduced-error pruning tree, MLP, and SVM, to detect different low-rate DDoS attacks. The experimental results showed that all of the above methods reached an accuracy of more than 90%. In summary, combining various machine learning-based detection methods with SDN is a hot research direction in the area of DDoS attack defense. Benefiting from the centralized control of SDN, such methods naturally can perceive the global attack situation. That is, although some methods are designed or defaulted to work as destination-based defense mechanisms, they can still be applied to source-based or network-based scenarios to a certain extent. Therefore, SDN is naturally suitable for solving the source-based defense problem as it is different from the traditional edge-based defense model.

In the area of SDN-based detection, many methods compute features based on the flow statistics collected from OpenFlow switches through directly and frequently sending requests to the controller, such as [21]–[23], [27]–[30]. But other methods choose to use sFlow to build a flow collection system independent of SDN. Giotis *et al.* [31] proposed an entropy-based anomaly detection method and mitigation mechanism based on combining sFlow with SDN. In this method, sFlow agents are distributed in the entire network to collect flow. DDoS attacks are detected according to the abnormal entropy of the source (or destination) IP address and port. Paper [32] used sFlow to propose an application called FlowTrAPP in the SDN environment to defend against DDoS attacks in the data center. This application uses flow rate and flow duration as features to define five types of attacks according to the upper and lower bounds, and then monitors the traffic in the data center based on preset thresholds. When some flows fall into the interval corresponding to an attack, the application will block attack flows by issuing OpenFlow rules. In summary, several reasons make sFlow-based flow collection better than that with controller-based. Firstly, frequently sending requests to the controller to obtain the flow information from all switches will add too much additional burden to the controller and switch, which brings a single point failure vulnerability to the controller. The high-speed traffic in the data plane may further form a DoS attack against the controller [31]. Secondly, as the OpenFlow flow entries are issued by multiple decision-making entities from the control layer and application layer, a specific entry format may not contain the necessary information in detection. Lastly, sFlow is supported by SDN switches, provides a network-wide view, is scalable and low cost, and could define customized monitoring metrics. Therefore, this work chose sFlow to collect flow in the detection.

III. THE PROPOSED METHOD

This work can be divided into four aspects: 1. build a flow collection system based on sFlow that could cover the entire network; 2. design a macro detection that could detect the occurrence of DDoS attacks; 3. design a micro detection that could distinguish attack flows from normal flows; 4. design a response strategy that could issue optimal and fine-grained rules according to the global view.

A. OVERVIEW OF WORKFLOW

The overall framework of the proposed method is shown in Fig. 1. OpenFlow switches in the data plane transmit and forward traffic, and sFlow agents are deployed on every switch to collect flow information according to the instruction of the sFlow controller (called ''Collector''). The SDN controller Floodlight and the collector software sFlow-RT (used to analyze the data returned by sFlow agents) run in the control plane. The controller provides functions such as obtaining network topology, finding routing paths, and issuing rules. The collector sends required traffic statistics to the defense application. The source-based defense application works at the application layer. It has four modules including data collection, macro detection, micro detection, and response strategy. The first module makes the data collection plan and processes the raw data acquired from sFlow-RT. The second module determines whether a DDoS attack has occurred in the network using a hypothesis testing method, and ascertains the attack target and intensity. The third module performs clustering analysis for all flow objects in current traffic based on an improved SOM model and distinguishes attack flows from normal flows. The fourth module generates and issues rules, and processes flash crowd [33].

The workflow is shown in Fig. 2. Firstly, the data collection module monitors and collects raw traffic statistical data in real-time through sFlow-RT. The raw data are preprocessed including symbolic feature digitization, discretization, and normalization. Then, the flows within a certain time interval are sent to the macro detection module to compute a ''collaboration'' metric. When this metric is below a preset threshold, no attack has occurred and the workflow continues to monitor the traffic. When the metric exceeds the threshold, the module determines that an attack has occurred, gets the IP address of the attack target and the flow details, and passes these information to the next module. Subsequently, the micro detection module gathers all flows in the time interval and performs clustering analysis for them based on an improved SOM model. This process is used to separate the normal flow contained in the attack traffic as much as possible, which minimizes the impact of false positive errors. At last, normal flows unrelated to the attack target are forwarded normally, and attack flows are processed by the response strategy module. Different flow table rules are generated according to the clustering result and the global network topology, and then issued to the optimal switches.

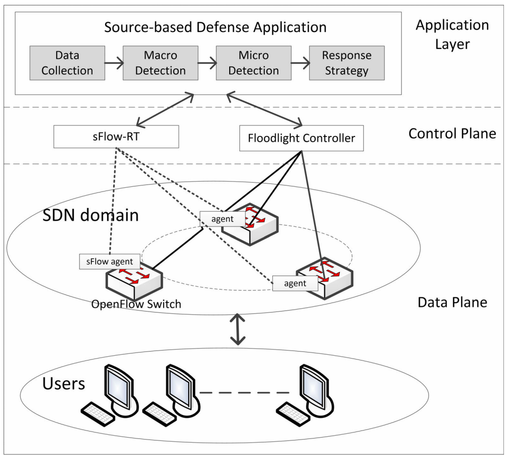


FIGURE 1. Framework of the proposed method.

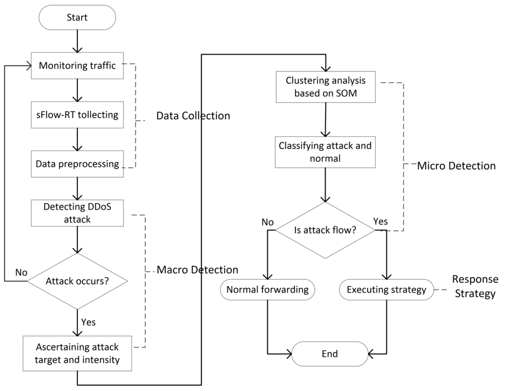


FIGURE 2. Workflow of the proposed method.

B. DATA COLLECTION BASED ON sFlow

The sFlow monitoring system is a centralized control architecture composed of multiple distributed sFlow agents and one sFlow collector. The agent is configured according to the instruction of the infrastructure device. In the experiment, the virtual switch software OpenvSwitch (OVS) is used to deploy agents. The sFlow-RT is selected as the collector to provide APIs to the upper application. It provides abundant RESTful APIs, which is called RESTflow API, to configure customized measurements and retrieve metrics. Through the RESTflow API, we can ignore the underlying details and easily implement the data collection module.

In the proposed method, we used common five tuples of IP packet fields, namely source or destination address, protocol type, and source or destination address, to define an IP flow as the detection object. The sFlow-RT is configured to collect the required flow data. Here only gives an example to illustrate the collection process, more details of configuring and using sFlow-RT could refer to its API documentation.

The RESTflow API is invoked by using HTTP methods to visit the uniform resource identifier (URI). For example, the URI "/flow/name/json" is used to manage flow definition. If we want to monitor the TCP flow and establish a cache to completely record all TCP flows, this RESTflow API is invoked through HTTP PUT method, and the arguments are set in JSON format as follows: '{"keys" : "ipsource, ipdestination, tcpsourceport, tcpdestinationport", "value" : "bytes", "log" : "true"}'. After this flow monitoring measurement is issued, the HTTP GET method is used to get real-time flow data. In the implementation, the flow data are harvested every second and maintained on the application side.

C. MACRO DETECTION BASED ON DCD

Macro detection for DDoS attacks refers to detecting the overall traffic and judging whether an attack has occurred, rather than identifying the attack traffic on more fine-grained metrics like flow or packet. The macro detection usually can be implemented based on simple volume measurement. For example, the target server can detect a DDoS flooding attack according to the deviation between the real-time input traffic load and the average value per unit time [34]. However, it is very difficult to macro-detect DDoS attacks in the source network. Because the attack flow covers up in the huge background traffic when it has not converged to a certain strength. Although researchers have proposed many macro detection methods based on metrics such as the entropy of IP source address [35]–[38], attackers can bypass such detection by imitating the distribution of normal traffic. In this work, two indispensable features are defined to characterize DDoS attacks, namely distributed collaboration and saturated attack strength. The metric used to quantify the number of concurrent flows to a same target is called distributed collaboration degree (DCD), and the total volume of these flows is called intensity.

Definition 1: Supposing IP flows in the network are expressed as set $F = \{f_1, \ldots, f_n\}$, and all nodes represented by the IP destination address of all flows are expressed as set $D = \{d_1, \ldots, d_m\}$, a destination flow (Dflow) is defined as set df_k:

$$df_k = \{f_i | f_i \in F \text{ and } f_i(\text{dstIP}) = d_k\}, \quad k = 1, 2, \ldots, m. \quad (1)$$

In equation (1), the $f_i(\text{dstIP})$ is expressed as the IP destination address of flow f_i, i.e. d_k, so a Dflow is a set of flows with the same IP destination address.

Definition 2: Supposing all Dflows in the network are expressed as set $DF = \{df_1, \ldots, df_l\}$, then for any Dflow df_i, its DCD is expressed as $\varphi(df_i)$:

$$\varphi(df_i) = |df_i| = \sum_{k=1}^{n} J(f_k(\text{dstIP})),$$

where

$$J(f_k(\text{dstIP})) = \begin{cases} 1, & \text{if } f_k(\text{dstIP}) = d_i \\ 0, & \text{if } f_k(\text{dstIP}) \neq d_i \end{cases} \quad (2)$$

Definition 3: Supposing all Dflows in the network are expressed as set $DF = \{df_1, \ldots, df_l\}$, and the rate of flow f_i is expressed as v_i,then for any Dflow df_i, its intensity is expressed as $\phi(df_i)$:

$$\phi(df_i) = \sum_{f_k \in df_i} v_k \quad (3)$$

According to equations (2) and (3), the DCD is used to measure the number of flows that concurrently access to the same target, and the intensity is used to measure the total rate of all flows in a Dflow set. The two features were proposed in our early work of an easy defense method [39]. In terms of probability, assuming the probability of a node to access a certain target at a certain time is p and this probability for all nodes obeys independent and identical distribution. For a network having n nodes, the number of nodes that concurrently access the target at time t, i.e. DCD, can be regarded as a random variable X. There is $P\{X = k\} = C_n^k p^k (1-p)^{n-k}$, namely, the random variable X follows a binomial distribution. According to Poisson theorem, when n is large, X approximately obeys the Poisson distribution $\pi(\lambda)$, which has only one unknown parameter, and λ can be estimated as the sample mean through maximum likelihood estimation. Therefore, in theory, the X that exceeds the estimated sample mean to a certain degree can be regarded as a small probability event, which means an abnormal value related to an attack. Generally speaking, high-concurrency access will only occur when a large number of nodes are organized, such as Botnet-based DDoS attacks and flash crowds. Accordingly, it is feasible to detect DDoS attacks based on the DCD feature. When the DCD of a Dflow shows an abnormal value, this Dflow can be determined as a possible DDoS attack. How to process flash crowds will be discussed in the response strategy. However, in actual situations, the assumption is unrealistic, and the user visit behavior does not ideally obey the binomial distribution or approximate the Poisson distribution. It is infeasible to figure out what kind of distribution the DCD obeys. As a result, we resort to Chebyshev's inequality theorem. It is simple and feasible to estimate the probability distribution based on this theorem.

Theorem 1: Supposing that the random variable X has mathematical expectation $E(X) = \mu$ and variance $D(X) = \sigma^2$, then $\forall \varepsilon > 0$, there is Chebyshev's inequality as follows:

$$P\{|X - \mu| \geqslant \varepsilon\} \leqslant \frac{\sigma^2}{\varepsilon^2} \quad (4)$$

Let $\varepsilon = k\sigma$, we can get another representation of (4):

$$P\{|X - \mu| \geqslant k\sigma\} \leqslant \frac{1}{k^2} \quad (5)$$

According to equation (5), we can get:

$$P\{X \geqslant \mu + k\sigma \ or \ X \leqslant \mu - k\sigma\} \leqslant \frac{1}{k^2} \quad (6)$$

As the random variable φ is positive, only abnormally large values of φ are worth of attention. Hence, the detection threshold is computed only based on the upper bound of equation (6). For example, when the confidence level is set to $1-\alpha = 0.8$, there is $1/k^2 = \alpha$, and $k = 2.24$, so the threshold $\theta = \lceil \mu + k\sigma \rceil = \lceil \mu + 2.24\sigma \rceil$. It means that, when current DCD exceeds the normal expectation of 2.24 times normal standard deviations, there is an 80% certainty that the corresponding Dflow can be determined as an attack. In the experiment, all Dflows in time interval T under the normal situation are used as observation samples, and the normal μ and σ are estimated based on the sample mean and sample variance of these samples.

Supposing that the number of Dflows with the DCD φ being i is n_i, then $P\{\varphi = i\} = p_i = \frac{n_i}{N} = \frac{n_i}{\sum_k n_k}$, thus:

$$\widehat{\mu} = \overline{\varphi} = \sum_i i \cdot p_i \tag{7}$$

$$\widehat{\sigma} = \sqrt{\frac{N}{N-1} \cdot \sum_i (i - \widehat{\mu})^2 p_i} \tag{8}$$

According to equations (7) and (8), the threshold θ can be set as $\theta = \lceil \widehat{\mu} + k\widehat{\sigma} \rceil$. The macro detection algorithm is described in algorithm 1. The flowchart of this algorithm is shown in Fig. 3. According to the algorithm, the macro detection is periodically executed every T_1 seconds, but the collection is executed every second, so the detection targets all Dflows in the time window T_1. When a Dflow is judged as an attack, the information of all flows in this time window, i.e. set F, will be sent to the next module. The attack intensity ϕ will also be used as an important decision basis in the response strategy. The threshold θ is computed and updated based on normal traffic. The time window parameter T_1 is set according to the experience.

Algorithm 1 Macro Detection

```
Input: $F \leftarrow \emptyset$, $D \leftarrow \emptyset$, $DF \leftarrow \emptyset$
for every seconds do
    read flow statistics and add all IP flows $f_i$ to $F$
end for
for every $T_1$ seconds do
    for every $f_i$ in $F$ do
        $d_i = f_i(\text{srcIP})$
        if $d_i \notin D$ then
            add $d_i$ to $D$
            add $df_{d_i} = \emptyset$ to $DF$
        end if
        add $f_i$ to $df_{d_i}$
    end for
    for every $df_{d_i} \in DF$ do
        if $\varphi(df_{d_i}) > \theta$ then
            Dflow $df_{d_i}$ is attack
        else
            Dflow $df_{d_i}$ is normal
        end if
    end for
end for
```

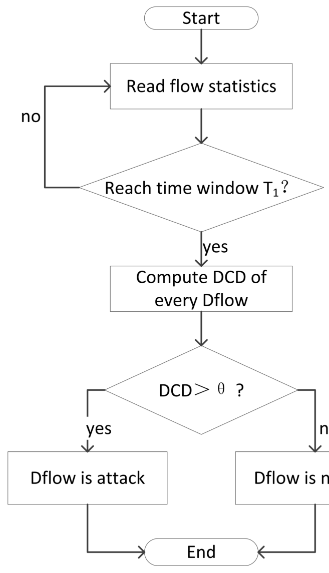


FIGURE 3. Flowchart of macro detection.

From the perspective of the principle of DDoS attacks based on Botnets, it is very hard to achieve the ideal attack effect when there are too few zombie nodes or the total attack intensity is too small. Therefore, the DCD and intensity are the most essential characteristics of DDoS attacks. There is an almost unbridgeable gap between normal and attack behavior on the characteristics. A simpler and ruder approach is to directly filter the flows whose DCD and intensity are both abnormal. When there are only a small number of zombies in a source work, the proposed method may not be able to effectively detect the attack. In this situation, it requires a more fine-grained and deeper detection mechanism to deal with, such as deep packet inspection (DPI), but resorting to destination-based or network-based mechanisms is a much more economical and effective way.

D. MICRO DETECTION BASED ON DGSOM

When the macro detection alarms, it is necessary to further analyze all flows in the attack Dflow to distinguish the real attack flow from the false alarmed normal flow. If only simply filtering all flows in a suspect Dflow without making any distinction, there will be a high possibility to affect some normal users. Harming user experience inside the domain while protecting targets outside the domain will greatly reduce the willingness of deploying source-based defense mechanisms. Therefore, for the source-defending scenario, the trade-off between low false alarm rate and high detection rate (also called recall or sensitivity) will tend to give higher priority to the former, and be more tolerant to the latter.

The complexity and variability of network traffic makes it difficult to simply regard DDoS attack detection as a classification problem, because not only is there a lack of effective labeled data for training the detection model, but it is also very difficult to get a detection model with enough generalization performance to cope with changing traffic. Therefore, the proposed method conducts traffic analysis based on a clustering algorithm called SOM, which is an effective and feasible unsupervised machine learning method. More details about SOM and how it is used in DDoS flooding attack detection can be seen in [21].

The orderly topology of SOM can better reflect the orderly distribution of the inherent attributes in data than other clustering algorithms, such as k-means. As the different traffic in each attack event will cause the clustering result to be quite different, a static SOM model tuned based on fixed training data cannot be competent to continuously analyze the traffic in different attack events. Therefore, it is necessary to perform real-time clustering, or clustering based on the model tuned by the most recent historical traffic. To solve this problem, an improved algorithm called dynamic generative SOM (DGSOM) is proposed based on the basic SOM. The algorithm does not need to preset the arrangement and size of the topology. It dynamically adjusts the topology and weights according to the data representation effect quantified by some metrics, which will be introduced later. Meanwhile, the algorithm can continuously update the detection model, when it is needed, through a pre-train process based on the latest historical samples.

For clustering the flows, ten features are defined as inputs of the DGSOM model, which is shown in Table 1. As numbered in the table, each flow is expressed as a vector $f_i = [x_1, x_2, \ldots, x_{10}]$. The first four features are directly extracted from the packet header filed, and the last six features are calculated based on the number of bytes and the number of packets in a flow. When calculating the last six features, the concept of pair flow is involved in the formula, which represents the communication flows between two nodes.

TABLE 1. Flow-based features: description and calculation.

No.	Feature	Description	Calculation
1	srcIP	source IP address	-
2	dstIP	destination IP address	-
3	srcPort	source port	-
4	dstPort	destination port	-
5	BR	byte rate	$\frac{\text{totalSrcBytes}}{\text{totalBytes}}$
6	PR	packet rate	$\frac{\text{totalSrcPackets}}{\text{totalPakcets}}$
7	srcBPP	source bytes per packet	$\frac{\text{totalSrcBytes}}{\text{totalSrcPackets}}$
8	dstBPP	destination bytes per packet	$\frac{\text{totalDstBytes}}{\text{totalDstPackets}}$
9	TB	total bytes	$totalBytes$
10	TP	total packets	$totalPackets$

Firstly, we define the indicator that is used to quantitatively evaluate the clustering effect [40], [41].

Definition 4: Supposing that the weight vector corresponding to neuron i in SOM is w_i, and all input vectors f_j mapped to neuron i constitute a cluster set C_i, then mean quantization error (mqe) used to quantitatively evaluate the clustering effect of neuron i is expressed as:

$$mqe_i = \frac{1}{n_i} \sum_{f_j \in C_i} \|w_i - f_j\|, \quad n_i = |C_i|, \ C_i \neq \emptyset. \tag{9}$$

According to equation (9), the quantitative indicator of the global clustering effect of SOM for all data can be expressed as:

$$MQE = \sum_{C_i \neq \emptyset} mqe_i. \tag{10}$$

The smaller the MQE, the tighter the distribution of data points in their respective clusters. Expanding the size of SOM topology can reduce the MQE, but the unlimited expansion of the topology violates the goal of clustering, and also increases the complexity of the model, which is not conducive to the rapid generation of the detection model. Therefore, there is a trade-off between MQE and topology size, that is, a regularization term needs to be introduced to control the complexity of the model. Before constructing the regularization term, in order to map the dense area of data points in the input space to a larger topological space, quantization error (qe) is used instead of mqe as the indicator to evaluate the clustering effect in the process of generating and training SOM. The qe and the corresponding global quantitative indicator QE are expressed as:

$$qe_i = \sum_{f_j \in C_i} \|w_i - f_j\|, \quad C_i \neq \emptyset. \tag{11}$$

$$QE = \sum_{C_i \neq \emptyset} qe_i. \tag{12}$$

According to equation (11), when the qe of a certain neuron is high, the clustering effect of the model is poor. This kind of poor is reflected in two aspects or both: either a non-uniform part of the input space corresponding to the neuron has a large number of different points, or the neuron represents at least a considerable number of points in a certain uniform part of the input space. When using qe or mqe to control the generation of SOM, the former pays more attention to the finer-grained characterization of the particle-intensive areas of the data, while the latter emphasizes on characterizing the overall uniformity of the data. This is the reason why quantitative indicator qe is used to construct the loss function. Choosing the L_2 norm of the weight vector as the regularization term to control the model complexity, the loss function can be expressed as:

$$L(W) = \sum_{C_i \neq \emptyset} qe_i + \lambda(n) \sum_i n_i \|w_i\| \tag{13}$$

Then, the problem of how to train a SOM model is transformed into an optimization problem, i.e. minimizing the loss function $L(W)$. In equation (13), W is the matrix of all weight vectors, parameter $\lambda(n) = \lambda_0 \sqrt{n}$ is used to adjust the trade-off between the two terms on the right side of the equation, where n is the total number of neurons and λ_0 is the scale factor. The loss function in (13) makes up for the disadvantage of the lack of objective loss function of the basic SOM algorithm. Since it is difficult to solve the minimization problem through gradient optimization, a heuristic algorithm is designed to find the optimal model. The DGSOM algorithm is described in algorithm 2.

Algorithm 2 DGSOM

```
Input: $F = \{f_1, \ldots, f_n\}$
set initial size of SOM model $M^0$ to be $2\times2$ units
set initial weight vectors $W^0 = \{w_1^0, w_2^0, w_3^0, w_4^0\}$ to random values
train the model $M^0$ according to basic SOM learning algorithm
$Q = \emptyset, L = +\infty, ter = 0$, Can $= \emptyset$
compute $qe^0$ for every unit
compute loss function $L^0$
$Q = \{qe_k^0|$k is all units$\}, L = L^0$
for $i = 1 : ITE$ do
    $s = argmax_k qe_k^{i-1}$
    $N = \{w_j^{i-1}|$unit j is the neighbor of unit s$\}$
    $d = argmax_k \|w_k^{i-1} - w_s^{i-1}\|$ subject to $k \in N$
    insert a row or column units between unit s and d with the weight vector of every unit being the mean of its respective neighbors
    retrain generated model $M^i$ using basic SOM learning algorithm
    compute $qe^i$ for every unit
    compute cost function $L^i$
    if $L^i \geqslant L^{i-1}$ then
        $ter = ter + 1$
        if $i \geqslant 2$ and $L^{i-2} < L^{i-1} < L$ then
            add $(L^{i-1}, M^{i-1})$ to Can
        end if
        if $ter \geqslant$ TER then
            add $(L^{i-\text{TER}}, M^{i-\text{TER}})$ to Can
            break
        end if
    else
        $ter = 0$
    end if
    $Q = \{qe_k^i|$k is all units$\}, L = L^i$
end for
$M = argmin_{M^j} L^j$ subject to $(L^j, M^j) \in$ Can
return $M$
```

The simplified flowchart of DGSOM is shown in Fig. 4. The details of the basic SOM learning algorithm can be seen in [42]. According to the algorithm, the process of dynamically generating SOM starts with a 2×2 topology, and then trains the initial model using the basic SOM learning algorithm. After this, the *qe* of every neuron is computed and the neuron with the largest *qe* is labeled as s. The neighbor neuron that is most dissimilar to neuron s is also selected according to the distance between s and its neighbor neurons, which is labeled as d. After determining the positions of neuron s and d, a row or column of neurons are inserted between s and d, and the weight vector of the newly added neurons is set to the mean of the weight vector of the old neighbor neurons [40]. After getting the expanded SOM model, its weight vectors are regarded as the new initialization, the parameters

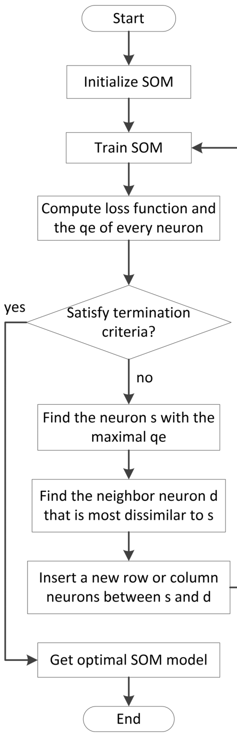

Start
Initialize SOM
Train SOM
Compute loss function and the qe of every neuron
Satisfy termination criteria?
yes
no
Find the neuron s with the maximal qe
Find the neighbor neuron d that is most dissimilar to s
Insert a new row or column neurons between s and d
Get optimal SOM model
End

FIGURE 4. Flowchart of DGSOM.

such as learning rate and neighborhood width are reset, and then the new map is retrained again. As the retraining process inherits the data representation of the old SOM and performs a more in-depth analysis on the clusters with the worst clustering effect, the newly generated SOM model can depict the distribution characteristics of the data in more detail. The process of expanding and retraining is repeated iteratively until the target loss function $L(W)$ does not continuously decrease for TER times or reaches the maximum iteration ITE. The parameters TER and ITE control the algorithm termination, and TER is the early stopping criterion. In the experiment, TER is set to a suitable small positive integer, for example, TER $= 3$, and ITE is set according to the upper limit of the preset SOM topology size. Finally, the optimal SOM model is selected from the candidate set generated in the iterative process.

Since the DGSOM algorithm needs to iteratively train the SOM model many times, in order to avoid the training process taking too much time and causing detection delay, the algorithm is divided into two parts on the time axis in actual operation, namely pre-generation process P_1 and generation process P_2, which is shown in Fig. 5. Process P_1 periodically executes DGSOM based on the historical flow data $F_{history}$ in the sliding time window T_2 to generate the latest SOM model. For example, when the macro detection alarms that an attack has occurred at t_1, process P_2 uses the SOM model generated by process P_1 as the initial state, and continues to train it based on current flow data $F_{current}$ in window T_1 to get the final model. The response strategy module will make decisions based on the final model. Since the process P_2 is

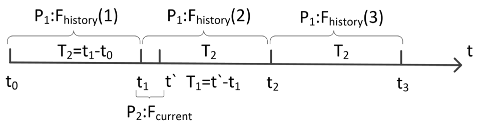


FIGURE 5. The diagram of pre-generation and generation on time axis.

trained based on the pre-generated model in P_1, the optimal topology has been determined and all weight vectors have been preliminarily tuned, the time spent on training process in detection will be saved a lot. When the traffic is small and the amount of flows is not very large, the pre-generation process can be omitted, and the training process is directly performed based on the current flow.

When the macro detection determines a Dflow as an attack, denoted as df_{att}, the attack target IP address, denoted as d_{att}, is the destination address of this Dflow. Supposing $F_{current} = \{f_1, f_2, \ldots, f_N\}$, and the clustering result given by DGSOM is expressed as set $\{C_i | C_i = \{f_1, \ldots, f_{n_i}\}\}$, where $N = \sum_i n_i$. The flow whose destination IP is d_{att} in cluster C_i is called attack flow that hits neuron u_i, and is gathered as hitting set $H_i = \{f_j | f_j(\text{dstIP}) = d_{att}\ and\ f_j \in C_i\}$. The number of flows $h_i = |H|$ is called hitting number of neuron u_i. After calculating the hitting number of all neurons and sorting them, the set H_i with the largest h_i is regarded as the most suspicious attack set, and the corresponding neuron and hitting set are denoted as u_{r0} and H_{r0}, which means all attack flows in set H_{r0} will be executed with a response strategy of level 0. For other neurons whose hitting number is not zero, after calculating the neighborhood distance with neuron u_{r0}, which is defined as $d = \|w - w_{r0}\|$, the maximum distance is selected and divided into K equal intervals, and the corresponding hitting sets in each interval are merged to determine the final hitting sets $H_{r_1}, H_{r_2}, \ldots, H_{r_K}$, as illustrated in Fig. 6. The subscript of each hitting set indicates the respective response level. Response strategy module will generate and issue rules based on the level.

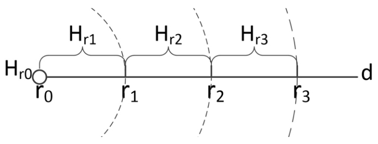


FIGURE 6. The illustration of dividing and merging hitting set.

E. RESPONSE STRATEGY BASED ON SDN

The response strategy module issues rules to the designated switch according to the detection result and the network global topology. Specifically, this module deals with three problems: how to make differentiated rules based on response levels; how to merge rules and determine their deployment location; how to cope with normal flash crowds.

For the first problem, according to the response level divided by the micro detection module, it needs to form K+1 different rules with different severity to handle the attack flows. The most severe rule directly filters the attack flow with response level 0, which is implemented by issuing flow entries with drop action. For the attack flows with other response levels, rules with different quantified queues are used to perform different QoS through issuing flow entries with queue action. The size of every queue changes gradually according to the response level. For example, if the response level of a flow is k and its attack intensity is ϕ, the queue bandwidth is $k/K \cdot \phi$.

For the second problem, merging rules and determining the deployment location are two sub-problems that affect each other. The purpose is to minimize the number of issued flow entries without conflicts and issue flow entries as close to the attack source as possible. The former aims to save valuable switch flow table resources, as some commercial OpenFlow switches can only support no more than 4K flow entries. The latter aims to avoid the waste of bandwidth caused by attack traffic.

When it comes to rules, a rule is usually expressed in the format of "match/mask". A mask field of 1 means the corresponding bit of the match field must be accurately equal to the related header field. A 0 mask is a wildcard, which means the corresponding match bit can match any value. For example, 0010/0011 means the first two bits of the match field can match any value, but the last two bits must exactly match 10.

Based on the wildcard, a rule merging method is proposed through iterative tree search. The steps are as follows and an illustration is shown in Fig. 7. Suppose the flows with the same flow entry action and deployment location are gathered as set $H_i = \{f_{i1}, f_{i2}, \ldots\}, i = 0, 1, 2, \ldots$.

Step 1: generate initial rules in the format of "full match/all 1 mask" for all flows in every set H according to their IP 5-tuples header (other header fields are not discussed here), denoted as $ru_i = \{ru_{i1}, ru_{i2}, \ldots\}$.

Step 2: sort all sets ru in descending order according to the number of rules, i.e. $|ru|$. For every set ru_i, respectively generate a rule that can include all initial rules in it and has the mask with the heaviest Hamming weight, denoted as RU_{i1}. Such rules are called candidate merged rules (CMR), denoted as set $RU_{i1}, RU_{i2}, \ldots$. The final merged rules (FMR) set for every ru_i is denoted as RU_i. In this step, all initial FMR sets are marked as empty.

Step 3: start from the first ru, match its CMR rules with the rules in the other FMR sets in sequence (match with the corresponding initial rule set if an FMR set is empty).

Step 4: if there is a rule matching successfully, split this RU_{ij} at the match bit corresponding to the first 0 mask bit to two new sub-rules, the bit corresponding to the first old 0 bit is changed to 1 in both of new sub-rules.

Step 5: judge the two split rules respectively, if one intersects with the corresponding initial rule set (ensure rule inclusion) and does not intersect with any other CMR or FMR set (avoid rule conflict), add it to the corresponding FMR set. Delete the rule if it does not intersect with the corresponding initial rule set. For any other cases, go to step 3 and continues to split the rule until it can not be split.

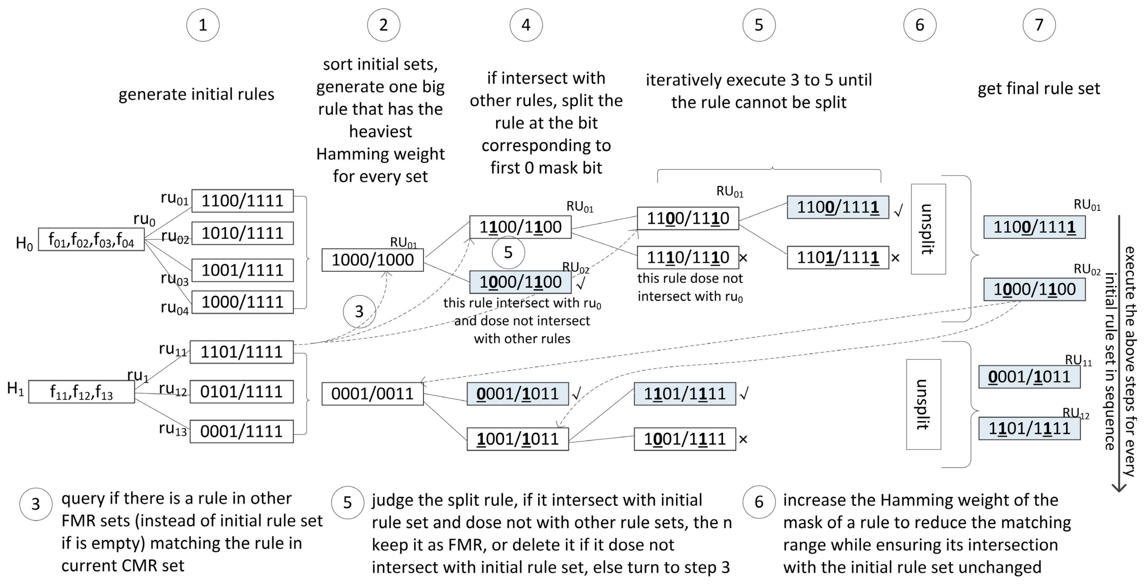

1
generate initial rules
2
sort initial sets, generate one big rule that has the heaviest Hamming weight for every set
4
if intersect with other rules, split the rule at the bit corresponding to first 0 mask bit
5
iteratively execute 3 to 5 until the rule cannot be split
6
7
get final rule set
H0 f01,f02,f03,f04
ru0
ru01 1100/1111
ru02 1010/1111
ru03 1001/1111
ru04 1000/1111
RU01 1000/1000
RU01 1100/1100
RU02 1000/1100 √
this rule intersect with ru0 and dose not intersect with other rules
RU01 1100/1110
1110/1110 ×
this rule dose not intersect with ru0
1100/1111 √
1101/1111 ×
unsplit
RU01 1100/1111
RU02 1000/1100
H1 f11,f12,f13
ru1
ru11 1101/1111
ru12 0101/1111
ru13 0001/1111
0001/0011
0001/1011 √
1001/1011
1101/1111 √
1001/1111 ×
unsplit
RU11 0001/1011
RU12 1101/1111
execute the above steps for every initial rule set in sequence
3 query if there is a rule in other FMR sets (instead of initial rule set if is empty) matching the rule in current CMR set
5 judge the split rule, if it intersect with initial rule set and dose not with other rule sets, the n keep it as FMR, or delete it if it dose not intersect with initial rule set, else turn to step 3
6 increase the Hamming weight of the mask of a rule to reduce the matching range while ensuring its intersection with the initial rule set unchanged

FIGURE 7. An illustration of rule merging algorithm.

Step 6: for every rule in the FMR set, generate the heaviest mask instead of the old mask according to its intersection with the corresponding initial rule set. This step is used to minimize the impact of the rule with too many wildcards on unknown flows.

Step 7: get the FMR set for the first *ru*, start from the next *ru* and repeat step 3 to 7 until getting all FMR sets for every set H_i.

It can be seen from Fig. 7 that 7 rules before merging are reduced to only 4 rules after merging. When preferring to save bandwidth, the default deployment location of every rule is the access switch corresponding to the source node. For the rules at the same location, they are first gathered to set H according to the response level before being merged. When preferring to save flow table resources, the rules with the same response level are firstly merged before deciding the location. The final rules are deployed on the switch that is effective and the closest to the attack source. Taking the simple tree topology in Fig. 8 as an example, and supposing the attack target is a node outside the domain, when the source IP match field of a rule RU includes nodes $\{n_1, n_2, n_5, n_8\}$, this rule will be deployed on s_1 switch. But, when the source IP match filed includes $\{n_1, n_2, n_{12}, n_{16}\}$, the rule needs to be deployed on s_3 switch to cover the related nodes and close to the attack source.

For the last problem, as flash crowds and DDoS attacks have the same effect and both appear as high concurrent access to the same target within a certain period, it is very difficult to distinguish between them based on the macro detection. This work used a simple feedback method to deal with flash crowds. The basic hypothesis is: although flash crowds and DDoS attacks have the same effect, the feedback after implementing the response strategy is very different. Normal users in flash crowds usually do not maliciously and

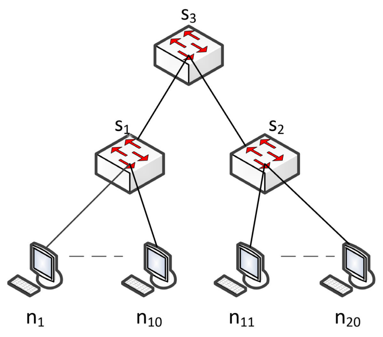


FIGURE 8. A simple tree topology network.

continuously send a large number of service requests, and they will respond by low-frequency refreshing or just waiting when perceiving the inaccessibility or QoS degradation. However, malicious nodes, i.e. zombies controlled by the attacker, will always continuously send a large number of requests to the target to achieve the attack purpose. Therefore, after deploying flow entries, the DCD metrics perceived from the access switch of the two phenomena will have a significant difference. The DCD of flash crowds will decrease to a normal level, while the DCD of DDoS attacks will remain at a high level. If the zombie program also imitates the normal behavior to bypass this method, the botnet must inevitably reduce the number of requests, which mitigates the threat of DDoS attacks and still achieves the defense purpose to some extent.

Supposing the default priority is to save flow table resources and the K sets received from micro detection are expressed as $H_{r_i} = \{f_{i1}, f_{i2}, \ldots, f_{in_i}, \}, i = 0, 1, \ldots, K$, network topology is expressed as graph G(V,E), where V represents all switches/nodes and E represent all links, the response strategy is described in algorithm 3.

Algorithm 3 Response Strategy

```
Input: $H_{r_0}, H_{r_1}, \ldots, H_{r_K}, G(V, E)$
{merge rules for every set}
for every set $H_{r_i}$ do
    merge rules in set $H_{r_i}$ to get FMR sets $RU_i = \{RU_{i1}, \ldots, RU_{in_i}\}$
end for
{add actions to every FMR}
for for every FMR $RU_{ij}$ do
    if $i = 0$ then
        add drop action to rule $RU_{0j}$
    else
        add queue action with the size as $\frac{i}{K}\sum_{ru_{ik}\in RU_{ij}}\phi$ to rule $RU_{ij}$
    end if
end for
{find the optimal location to issue rules}
for every rule $RU_{ij}$ do
    node IP set
$IPset = \{IP|IP\in V \textit{ and } IP\in(\text{src IP match field of}RU_{ij}\}$
    path set
$P = \{\text{every path from every IP to } IP_{attack}|IP\in\text{IPset}\}$
    find switch node set $s = \{\forall\text{path}\in P\ \exists\ s\in\text{path}\}$
    find optimal switch node
$s_{op} = \{\forall s_{op}\in s\ \nexists\text{path}(IP\to s_{op}) = \text{path}(IP\to s'\to s_{op})\}$
    issue rule $RU_{ij}$ to optimal switch $s_{op}$
end for
{feedback judgement}
if $\varphi(IP_{attack})$ returns to normal threshold in $T_3$ seconds then
    abandon all issued rules
    add $IP_{attack}$ to whitelist for $T_4$ seconds
else if $\varphi(IP_{attack})$ returns to normal threshold then
    abandon all issued rules
end if
```

The simplified flowchart is shown in Fig. 9. After rules are issued, the response strategy will calculate the DCD of the attack Dflow based on the information fed back from the access switch. When the DCD falls within the normal threshold in T_3, the corresponding Dflow will be treated as a flash crowd, the issued rules will be discarded, and the target IP will be added to a whitelist for time T_4. Time parameter T_3 is set according to the experience. Parameter T_4 is used to prevent the real attack target from being wrongly put in the whitelist when bots imitating the flash crowd, and it can be set as the average duration of a connection. In the case DCD returns to normal caused by the stop of the attack, the strategy only discards the issued flow entries.

IV. EXPERIMENTS AND RESULTS

In this section, the proposed method will be validated in a simulation environment. The traffic data used in the experiment includes two parts: the traffic captured from our campus network and the ISCX-IDS2012 traffic data [43], which are labeled as D1 and D2 respectively. The ISCX-IDS2012 is an open intrusion detection evaluation dataset published by the University of New Brunswick. It provides real network traffic samples and has been well labeled. This data captures the traffic in a week and is divided into seven datasets by day. However, as only the dataset on Thursday contains normal traffic and IRC-based DDoS attack traffic, we only used the Thursday traffic in this paper. The campus network traffic is captured from our university, and it is regarded as normal traffic. The composition of datasets D1 and D2 is shown in Table 2. The experiment is implemented on a server with 8 CPUs and 32 GB RAM, mininet is used to simulate SDN, the controller software is Floodlight, the virtual switch software is OVS, and the traffic replay software is tcpreplay.

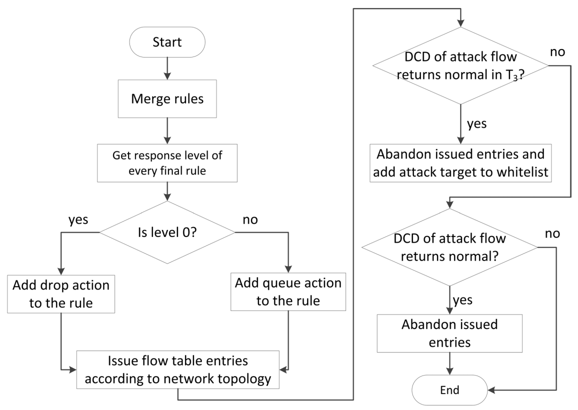

Start
Merge rules
Get response level of every final rule
Is level 0?
yes
no
Add drop action to the rule
Add queue action to the rule
Issue flow table entries according to network topology
DCD of attack flow returns normal in T3?
no
yes
Abandon issued entries and add attack target to whitelist
DCD of attack flow returns normal?
no
yes
Abandon issued entries
End

FIGURE 9. Flowchart of response strategy.

A. VALIDATION OF MACRO DETECTION

In order to validate the effectiveness of the macro detection, dataset D1 is replayed on the simulation network to observe the change of Dflow DCD in a real source network. The simulation network is implemented by mininet. Its topology is shown in Fig. 10. For the facilitation of replaying the traffic on each simulation node, every access switch in Fig. 10 only has one node, but the actual number of nodes can be seen on the visual topology (as shown in Fig. 11) provided by Floodlight.

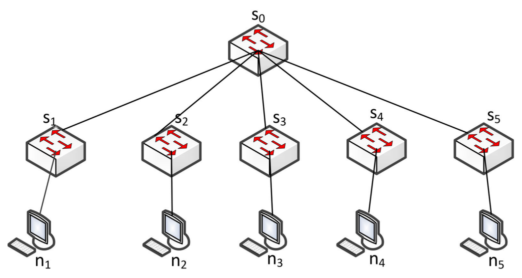


FIGURE 10. Topology of simulation network.

Setting the parameter $T_1 = 60s$, the change of DCD on D1 is shown in Fig. 12. In the figure, the x-axis represents the

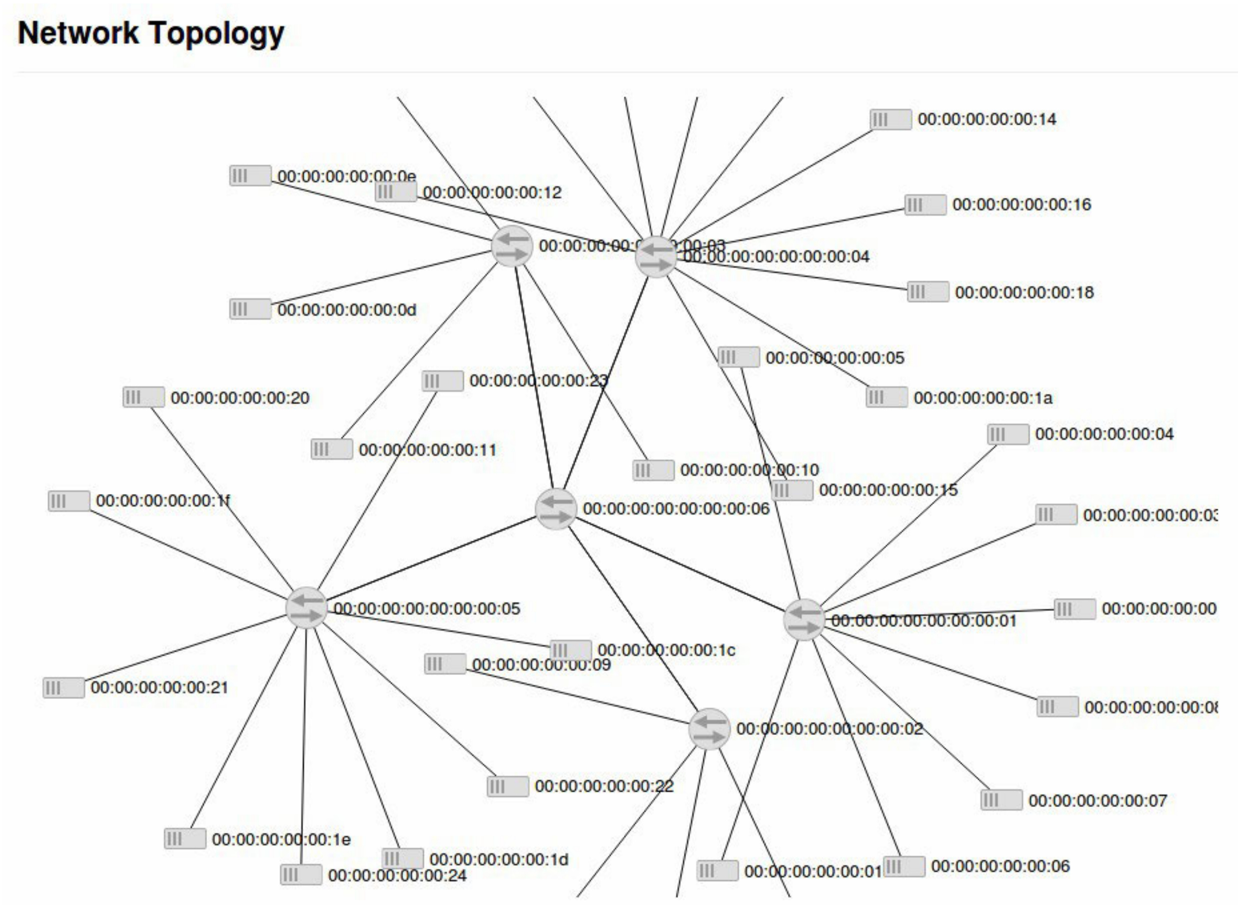


FIGURE 11. Visual topology provided by Floodlight.

TABLE 2. The composition of dataset D1 and D2.

data	D1	D2
sample source	campus network	ISCX-IDS2012Jun15
before processing	11,159 TCP flows	534,320 normal flows
	19,747 UDP flows	37,378 attack flows
after processing	11,159 TCP flows	14,000 normal flows
		35,073 attack flows

value of DCD, the y-axis is the time axis and has a total of 60 time units, and the z-axis represents the number of Dflows with a certain DCD. For example, for the first detection period, there are 28 Dflows with DCD being 1, 2 Dflows with DCD being 2, and 0 Dflows with DCD being more than 2. It can be seen from the distribution in Fig. 12 that most nodes in this period are accessing different targets, and there is no large-scale concurrent access to the same target. Among the 36 active nodes in the network, the highest DCD is 5, which indicates that an abnormal large DCD is indeed an event with small probability. Using the traffic data in this period as a baseline, and setting the confidence level to be $1-\alpha = 0.99$, the threshold is calculated to $\theta = 7$. Therefore, it has 99% confidence to judge the Dflow with the DCD bigger than 7 as an attack.

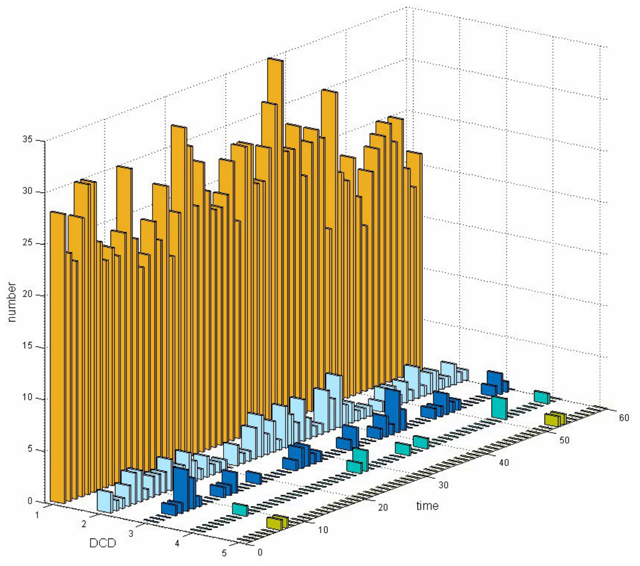


FIGURE 12. DCD on D1 in one hour.

Continuing to observe the change of DCD for eight hours and using them as the baseline data, and setting the confidence level to 0.8, 0.9, 0.95, 0.99, 0.995, the change of the

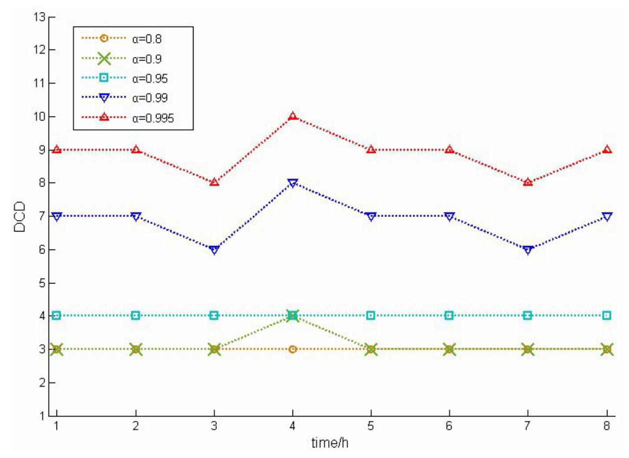


FIGURE 13. Threshold change in eight hours.

threshold is shown in Fig. 13. From the figure, the largest threshold is 10 at the confidence level of 0.995, and is 3 at 0.8. This further indicates that the normal DCD in a source network, at least in this observation network, being abnormal large is a very rare event. Therefore, using the metric DCD to implement an abnormal detection can detect the DDoS attack. As most Dflows in D1 only have 1 DCD, the threshold calculated according to the baseline data is very small, and the detection is a little sensitive. For most Botnet-based DDoS attacks, the DCD is much larger. Hence, for this simple network, the threshold can be manually set to a larger value to avoid some false positive errors.

B. VALIDATION OF MICRO DETECTION

To validate the effectiveness of the micro detection, it is necessary to evaluate the effect of distinguishing the false alarmed normal flow from the real attack flow, and quantitatively evaluate the impact of response actions on normal flows. According to the response level and the corresponding response action, the flow with response level 0 (drop action) can be regarded as predicated attack flow, and the flow with non-zero level can be regarded as predicted normal flow. In this way, a customized confusion matrix can be defined as shown in Table 3. It should be noted that this confusion matrix is different from the standard confusion matrix. The customized confusion matrix is used to evaluate the detection performance of the micro detection on reclassifying the suspected attack flow, but the standard one is used to evaluate the overall detection performance. According to the confusion matrix, four commonly used evaluation metrics can be calculated, namely accuracy (Acc), precision (Pre), detection rate (DR), and false alarm rate (FAR). The calculation can refer to our another work in [44].

As the purpose of micro detection is to separate the false alarmed normal flow from the real attack flow as much as possible, the detection performance should first consider lower FAR and then consider higher DR. Only these two metrics are given in the experiment. The DR is the ratio of real attack flows with level 0 to all real attack flows, and the FAR is the ratio of real normal flows with level 0 to all real normal flows. For the quantitative evaluation of response level division, only the impact on real normal flows is considered, which means the optimal division should assign the highest response level to real normal flows. The higher the level, the lower the impact on normal flows. The impact of not filtering attack flows is ignored by default. Because in a source network, not filtering some real attack flows but only rate-limiting them will not affect the local network and still mitigate the attack. Therefore, the quantitative evaluation metric of response level division is defined as false alarm impact (FAI) as follows:

TABLE 3. Customized confusion matrix.

		Predicted Class (Response Level)	
		Attack (0 level)	Normal (¬0 level)
Actual Class	Attack	True Positive (TP)	False Negative (FN)
	Normal	False Positive (FP)	True Negative (TN)

Definition 5: Supposing the impact of assigning a normal flow f with response level K is unit 1, and the impact of assigning level k is computed as 2^{K-k}, the FAI is computed as:

$$\mathrm{FAI} = \sum_{f \in \mathrm{normal}} f_{r_k} \cdot 2^{K-k}, \tag{14}$$

where f_{r_k} is the number of real normal flows with response level k.

The smaller the FAI, the better the response level division. In the ideal case, all real normal flows that are falsely judged as the attack in macro detection are assigned to the highest response level, which means they can maintain the original communication rate and continue to access the target.

D1 and D2 are taken as the experimental data. The preset scenario conditions include: 1. the upper limit of the number of active nodes in the network is 2,000; 2. do not consider the case of using forged source IP, which can be regarded as increasing the number of active nodes; 3. the average number of real normal flows is 3,000 and the proportion of a suspected attack Dflow containing the real normal flow is fixed at 20%. Since all DDoS attack flows in D2 are HTTP Web traffic targeted to "192.168.5.122:80", the target IP of the attack Dflow is "192.168.5.122". The threshold is set to 49, the experiment starts from an attack with DCD being 50, and then the DCD is gradually increased from 100 to the upper bound 2,000 in 100 steps. Accordingly, 21 groups of experiments can be carried out. The dataset of each experiment is generated as follows: randomly sample 3,000 normal flows from D1 and label them as normal; randomly sample a specified number of normal and attack flows from D2 according to the proportion and corresponding DCD, and label them as attack; randomly mix all samples to get the experimental dataset. The composition of these 21 groups of data is shown in Table 4. In this table, $i = 0.5, 1, 2, 3, \ldots, 20$. The topology limitation of SOM is set to 15×15, and the maximum iteration parameter ITE should be 27. Set the early stopping parameter

TABLE 4. Composition of the synthetic experimental data.

Label	Normal	Attack	
Real Label	real normal flow	real attack flow	real normal flow
Data Source	D1 normal flow	D2 attack flow	D2 normal flow
Number	3000	$100 \cdot i \cdot 80\%$	$100 \cdot i \cdot 20\%$

TER = 3, parameter $\lambda_0 = 0.008$, and the maximum response level K = 4.

The experiment with $i = 20$ is chosen as an illustration to observe the changes of related indicators of DGSOM algorithm during the training iteration process, and compare them with k-means and k-medoids algorithms. In this experiment, TER is set to an integer bigger than ITE to observe the entire iteration. In the comparison, the number of clusters of the two k-algorithms is equal to the number of corresponding neurons.

Fig. 14 shows the loss function during the entire iteration when $\lambda_0 = 0.008$. It can be seen that the loss function of the DGSOM and the k-algorithms firstly decreases and then increases with the increasing of model complexity. All of them have the global minimum during the entire iteration, which means the regularized loss function in (13) can guide the DGSOM algorithm to generate the optimal model. If the early stopping strategy is executed, i.e. TER = 3, according to Fig. 14, the DGSOM algorithm will stop at the 10th iteration, and the optimal SOM topology is 4×6 generated in the 7th iteration. The number of clusters of the k-algorithms used for comparison is 24. To observe the change of *QE* during the iteration, Fig. 15 shows the loss function when $\lambda_0 = 0$, which means ignoring the second part on the right side of equation (13). It can be seen that the *QE* of all algorithms is gradually converging, which means expanding the SOM topology or increasing the number of clusters will not significantly improve the clustering effect when the model complexity increases to a certain degree. Combing the results in Fig. 14 and Fig. 15, it can be seen that, in the case of the same model complexity, the loss function of DGSOM is overall slightly lower than the k-medoids, and it is lower than the k-means only when the SOM topology is large.

Fig. 16 and Fig. 17 show the changes of FAR and FAI during the entire iteration. Firstly, horizontally comparing the FAR and FAI of the DGSOM algorithm, both FAR and FAI firstly gradually decline and then fluctuate in a small range with the SOM topology expanding, which means a larger SOM topology is beneficial to strip the false alarmed normal flows and reduce the impact on them. But unlimited expansion of the SOM topology will not continue to significantly reduce the FAR and FAI. When TER = 3, the optimal SOM model is generated in the 7th iteration, and both FAR and FAI are in a small value range, which means the model achieves the expected effect. Secondly, according to the longitudinal comparison of the three algorithms during the entire iteration, the DGSOM algorithm is better than the k-algorithms. It not only maintains lower FAR and FAI during most iterations, but also shows better stability after the algorithm converges.

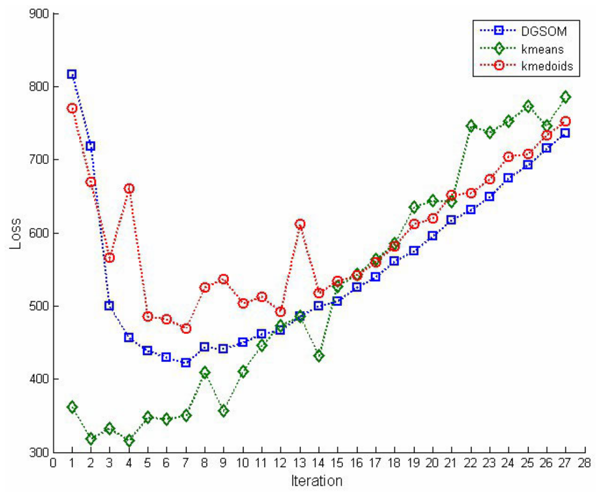


FIGURE 14. Loss function with $\lambda_0 = 0.008$ during iteration.

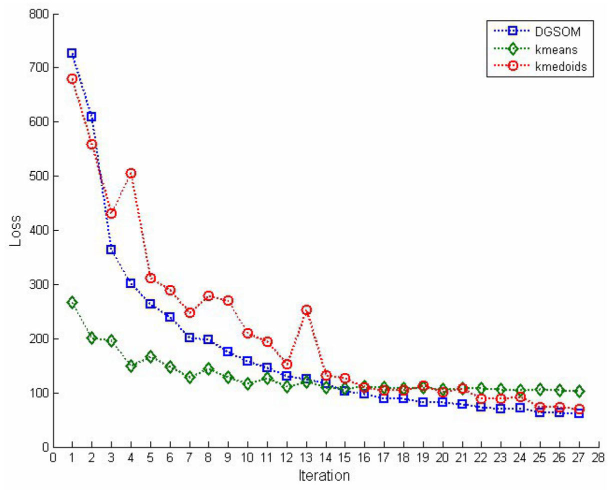


FIGURE 15. Loss function with $\lambda_0 = 0$ during iteration.

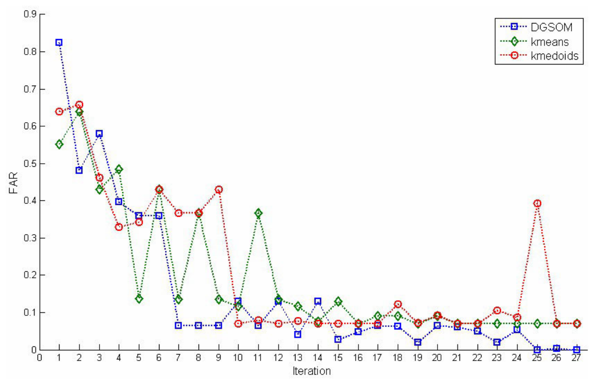


FIGURE 16. FAR during iteration.

Fig. 18 shows the change of DR during the entire iteration. Although the DR of DGSOM is not the best, it still maintains a stable and high level. The DR corresponding to the optimal model is the best performance 95.41%, which means

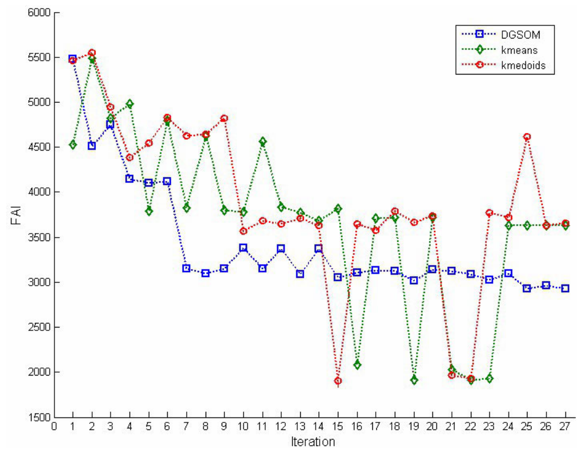


FIGURE 17. FAI during iteration.

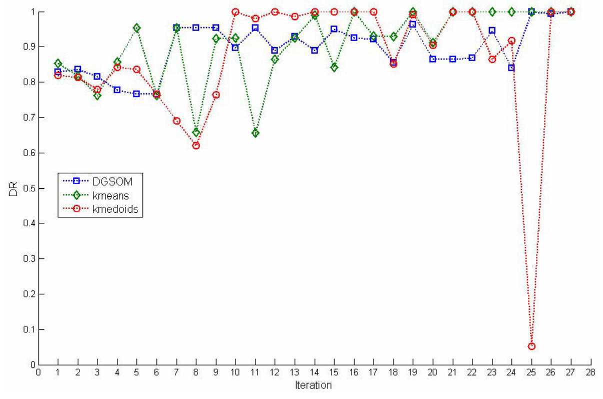


FIGURE 18. DR during iteration.

the response strategy will perform filtering action on more than 95% of real attack flows.

Fig. 19 and Fig. 20 show the time cost of each iteration and cumulative iteration of the DGSOM algorithm during the entire iteration process. In the comparison of each iteration, the time cost in every iteration of k-means and k-medoids is computed based on the clustering time of executing the algorithms for once with the number of clusters being equal to the number of neurons. But in the comparison of cumulative iteration, it defaults to compute the total time cost of multiple clustering processes starting from 4 clusters to the corresponding number of neurons one by one. According to the longitudinal comparison in the two figures, it can be seen that, with the expansion of SOM topology, the time consumed by each iteration of DGSOM is much higher than the k-algorithms, but the cumulative time is higher than k-means and lower than k-medoids. This is because when comparing the cumulative time cost, it defaults to find the optimal number of cluster center of k-means and k-medoids through ergodic search. Assuming that the upper limit of SOM topology is $n \times n$, the complexity of ergodic search is $O(n^2)$, but the greedy search strategy used in the DGSOM algorithm reduces the complexity to $O(n)$, which greatly

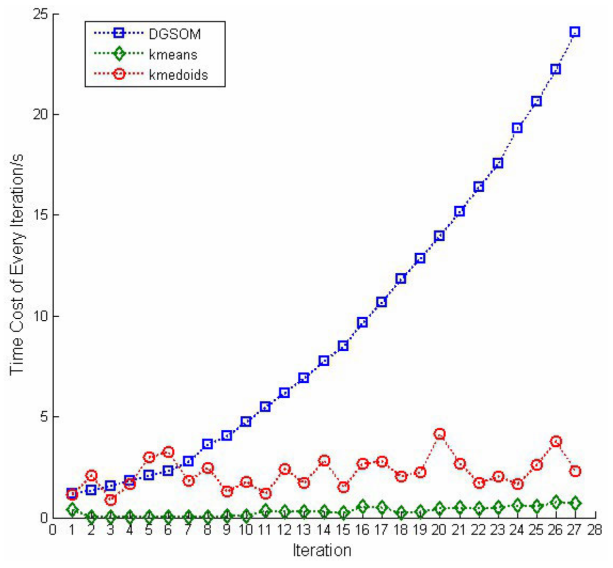


FIGURE 19. Time cost of each iteration during iteration.

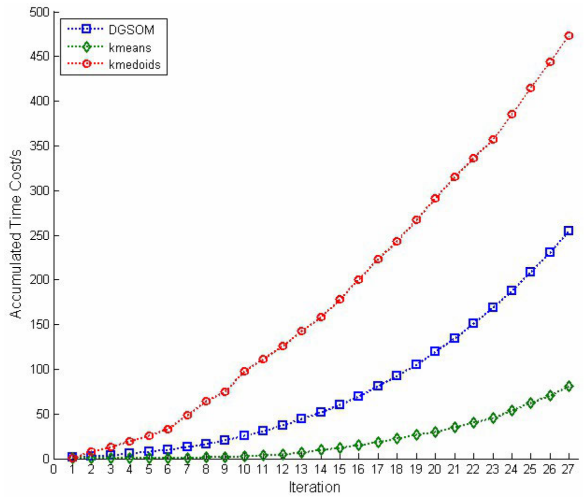


FIGURE 20. Time cost of cumulative iteration during iteration.

reduce the time of repeatedly training the model. Meanwhile, choosing appropriate early stopping parameter can further reduce training time. The training process taking too long will delay the model generation and affect the response agility. This is the reason why taking the pre-generation process. Although there is no guarantee that the SOM topology generated by the pre-generation process is always optimal for the current traffic, the sub-optimal topology and timely getting the clustering result are the most important things and should be considered first. When TER = 3, according to Fig. 19, the time cost of micro detection for 5,000 flows, including 3,000 normal flows and 2,000 attack flows, is 2.77 seconds (7th iteration). The cumulative time of stopping at the 10th iteration, according to Fig. 20, is 25.53 seconds. In other words, it takes about 28 seconds to get the clustering result without the pre-generation, but only about 3 seconds with pre-generation. Therefore, the time cost

TABLE 5. Optimal topology and iteration in each experiment.

No. i	optimal Topo	optimal iteration	stop iteration	number of clusters
0.5	5x5	7	10	25
1	5x6	8	11	30
2	5x4	6	9	20
3	6x4	7	10	24
4	5x5	7	10	25
5	5x7	9	12	35
6	5x6	8	11	30
7	6x6	9	12	36
8	6x7	10	13	42
9	4x8	9	12	32
10	4x4	5	8	16
11	6x4	7	10	24
12	6x6	9	12	36
13	4x6	7	10	24
14	6x5	8	11	30
15	4x6	7	10	24
16	6x3	6	9	18
17	7x5	9	12	35
18	5x6	8	11	30
19	5x6	8	11	30
20	4x6	7	10	24

of cumulative iteration is the basis to consider whether to use the pre-generation and how to configure parameter T_2.

In order to validate the detection effect under different numbers of attack flows without losing generality, the results of all 21 groups of experiments are given as follows, including FAR, FAI, DR, and time cost. Table 5 lists the topology structure of the optimal SOM model and the corresponding iteration number in each experiment, and also indicates the number of clusters of the k-algorithms used for comparison. Taking the first line in this table as an example, in the experiment with 50 attack flows, i.e. $i = 0.5$, the optimal SOM topology generated by DGSOM is 5×5, the algorithm stops at the 10th iteration, the optimal model is generated in 7th iteration, and the number of clusters used for comparison is 25. According to the difference between "optimal iteration" and "stop iteration", all experiments meet the early stopping criterion TER = 3. Actually, the results of configuring TER bigger than ITE in all experiments show that all the minimum loss functions obtained by the early stopping strategy are the global minimum.

Fig. 21 and Fig. 22 show the FAR and FAI respectively when the number of attack flows changes. According to the longitudinal comparison in both two figures, there are only 8 times that the FAR and FAI of DGSOM are not the lowest, which indicates DGSOM performs better than the other two algorithms overall. According to the horizontal comparison of FAR in Fig. 21, the DGSOM can always maintain the FAR at the lowest level when the number of attack flows is large, which is more stable than the other two algorithms. But when the number of attack flows is small, the FAR is maintained at about 40%, which means about 40% of false alarmed normal flows are filtered, while the remaining 60% are rate-limited. Fig. 23 shows the DR in each experiment. Although the DR of DGSOM is not always the best, its worst performance still

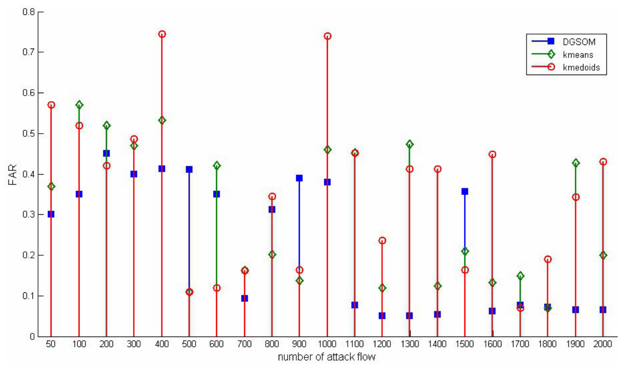


FIGURE 21. FAR of each experiment.

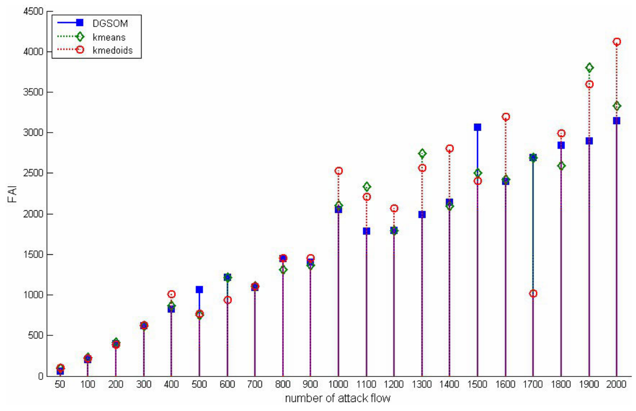


FIGURE 22. FAI of each experiment.

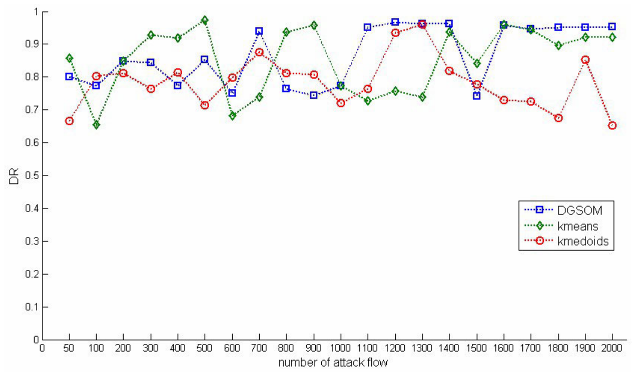


FIGURE 23. DR of each experiment.

reaches 74.15%, which means there are only nearly 26% of real attack flows being rate-limited, while the other real attack flows are filtered.

Fig. 24 shows the distribution of the response level division. It can be seen that the false alarmed normal flow with level 1 occupies the main component in every distribution, which does not achieve the desired ideal effect. According to the strategy of dividing response levels, the reason is that the difference between real attack flows and false alarmed normal flows is small, which causes the normal flow that is similar to the real attack flow having a lower response level. In this

TABLE 6. Details of the simulation attack scenario.

composition of replay traffic		IP of mapping node	access switch	response level
number of normal flows	number of attack flows			
10	0	10.0.0.1-10.0.0.10	s1	0
10	0	10.0.0.101-10.0.0.110	s2	
10	0	10.0.2.1-10.0.2.10	s3	
2	3	10.0.3.1-10.0.3.5	s4	
1	0	10.0.3.6	s4	1
3	1	10.0.3.7-10.0.3.10	s4	2
1	4	10.0.4.1-10.0.4.5	s5	
3	1	10.0.4.20-10.0.4.23	s5	3
0	1	10.0.4.24	s5	4

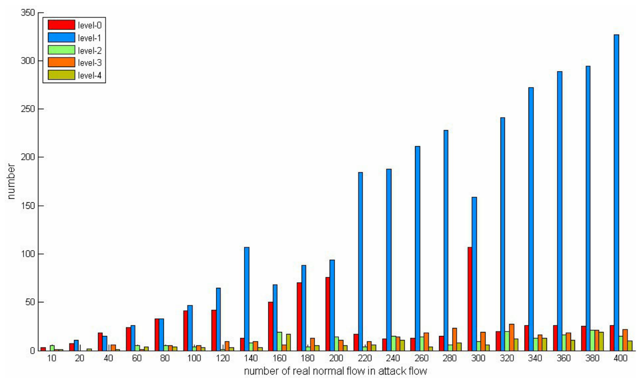


FIGURE 24. Distribution of response level division in each experiment.

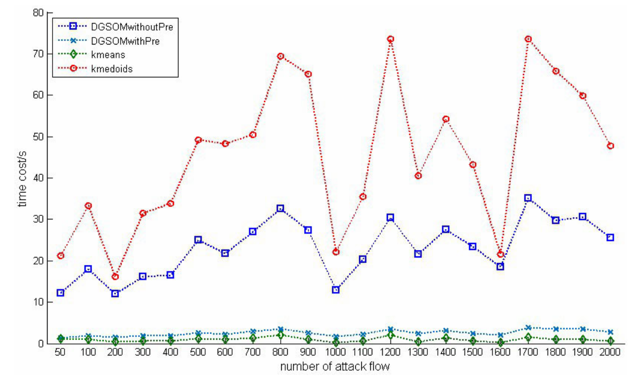


FIGURE 25. Cumulative time cost in each experiment.

case, reducing the response level K could effectively reduce FAI, but this will responsively relax the restriction on the real attack flow that is "false negative". There is a trade-off when the distinction between normal and attack flows is small.

Fig. 25 shows the cumulative time consumed by the DGSOM algorithm during training with and without the pre-generation process. According to the figure, the time cost of generating optimal SOM model through DGSOM is reduced from tens of seconds to less than a few seconds after using the pre-generation process.

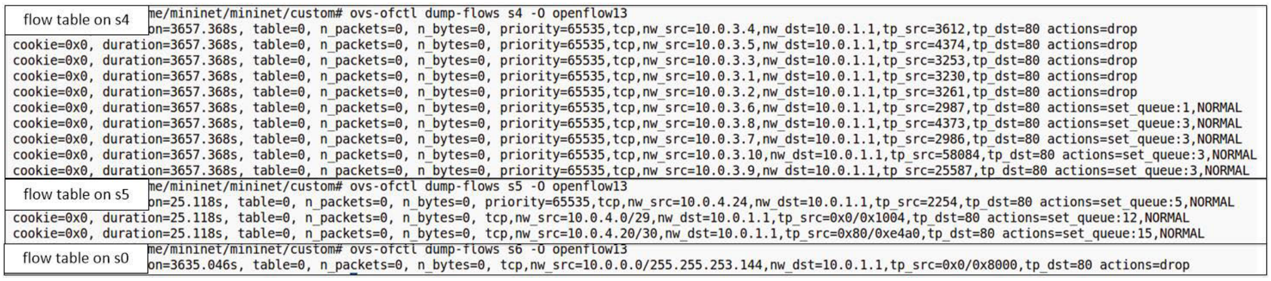
In summary, the proposed DGSOM algorithm can dynamically generate an optimal SOM model with a good clustering effect and low model complexity under the guidance of the loss function in an acceptable time. The experimental results validate that the DGSOM has a low FAR and a high DR in distinguishing the normal flow from the attack flow. It indicates that the response strategy will filter most real attack flows and apply loose rate-limiting to the most false alarmed normal flows. The micro detection has achieved the desired effect. Although the FAI is not ideal enough, the rate-limiting strategy at least ensures the accessibility of normal users.

C. VALIDATION OF RESPONSE STRATEGY

The experiment of validating the response strategy is based on the detection result corresponding to the experiment with 50 attack flows in the previous subsection. The simple network shown in Fig. 10 is used as the simulation network. The source IP addresses of all flows are first mapped to the IP address of the simulation nodes. And then, the specific response strategy is implemented. Finally, the effect of the response strategy is evaluated by observing the flow table information to see whether rules are generated, merged, and issued correctly. According to the detection result, there are 35, 1, 9, 4, 1 attack flows in each level from 0 to 4, and the number of false alarmed normal flows in each level is 3, 0, 5, 1, 1 respectively. The details of simulating the attack scenario are shown in Table 6. As the DDoS attack is aimed at "192.168.5.122:80", the destination IP addresses of all attack flows are mapped to 10.0.1.1 in the simulation experiment, while the source and destination port remaining unchanged. In addition, all flows are replayed by tcpreplay in zero-delay loop mode, and the replay rate is fixed at 1 Mbps.

The final flow rules generated by the response strategy module are shown in Table 7. According to the table, there are only 14 rules after the initial 50 rules being merged. These rules are issued to switch s0 (corresponding to s6 in the mininet simulation network), s4, and s5 respectively. As the destination IP address and port are fixed at "10.0.1.1:80", the related fields are ignored in the table.

Fig. 26 shows the flow table information on related switches viewed through OVS commands. It can be seen that all rules are correctly sent to the designated switch. The flow entry that can exactly match a flow is set with the highest priority 65535. In the experiment, implementing the queue rate limit strategy is based on the function provided by OVS. Multiple queues with fixed size are preset according to the replay rate. When generating the queue action of a flow



```
me/mininet/mininet/custom# ovs-ofctl dump-flows s4 -O openflow13
on=3657.368s, table=0, n_packets=0, n_bytes=0, priority=65535,tcp,nw_src=10.0.3.4,nw_dst=10.0.1.1,tp_src=3612,tp_dst=80 actions=drop
 cookie=0x0, duration=3657.368s, table=0, n_packets=0, n_bytes=0, priority=65535,tcp,nw_src=10.0.3.5,nw_dst=10.0.1.1,tp_src=4374,tp_dst=80 actions=drop
 cookie=0x0, duration=3657.368s, table=0, n_packets=0, n_bytes=0, priority=65535,tcp,nw_src=10.0.3.3,nw_dst=10.0.1.1,tp_src=3253,tp_dst=80 actions=drop
 cookie=0x0, duration=3657.368s, table=0, n_packets=0, n_bytes=0, priority=65535,tcp,nw_src=10.0.3.1,nw_dst=10.0.1.1,tp_src=3230,tp_dst=80 actions=drop
 cookie=0x0, duration=3657.368s, table=0, n_packets=0, n_bytes=0, priority=65535,tcp,nw_src=10.0.3.2,nw_dst=10.0.1.1,tp_src=3261,tp_dst=80 actions=drop
 cookie=0x0, duration=3657.368s, table=0, n_packets=0, n_bytes=0, priority=65535,tcp,nw_src=10.0.3.6,nw_dst=10.0.1.1,tp_src=2987,tp_dst=80 actions=set_queue:1,NORMAL
 cookie=0x0, duration=3657.368s, table=0, n_packets=0, n_bytes=0, priority=65535,tcp,nw_src=10.0.3.8,nw_dst=10.0.1.1,tp_src=4373,tp_dst=80 actions=set_queue:3,NORMAL
 cookie=0x0, duration=3657.368s, table=0, n_packets=0, n_bytes=0, priority=65535,tcp,nw_src=10.0.3.7,nw_dst=10.0.1.1,tp_src=2986,tp_dst=80 actions=set_queue:3,NORMAL
 cookie=0x0, duration=3657.368s, table=0, n_packets=0, n_bytes=0, priority=65535,tcp,nw_src=10.0.3.10,nw_dst=10.0.1.1,tp_src=58084,tp_dst=80 actions=set_queue:3,NORMAL
 cookie=0x0, duration=3657.368s, table=0, n_packets=0, n_bytes=0, priority=65535,tcp,nw_src=10.0.3.9,nw_dst=10.0.1.1,tp_src=25587,tp_dst=80 actions=set_queue:3,NORMAL
me/mininet/mininet/custom# ovs-ofctl dump-flows s5 -O openflow13
on=25.118s, table=0, n_packets=0, n_bytes=0, priority=65535,tcp,nw_src=10.0.4.24,nw_dst=10.0.1.1,tp_src=2254,tp_dst=80 actions=set_queue:5,NORMAL
 cookie=0x0, duration=25.118s, table=0, n_packets=0, n_bytes=0, tcp,nw_src=10.0.4.0/29,nw_dst=10.0.1.1,tp_src=0x0/0x1004,tp_dst=80 actions=set_queue:12,NORMAL
 cookie=0x0, duration=25.118s, table=0, n_packets=0, n_bytes=0, tcp,nw_src=10.0.4.20/30,nw_dst=10.0.1.1,tp_src=0x80/0xe4a0,tp_dst=80 actions=set_queue:15,NORMAL
me/mininet/mininet/custom# ovs-ofctl dump-flows s6 -O openflow13
on=3635.046s, table=0, n_packets=0, n_bytes=0, tcp,nw_src=10.0.0.0/255.255.253.144,nw_dst=10.0.1.1,tp_src=0x0/0x8000,tp_dst=80 actions=drop
```

FIGURE 26. Screenshot of switch flow table.

TABLE 7. Flow rules provided by response strategy.

level	flow rules					location
	match field		mask field		action	
	source IP	source port	IP mask	port mask		
0	10.0.0.0	0	255.255.253.144	0x8000	drop	s0
	10.0.3.4	3612	255.255.255.255	0xFFFF		s4
	10.0.3.5	4374	255.255.255.255	0xFFFF		
	10.0.3.3	3253	255.255.255.255	0xFFFF		
	10.0.3.1	3230	255.255.255.255	0xFFFF		
	10.0.3.2	3261	255.255.255.255	0xFFFF		
1	10.0.3.6	2987	255.255.255.255	0xFFFF	set_queue: 1, normal	s4
2	10.0.4.0	0	255.255.255.248	0x1004	set_queue: 12, normal	s5
	10.0.3.8	4373	255.255.255.255	0xFFFF	set_queue: 3, normal	s4
	10.0.3.7	2986	255.255.255.255	0xFFFF	set_queue: 3, normal	
	10.0.3.10	58048	255.255.255.255	0xFFFF	set_queue: 3, normal	
	10.0.3.9	25587	255.255.255.255	0xFFFF	set_queue: 3, normal	
3	10.0.4.20	128	255.255.255.252	0xE4A0	set_queue: 15, normal	s5
4	10.0.4.24	2254	255.255.255.255	0xFFFF	set_queue: 5, normal	s5

entry, the queue size is computed according to the response level and the total attack intensity of the corresponding attack flows. The queue with the closest size is selected and configured, which is the reason why some actions in Table 7 are the same. In addition to using the QoS provided by OVS, the OpenFlow-1.3 also provides a newly added Meter function to implement a simple OpenFlow-based QoS strategy. The rate-limiting strategy can be implemented by configuring the Rate filed in the Meter Bands. Besides, stopping replaying the traffic to simulate the end of the attack, or stopping replaying part of the traffic to simulate the feedback of flash crowds, the response strategy module will abandon the issued flow entries according to the feedback mechanism. As this process only involves the execution of commands, the experimental result will not be repeated here.

V. CONCLUSION

In this paper, we make full use of the advantages of SDN, including centralized control, programmability, global view, flexible response, and easy deployment, to propose a lightweight source-based defense method based on sFlow and SOM.

Compared with the traditional edge-based defense model, which lacks effective inner protection, the proposed method in this paper establishes a detection that can cover the entire network to perceive the DDoS attack. Benefiting from the characteristics of SDN, the proposed method is much easier to develop and deploy compared with traditional methods based on dedicated devices and closed systems. As it has very low CAPEX and OPEX, the network administrator has a higher willingness to deploy a source-based defense mechanism. Meanwhile, this work combines macro and micro detection to take into account detecting the occurrence of DDoS attacks as well as recognizing attack and normal flows. On one hand, a metric called DCD is proposed to characterize the intrinsic features of DDoS attacks. As collaboration and distribution is prerequisite for a DDoS attack, the detection based on DCD is much harder to be bypassed than the detection based on other features, such as entropy. On the other hand, an improved clustering algorithm called DGSOM is proposed based on the standard SOM. It can dynamically generate the optimal model under the guidance of loss function. It is effective to distinguish the attack flow from the normal flow. This unsupervised method does not require careful preparation of perfect and labeled data. The improved clustering algorithm is also overall better than the k-means and k-medoids. In addition, a flexible response strategy is designed to handle attack flows.

The defect of this work mainly includes two aspects. Firstly, it is necessary to conduct a more rigorous and comprehensive verification experiment in the real network to test its detection performance. Secondly, in the case that normal flows are very similar to attack flows, the response strategy will impose strict restrictions on more false alarmed normal flows, which will affect the experience of normal users. How to distinguish between attack and normal with high similarity is a difficulty in the field of DDoS attack detection.

As people pay more and more attention to the cyber security, addressing the cyber security issues at the source will become increasingly important. Source-based defense mechanisms should have the same importance as the mechanisms based on other deployment locations. In the future work, it is very promising to study cross-domain collaborative defense mechanisms based on SDN.

CONFLICTS OF INTEREST

The authors declare that there are no conflicts of interest regarding the publication of this paper.

REFERENCES

[1] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, "Botnet in DDoS attacks: Trends and challenges," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2242–2270, 4th Quart., 2015, doi: 10.1109/COMST.2015.2457491.

[2] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 2046–2069, 4th Quart., 2013, doi: 10.1109/SURV.2013.031413.00127.

[3] CNCERT/CC. *2019 China Internet Cybersecurity Report*. National Computer Network Emergency Response Technical Team/Coordination Center of China. Accessed: May 20, 2021. [Online]. Available: https://www.cert.org.cn/publish/main/index.html

[4] Q. Yan and F. Yu, "Distributed denial of service attacks in software-defined networking with cloud computing," *IEEE Commun. Mag.*, vol. 53, no. 4, pp. 52–59, Apr. 2015, doi: 10.1109/MCOM.2015.7081075.

[5] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 602–622, 1st Quart., 2016, doi: 10.1109/COMST.2015.2487361.

[6] M. A. Khan and K. Salah, "IoT security: Review, blockchain solutions, and open challenges," *Future Gener. Comput. Syst.*, vol. 82, pp. 395–411, May 2017, doi: 10.1016/j.future.2017.11.022.

[7] Z. Lu, W. Wang, and C. Wang, "On the evolution and impact of mobile botnets in wireless networks," *IEEE Trans. Mobile Comput.*, vol. 15, no. 9, pp. 2304–2316, Sep. 2016, doi: 10.1109/TMC.2015.2492545.

[8] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015, doi: 10.1109/JPROC.2014.2371999.

[9] N. Z. Bawany, J. A. Shamsi, and K. Salah, "DDoS attack detection and mitigation using SDN: Methods, practices, and solutions," *Arabian J. Sci. Eng.*, vol. 42, no. 2, pp. 425–441, 2017, doi: 10.1007/s13369-017-2414-5.

[10] J. Singh and S. Behal, "Detection and mitigation of DDoS attacks in SDN: A comprehensive review, research challenges and future directions," *Comput. Sci. Rev.*, vol. 37, Aug. 2020, Art. no. 100279, doi: 10.1016/j.cosrev.2020.100279.

[11] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, Jul. 2013, doi: 10.1109/MCOM.2013.6553676.

[12] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using OpenFlow: A survey," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 493–512, 1st Quart., 2014, doi: 10.1109/SURV.2013.081313.00105.

[13] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 27–51, 1st Quart., 2015, doi: 10.1109/COMST.2014.2330903.

[14] P. Phaal, S. Panchen, and N. McKee, *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks*, document RFC 3176, 2001.

[15] P. Ferguson, "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing," Netw. Work. Group, IETF, Fremont, CA, USA, Tech. Rep., 2000, pp. 4–6.

[16] S. Kent and K. Seo, "Security architecture for IP," Netw. Work. Group, IETF, Fremont, CA, USA, 2005, pp. 11–50.

[17] S. Kent, "IP authentication header," Netw. Work. Group, IETF, Fremont, CA, USA, 2005, pp. 9–20.

[18] J. Mirkovic and P. Reiher, "D-WARD: A source-end defense against flooding denial-of-service attacks," *IEEE Trans. Dependable Secure Comput.*, vol. 2, no. 3, pp. 216–232, Jul. 2005, doi: 10.1109/TDSC.2005.35.

[19] S. Abdelsayed, D. Glimsholt, C. Leckie, S. Ryan, and S. Shami, "An efficient filter for denial-of-service bandwidth attacks," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Dec. 2003, pp. 1353–1357, doi: 10.1109/GLOCOM.2003.1258459.

[20] CISCO. (2018). *The Reverse Firewall: Defeating DDoS Attacks*. Accessed: Nov. 14, 2020. [Online]. Available: https://www.cisco.com/c/en/us/about/security-center/guide-ddos-defense.html

[21] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *Proc. IEEE Local Comput. Netw. Conf.*, Oct. 2010, pp. 408–415, doi: 10.1109/LCN.2010.5735752.

[22] Y. Xu and Y. Liu, "DDoS attack detection under SDN context," in *Proc. IEEE INFOCOM - 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9, doi: 10.1109/INFOCOM.2016.7524500.

[23] T. M. Nam, P. H. Phong, T. D. Khoa, T. T. Huong, P. N. Nam, N. H. Thanh, L. X. Thang, P. A. Tuan, L. Q. Dung, and V. D. Loi, "Self-organizing map-based approaches in DDoS flooding detection using SDN," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2018, pp. 249–254, doi: 10.1109/ICOIN.2018.8343119.

[24] K. S. Sahoo, B. K. Tripathy, K. Naik, S. Ramasubbareddy, B. Balusamy, M. Khari, and D. Burgos, "An evolutionary SVM model for DDOS attack detection in software defined networks," *IEEE Access*, vol. 8, pp. 132502–132513, 2020, doi: 10.1109/ACCESS.2020.3009733.

[25] S. Haider, A. Akhunzada, I. Mustafa, T. B. Patel, A. Fernandez, K. Kwang R. Choo, and J. Iqbal, "A deep cnn ensemble framework for efficient DDOs attack detection in software defined networks," *IEEE Access*, vol. 8, pp. 53972–53983, 2020, doi: 10.1109/ACCESS.2020.2976908.

[26] J. A. Pérez-Díaz, I. A. Valdovinos, K.-K. R. Choo, and D. Zhu, "A flexible SDN-based architecture for identifying and mitigating low-rate DDoS attacks using machine learning," *IEEE Access*, vol. 8, pp. 155859–155872, 2020, doi: 10.1109/ACCESS.2020.3019330.

[27] R. Wang, Z. Jia, and L. Ju, "An entropy-based distributed DDoS detection mechanism in software-defined networking," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2015, pp. 310–317, doi: 10.1109/Trustcom.2015.389.

[28] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Lecture Notes in Computer Science), vol. 6961. Berlin, Germany: Springer, 2011, pp. 161–180, doi: 10.1007/978-3-642-23644-0_9.

[29] Q. Niyaz, W. Sun, and A. Y. Javaid, "A deep learning based DDoS detection system in software-defined networking (SDN)," *ICST Trans. Secur. Saf.*, vol. 4, no. 12, Dec. 2017, Art. no. 153515, doi: 10.4108/eai.28-12-2017.153515.

[30] G. Sun, W. Jiang, Y. Gu, D. Ren, and H. Li, "DDoS attacks and flash event detection based on flow characteristics in SDN," in *Proc. 15th IEEE Int. Conf. Adv. Video Signal Based Surveill. (AVSS)*, Nov. 2018, pp. 1–6, doi: 10.1109/AVSS.2018.8639103.

[31] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Comput. Netw.*, vol. 62, pp. 122–136, Apr. 2014, doi: 10.1016/j.bjp.2013.10.014.

[32] C. Buragohain and N. Medhi, "FlowTrApp: An SDN based architecture for DDoS attack detection and mitigation in data centers," in *Proc. 3rd Int. Conf. Signal Process. Integr. Netw. (SPIN)*, Feb. 2016, pp. 519–524, doi: 10.1109/SPIN.2016.7566750.

[33] C. Wang, J. Zheng, and X. Li, "Research on DDoS attacks detection based on RDF-SVM," in *Proc. 10th Int. Conf. Intell. Comput. Technol. Autom. (ICICTA)*, Oct. 2017, pp. 161–165, doi: 10.1109/ICICTA.2017.43.

[34] Y. Chen, K. Hwang, and W. S. Ku, "Collaborative detection of DDoS attacks over multiple network domains," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 12, pp. 1649–1662, Dec. 2007, doi: 10.1109/TPDS.2007.1111.

[35] J. Boite, P.-A. Nardin, F. Rebecchi, M. Bouet, and V. Conan, "Statesec: Stateful monitoring for DDoS protection in software defined networks," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Jul. 2017, pp. 1–9, doi: 10.1109/NETSOFT.2017.8004113.

[36] K. Kalkan, L. Altay, G. Gür, and F. Alagöz, "JESS: Joint entropy-based DDoS defense scheme in SDN," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2358–2372, Oct. 2018, doi: 10.1109/JSAC.2018.2869997.

[37] G.-C. Hong, C.-N. Lee, and M.-F. Lee, "Dynamic threshold for DDoS mitigation in SDN environment," in *Proc. Asia–Pacific Signal Inf. Process. Assoc. Annu. Summit Conf. (APSIPA ASC)*, Nov. 2019, pp. 1–7, doi: 10.1109/APSIPAASC47483.2019.9023229.

[38] A. Ahalawat, S. S. Dash, A. Panda, and K. S. Babu, "Entropy based DDoS detection and mitigation in OpenFlow enabled SDN," in *Proc. Int. Conf. Vis. Towards Emerg. Trends Commun. Netw. (ViTECoN)*, Mar. 2019, pp. 1–5, doi: 10.1109/ViTECoN.2019.8899721.

[39] Y. Lu and M. Wang, "An easy defense mechanism against botnet-based DDoS flooding attack originated in SDN environment using sFlow," in *Proc. 11th Int. Conf. Future Internet Technol.*, 2016, pp. 14–20, doi: 10.1145/2935663.2935674.

[40] A. Rauber, D. Merkl, and M. Dittenbach, "The growing hierarchical self-organizing map: Exploratory analysis of high-dimensional data," *IEEE Trans. Neural Netw.*, vol. 13, no. 6, pp. 1331–1341, Nov. 2002, doi: 10.1109/TNN.2002.804221.

[41] E. de la Hoz, E. de la Hoz, A. Ortiz, J. Ortega, and A. Martínez-Álvarez, "Feature selection by multi-objective optimisation: Application to network anomaly detection by hierarchical self-organising maps," *Knowl.-Based Syst.*, vol. 71, pp. 322–338, Nov. 2014, doi: 10.1016/j.knosys.2014.08.013.

[42] Y. Cao, H. B. He, and H. Man, "SOMKE: Kernel density estimation over data streams by sequences of self-organizing maps," *IEEE Trans. Neural Netw., Learn. Syst.*, vol. 23, no. 8, pp. 1254–1268, Aug. 2012, doi: 10.1109/TNNLS.2012.2201167.

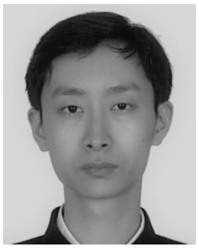
[43] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Secur.*, vol. 31, no. 3, pp. 357–374, May 2012, doi: 10.1016/j.cose.2011.12.012.

[44] M. Wang, Y. Lu, and J. Qin, "A dynamic MLP-based DDoS attack detection method using feature selection and feedback," *Comput. Secur.*, vol. 88, Jan. 2020, Art. no. 101645, doi: 10.1016/j.cose.2019.101645.

MENG WANG is currently pursuing the Ph.D. degree with the School of Electronic and Information Engineering, South China University of Technology, Guangzhou, China. His current research interests include network security and software-defined networking.

YIQIN LU was born in Zhaoqin, Guangdong, China. He received the Ph.D. degree in electronic circuit and systems form the South China University of Technology (SCUT), Guangzhou, China, in 1996.

He was appointed as a Professor with SCUT, in 2006. He was also the Doctoral Supervisor of the College of Electronic and Information, SCUT. He is currently the Dean of the Office of the Leading Group for Cyberspace Affairs, SCUT; the Director of the Network Center of Southern China Region, China Education and Research Network (CERNET); and the President of Computer Information Network Safety Association of Guangdong Province. His research interests include communication networks, network security, error correcting code, and the Internet of Things.



JIANCHENG QIN received the bachelor's degree in computer engineering from the School of Computer Science and Technology, Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 1999, and the master's degree from the School of Software Engineering, BUPT, in 2008, and the Ph.D. degree from School of Computer, BUPT, in 2011. His research interests include information security and mobile computing.