IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 8 March 2024, accepted 27 March 2024, date of publication 2 April 2024, date of current version 9 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3383877

RESEARCH ARTICLE

Research on Detection and Mitigation Methods of Adaptive Flow Table Overflow Attacks in Software-Defined Networks

YING ZENG, YONG WANG, AND YUMING LIU
School of Computer and Information Security, Guilin University of Electronic Technology, Guilin, Guangxi 541004, China

Corresponding author: Yuming Liu (suiyanlovetian@foxmail.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61861013, in part by the Science and Technology Major Project of Guangxi under Grant AA18118031, and in part by the Innovation Project of Guilin University of Electronic Technology (GUET) Graduate Education under Grant 2023YCXS056.

ABSTRACT In Software-Defined Networks (SDN), the ternary content addressable memory (TCAM) capacity in switches is limited, making them vulnerable to low-rate flow table overflow attacks. Most existing research in this field has not focused on the influence of flow entry eviction mechanisms on the effectiveness of such attacks. This paper proposes an adaptive low-rate flow table overflow attack (ALFO), which can adopt corresponding attack modes under different flow entry eviction mechanisms, significantly degrading network service quality. Due to the different features of ALFO under different attack modes, the existing attack detection methods are ineffective in this attack. Therefore, this paper proposes a detection and mitigation framework, which is called adaptive low-rate flow table overflow attack guard framework (ALFO-Guard). It extracts flow features from flow entry information in the switch and aggregates them into a current-time graph model. Then, combining graph neural networks, it performs graph anomaly detection and flow entry classification to identify attack flow entries. Finally, the attack can be eliminated by deleting the identified attack flow entries and blocking the attack flows. The effectiveness of ALFO and ALFO-Guard is validated through extensive experiments, and the experimental results demonstrate that ALFO-Guard can effectively defend against ALFO.

INDEX TERMS SDN, flow table overflow, low-rate attacks, graph neural network.

I. INTRODUCTION

Software-Defined Networks(SDN) [1], as a novel network architecture, exhibits characteristics such as centralized control, separation of forwarding and control planes, and network programmability [2]. However, this new architectural paradigm also introduces novel network threats [3], [4], including security concerns in the data plane [5], [6]. Currently, switches that support the OpenFlow protocol [7] employ TCAM [8] to store flow entries. Due to cost and capacity limitations [9], [10], [11], most commercial switches can only accommodate a few thousand to tens of thousands of flow entries [12]. Consequently, flow table overflow attacks have emerged as a significant threat in the realm of SDN security, particularly low-rate flow table overflow attacks that possess higher levels of stealthiness [13].

Most existing research in the field has largely overlooked the influence of flow entry eviction mechanisms on the effectiveness of flow table overflow attacks. In this paper, we present a novel attack called ALFO (Adaptive Low-rate Flow Table Overflow Attack). ALFO adjusts its attack modes under different flow entry eviction mechanisms. When overflow occurs and triggers the eviction mechanism, ALFO manipulates the controller to favor evict legitimate flow entries, resulting in a more significant attack effect than the existing low-rate flow table overflow attack methods. Literature [14] indicates that different attack flow features can affect the detection effectiveness of machine learning-based flow feature detection methods. Due to the different attack modes under different flow entry eviction mechanisms,

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

ALFO exhibits more diverse flow features than existing flow table overflow attacks. Consequently, the existing attack detection methods are inadequate for detecting ALFO. In conclusion, effectively defending against ALFO has become a new challenge in the SDN network environment.

In recent years, the application of graph neural networks (GNNs) [15] in the field of network security has shown promising results, such as network intrusion and anomaly detection [16]. Given the GNN's ability to capture structural information hidden in network topologies, we apply it in detecting flow table overflow attacks for the first time. Concretely, we introduce a GNN-based attack detection and mitigation framework called ALFO-Guard. It collects real-time flow entry information and uses a graph model represents the interrelationships among them. Subsequently, combining graph neural networks, it performs graph anomaly detection and flow entry classification to identify attack flow entries. Additionally, the framework incorporates the attack mitigation mechanism which primarily consists of two operations: deleting attack flow entries and adding them to a blacklist, to minimize flow table overflow issues and mitigate their impact on network performance.

The main contributions of this paper are as follows:

1) We propose a novel adaptive low-rate flow table overflow attack called ALFO. Compared to existing low-rate flow table overflow attacks, we consider the influence of flow entry eviction mechanisms. ALFO adopts corresponding attack modes under different eviction mechanisms, significantly degrading the quality of service of normal applications.

2) Existing methods show limited defense performance for ALFO. To defend against this attack effectively, we propose a detection and mitigation framework called ALFO-Guard. It extracts flow features to generate current-time graph model. Then, combining graph neural networks, it performs graph anomaly detection and flow entry classification to identify attack flow entries. Finally, it deletes the identified attack flow entries and activates a blacklist mechanism to defend against attack.

3) We conduct extensive experiments to validate the feasibility and effectiveness of ALFO and ALFO-Guard. The experimental results demonstrate that the ALFO-Guard can effectively defend against ALFO.

The organizational structure of this paper is as follows. In Section II, related work is presented. Section III describes the attack model and explaining the principles of ALFO and the attack mode under different eviction mechanisms. In Section IV, the workflow of the detection and mitigation framework, ALFO-Guard, is detailed. Section V introduces the relevant experiments conducted to evaluate the effectiveness of the attacks and ALFO-Guard. Finally, in Section VI, a summary of this paper is provided.

II. RELATED WORKS

This section categorizes LDoS attacks, focusing on flow table overflow attacks and LDoS attacks targeting hosts in SDN. Firstly, we discuss flow table overflow attacks against the data plane, analyzing their attack principles and relevant detection and defense methods. Secondly, we explore LDoS attacks against hosts in SDN and summarize existing attack detection and defense methods. Finally, we summarize all the references in a table format, as shown in Table 1.

A. FLOW TABLE OVERFLOW ATTACK

Currently, research efforts on flow table overflow attacks in SDN can be classified into two categories based on the overflow rate: low-rate and high-rate flow table overflow attacks. On the one hand, attackers can leverage the operational mechanisms of the OpenFlow protocol to gather information about the size of a switch's flow table and the timeout duration of its flow entries [17]. By delivering well-crafted packets, they force the targeted switch to install flow entries for the attack flows, thereby persistently occupying the switch's TCAM memory space. Attackers aims to achieve flow table overflow with minimal attack flows, resulting in a degradation of network service quality. On the other hand, with the gathered information, attackers can send numerous well-crafted packets to disrupt the network by overloading the control channel or overwhelming the processing capabilities of the controller. This effectively hampers the controller's ability to serve normal applications. Subsequently, this section provides a comprehensive overview of existing approaches for defending against flow table overflow attacks.

1) LOW-RATE FLOW TABLE OVERFLOW ATTACK

Pascoal et al. [18] proposed a low-rate flow table overflow attack called Slow-TCAM. This attack utilizes a botnet consisting of half the number of switches to send attack packets and occupy flow table resources. The sending interval of each packet is smaller than the inferred timeout duration of flow entries obtained through network configuration information. Cao et al. [19] introduced a two-stage low-rate overflow attack, LOFT. In the probing stage, attackers probe network configuration (timeout of flow entries, etc.) and calculate the lower bound of the attack rate. In the attack stage, attackers periodically send attack packets to consume flow entries, causing the flow table to remain in an overflow state. Building upon the work of Pascoal et al. [18], Zhijun et al. [20] proposed a linearly increasing low-rate flow table overflow attack. In the initiation stage, the attacker gradually increases the number of attack packets between adjacent attack cycles to enhance the stealthiness. However, these attacks do not consider the influence of flow entry eviction mechanisms on the effectiveness of the attacks. Additionally, they implicitly assume a continuous increase in flow entries during the attack initiation stage, using it as a triggering condition for detection mechanisms. However, such an assumption may not always hold true.

Currently, there are several research studies on defending against low-rate flow table overflow attacks. Zhijun et al. [20] proposed an attack detection and mitigation mechanism based on Factorization Machine. This mechanism consists of multi-feature attack detection and dynamic removal

TABLE 1. Related work summary.

Classification	Paper	Detection method	Mitigation methods
Low-rate flow table overflow attack	[18]	—	selective strategies
	[19]	—	—
Low-rate flow table overflow attack — statistical features	[21]	evaluation strategy base on flow feature, threshold	selecting the appropriate flow entries to delete, LRU
Low-rate flow table overflow attack — machine learning	[20]	Factorization Machine	dynamically deleting flow rules
	[22]	BayesNet	the controller issue the rules to block attack flow
	[23]	RandomForests	dynamically migrates flow rules, the blacklist of malicious flows
	[24]	XGBoost	remove rules, a blacklist of suspicious source IPs
	[25]	D3 (Discriminative Drift Detector)	a table segmentation module, score-based flow entry eviction module
High-rate flow table overflow attack — statistical features	[26]	traffic states of real addresses, evaluation scores, network resource usages	packet filter
	[27]	new frequency features	table-miss engineering, packet filter, flow rule management
High-rate flow table overflow attack — machine learning	[28]	PSO-BP neural network	install flow rule in switch
	[29]	supervised classifiers: KNN, SVM, and Naive Bayes semi-supervised classifiers: One-Class SVM, Isolation Forest, Basic Autoencoder, and Variational Autoencoder	identify the zombie hosts, install high priority blocking rules
LDoS attacks targeting hosts in SDN — statistical features	[30]	Renyi entropy, Information distance	install flow rule in switch
	[31]	the number of connections it engages that sent incomplete HTTP requests	instructing the controller to establish a new flow rule
	[32]	Mean Euclidean Distance	install mitigation rules in ingress switches
	[33]	expectation of packet size	—
LDoS attacks targeting hosts in SDN — machine learning	[34]	GBDT, Logistic Regression	locate attack sources and victims, drop packets
	[35]	WMS-K-Means	—
	[36]	LightGBM	locate attack, packet filter
Low-rate flow table overflow attack — deep learning	ALFO-Guard	GNN (Graph neural network)	delete attack flow entry, the ip_port blacklist base on threshold

of flow entries, to achieve fine-grained detection results. Xie et al. [21] presented an attack detection and mitigation mechanism called SAIA, which consists of four modules: data collection, overflow prediction, attack detection, and overflow mitigation. SAIA periodically samples the required data, performs attack detection based on thresholds, removes detected attack flow entries, and deletes a certain number of normal flow entries based on the Least Recently Used (LRU) algorithm to prevent flow table overflow. Xingshu et al. [22] proposed a detection framework for in SDN. This method extracts ten-dimensional flow features and employs a Bayesian network for attack detection. Cao et al. [23] introduced a mitigation system called LOFTGuard. LOFTGuard dynamically migrates flow entries between TCAM and software regions based on flow activity and rate. It detects attacks using long-flow features and flow table utilization features, and identifies attack flows based on Random Forests model. Once an attack flow is identified, the corresponding flow entry is removed and blocked. Tang et al. [24] proposed an attack detection and mitigation system called FTMaster. It monitors the number of flow entries in real time and activates the flow table detection module when it exceeds a detection threshold. Once an attack is detected, FTMaster classifies each flow entry based on XGBoost model, removes the detected attack flow entries, and initiates a blacklist mechanism. However, the aforementioned detection methods either rely on threshold-based approaches, which have limitations under different network environments or attack modes, or heavily depend on assumed attack flow features, making it easy for attackers to bypass detection by disguising the attack flow features. Liu et al. [25] presented a defense mechanism called POAGuard for defending against flow table overflow attacks. It consists of a flow table partitioning module for isolating attack flows, a score-based flow table entry eviction module, and an attack detection module based on concept drift. Although the attack modes under the LFU eviction mechanisms are described in [25], it does not consider other flow entry eviction mechanisms, making it unsuitable for detecting ALFO.

2) HIGH-RATE FLOW TABLE OVERFLOW ATTACK

Zhang et al. [26] designed an SDN mitigation framework called FloodShield. It filters attack packets directly based on the source IP of flows and monitors traffic status. Besides, it employs a probabilistic acceptance mechanism for packet-in messages based on evaluation scores and controller CPU utilization. Gao et al. [27] proposed a mitigation framework named FloodDefender, which establishes a traffic migration model based on queuing theory formulas and link utilization. When an attack is detected, it filters packet-in messages, removes unnecessary flow entries, and migrates network packets to adjacent switches based on the available secure channel capacity with those switches. Liu et al. [28] presented a attack detection method that combines information entropy and BP neural network. This method uses information entropy to locate anomalous switches and then extracts flow features from these switches. Besides, BP neural network model is employed to identify DDoS attack flows. Khamaiseh et al. [29] proposed a machine learning-based attack detection framework called vSwitchGuard. It extracts six features from flows to establish its detection model. Besides, it determines an ongoing attack and locates attack hosts based on packet-in messages. The aforementioned methods focus on detecting high-rate flow table overflow attacks. Since the flow features of high-rate flow table overflow attacks differ from those of low-rate overflow attacks, these methods are not suitable for detecting the stealthier low-rate flow table overflow attacks.

B. LDOS ATTACKS TARGETING HOSTS IN SDN

In SDN networks, low-rate DoS attacks can target various components within SDN, such as attacks against switches and attacks against hosts. The previous section primarily summarized the research on flow table overflow attacks. In this section, we will discuss existing methods for other types of LDoS attacks, which can be categorized into statistical feature-based detection and machine learning-based detection methods.

1) STATISTICAL FEATURE-BASED DETECTION METHODS

Statistical feature-based detection methods [30], [31], [32], [33] detect attacks based on the effect caused and reasonable thresholds. Anchal et al. [30] proposed an LDoS detection method called REPD, which utilizes information distance measurement to evaluate network traffic fluctuations under different probability distributions. It adopts packet dropping as a means to mitigate LDoS attacks. Hong et al. [31] proposed a SDN-based defense method called SHDA to mitigate low-rate HTTP DDoS. This method detects attacks by monitoring if the number of HTTP request connections exceeds a predefined threshold. Xie et al. [32] introduced a lightweight mitigation system called SoftGuard, which effectively defends against low-rate TCP attacks. It designs an adaptive fast Fourier transform algorithm to infer attack cycles and uses Euclidean distance to identify attack flows, followed by attack source localization based on flow paths. Zhou et al. [33] proposed a detection method for LDDoS based on packet size expectations. The researchers used the variance of packet sizes as a metric to measure the attack.

2) MACHINE LEARNING-BASED DETECTION METHODS

Machine learning-based detection methods [34], [35], [36] classify flows based on their features to accurately identify attack flows. Tang et al. [34] proposed a detection and mitigation framework called P&F. This framework extracts traffic features and categorizes them into two types: performance features and behavior features for LDoS attack detection. P&F utilizes time-frequency analysis to locate the source and target of LDoS attacks based on the flow characteristics. Finally, based on the detection and localization results, P&F deploys defense rules to filter LDoS attack traffic. Liu et al. [35] proposed a low-rate Denial of Service (DoS) attack detection method in SDN environments based on the WMS-Kmeans algorithm. This method uses the six-tuple features of flow table entries in switches as input for an improved Mean-Shift algorithm, which generates initial clustering centers. These centers are then used in the WMS-Kmeans algorithm to perform clustering and detect LDoS attacks. Tang et al. [36] proposed a detection and mitigation framework called ADMS for LDoS attacks targeting TCP congestion control mechanisms. It combines the traffic features of each switch port with a LightGBM classifier for flow features. However, these methods are not applicable to the attacks proposed in the current study due to the differences in attack targets or attack modes.

TABLE 2. Symbols and their meanings.

Symbol	Description
C_{max}	The flow table capacity of the switch, for instance, if the TCAM capacity of the switch is 1500, then $C_{max} = 1500$
F_M	The set of attack flows, e.g., $F_M = \{f_1, f_2, ..., f_m\}$ where f_i reprensts an attack flow
C_N	The number of normal flows,which is assumed to be constant
C_M	The number of attack flows , which change over time
v_N^i	Transmission rate of the i-th normal flow (pkts/s)
v_M^j	Transmission rate of the j-th attack flow (pkts/s)
$\tilde{v}_{max}$	Application-limited maximum flow rate (pkts/s)
idle_time	The default timeout value of flow entry, configured by the controller
T	Attack cycle
$T_{overflow}$	The time when overflow occurs

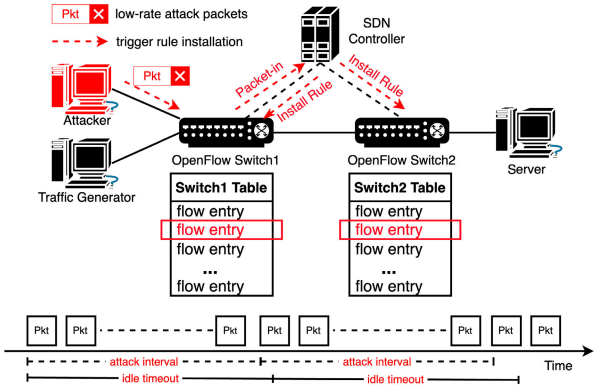


FIGURE 1. Low rate flow table overflow attack.

III. ATTACK MODEL

The ALFO is built upon the foundation of the low-rate flow table overflow attack. Therefore, in this section, before introducing the ALFO model, we first describe the principles of the low-rate flow table overflow attack. Next, we establish the model of ALFO, followed by an analysis of its attack mode. Since prior works [19], [37], [38] have already discussed how to probe the installation rules, timeout mechanisms, flow entry eviction mechanisms, and the capacity of the flow table, we will not delve into them here while assuming that they are readily available. Table 2 presents the symbols used in this paper along with their corresponding meanings.

A. LOW RATE FLOW TABLE OVERFLOW ATTACK

According to the workflow of the OpenFlow protocol, packets that do not match any flow entries will trigger SDN switches to generate Packet-in messages. Upon receiving such a message, the controller adds the corresponding flow rule to the switch based on the request. Inspired by low-rate DoS attacks, researchers have proposed low-rate DoS attacks targeting the flow table [18], [19], as shown in Fig.1. In this attack, attackers send well-crafted packets at a low rate, gradually consuming flow entries until the flow table overflows degrading the network's service performance.

Existing attack methods set the attack cycle based on idle timeout and achieve flow table overflow by sending attack packets at a low rate. However, they overlook the influence of flow entry eviction mechanisms on the attack's effectiveness.

In this paper, we investigate the effect of flow entry eviction mechanisms and propose the ALFO, which prompts the controller to preferentially evict the flow entries of normal applications when overflow occurs, leading to an amplified degradation of the quality of service of normal applications. Next, we will provide a detailed introduction to ALFO.

B. ALFO MODEL

The ALFO process consists of two phases: the overflow phase and the attack maintenance phase. In the overflow phase, it is assumed that the attacker's objective is to ensure the stealthiness H of the attack before overflowing the flow table. In the attack maintenance phase, it is assumed that the attacker's objective is to employ different modes based on various flow entry eviction mechanisms to sustain the effect generated by the attack. A detailed analysis is provided below:

In the overflow phase, the attacker aims to overflow the flow table covertly. The research conducted in [23] and [24] indicates a certain correlation between stealthiness and the growth trends of packet-in messages and flow entries. Specifically, when there is a significant increase in the number of packet-in messages or a more pronounced growth trend in flow entries, the stealthiness of the attack becomes lower. Therefore, we denote the growth trend of flow entries at time t as d_t, as shown in (1). Additionally, when the attack cycle exceeds, the switch will delete all attack flow table entries in each attack cycle, leading to a noticeable increase in packet_in messages. In such cases, the attack lacks stealthiness. Hence, we define the expression for the stealthiness of the attack, H, as shown in (2). In equation (2), the value of H ranges from 0 to 1, where a higher value indicates a more stealthy overflow process. Furthermore, we define the objective function for the overflow phase of the ALFO as shown in (3). Here, $C_M(t)$ represents the number of attack flows at time t.

$$d_t = \max(0, C_M(t) - C_M(t-1)) \tag{1}$$

$$H = \begin{cases} \exp(1 - \max\{d_t\}_{t=1}^{T_{overflow}}), & T \le \text{idle\_time} \\ 0, & T > \text{idle\_time} \end{cases} \tag{2}$$

$$\max H \quad \text{s.t.} \begin{cases} C_M(t) \ge C_{max} - C_N \\ 0 < t < T_{overflow} \end{cases} \tag{3}$$

In the attack maintenance phase, the attacker can reduce the quality of service provided to normal flows by increasing the number of flow entries occupied by attack flows [20]. In other words, the proportion of attack flow entries in the flow table reflects the degree of attack effect. Therefore, we define the occupancy rate of attack flow entries, R, as shown in (4), which represents the percentage of attack flow entries in the flow table. Here, P_f denotes the survival rate of an attack flow during flow table eviction, and it exhibits different behavior under different flow entry eviction mechanisms. We will subsequently conduct a detailed analysis of P_f under various flow entry eviction mechanisms. In equation (4), a higher occupancy rate of attack flow entries, R, signifies a greater degree of attack effectiveness. Consequently, we define the objective function for the attack maintenance phase of the ALFO, as shown in (5).

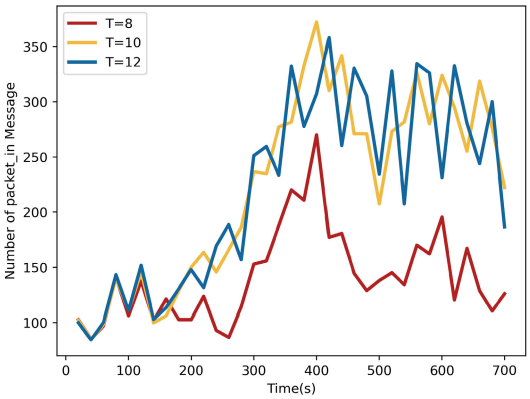


FIGURE 2. Changes in the number of packet_in messages for different attack cycle over time.

$$R = \frac{\sum_{j=1}^{C_M} P_f(f \in F_M)}{C_{max}} \tag{4}$$

$$\max R \quad \text{s.t.} \begin{cases} 0 < T < \text{idle\_time} \\ C_{max} - C_N < C_M \le \tilde{v}_{max} \cdot T \end{cases} \tag{5}$$

C. ATTACK MODE ANALYSIS

In the overflow phase, based on the default configuration policy of the switch, when a new flow arrives that does not match any flow entries in the flow table, the switch installs the corresponding flow entry, as indicated by the red arrow in Fig. 1. Additionally, the switch will delete flow entries that have not matched any packets within the timeout period. Therefore, when the attack cycle is greater than idle_time, the switch will delete all attack flow table entries in each attack cycle, leading to the generation of a large number of OpenFlow messages in each attack cycle, as indicated by the red arrows in Fig. 1, referring to equation (2), it can be observed that the stealthiness of the attack is significantly compromised. This paper conducted experiments comparing changes in the number of packet_in messages sent by the switch over time before and after the attack. The experiments involved $T = 8, 10, 12$ (the default timeout value of flow entry set to 10), aiming to assess the impact of the attack cycle on attack stealthiness, the results are shown in Fig.2.

From Fig.2, it can be observed that, compared to T=10 or 12, the number of packet_in messages sent by the switch is significantly lower when $T = 8$. When the attack cycle is smaller than the idle_time, the growth trend of

the number of attack flow entries in the current cycle compared to the previous cycle becomes more significant, further deteriorating the stealthiness of the attack. Therefore, by setting the attack cycle T slightly smaller than the idle_time (such as T = 8 or closer to 10 seconds) and adjusting the number of attack flows within a single attack cycle, the stealthiness H of the attack can be maximized.

In the attack maintenance phase, the flow table has already overflowed, and the controller starts to evict flow entries based on eviction mechanisms. The value of the objective function is closely related to the survival rate P_f of the attack flow f during flow entry eviction, and P_f is defined differently under different flow entry eviction mechanisms. Therefore, we analyze each flow entry eviction mechanism separately.

1) When FIFO (First-In-First-Out) is deployed, the controller evicts flow entries based on their positions in the flow table. The position of a flow entry corresponds to its installation time, whereas flow entries installed earlier have lower positions and are given higher priority for eviction. Assuming the desired number of flow table evictions is λ, it can be inferred that flow entries with positions less than λ in the flow table will be evicted. Therefore, we define the survival rate x_f of flow entry f during each eviction as shown in (6), where L_f represents the position index of flow f in the flow table (starting from 0). Assuming the number of batches of attack flows sent in a single attack cycle is d, which represents the number of times the flow table overflows within a single attack cycle, the survival rate P_f of attack flow entries within a single attack cycle can be defined as shown in (7). It should be noted that d takes values in the range $\lfloor 1, d_{max} \rfloor$, where $d_{max} = \frac{T}{T_{batch}}$ and T_{batch} represents the time spent by the application to send a single group of attack flows. From (7), it can be observed that increasing the number of batches d can increase the value of P_f. The maximum value of the objective function is achieved when $d = d_{max}$.

$$x_f = \begin{cases} 0, & L_f < \lambda \\ 1, & L_f \geq \lambda \end{cases} \tag{6}$$

$$P_f = \frac{\sum_{i=1}^{d} x_f^i}{d} \tag{7}$$

2) When Random is deployed, the controller randomly selects flow entries for eviction. Thus the probability of a single flow entry being evicted is $1/C_{max}$, where C_{max} is the total number of flow entries. The survival rate of an attack flow during flow table eviction P_f is defined as shown in (8). Since P_f remains constant, referring to (6), it can be observed that increasing the number of attack flows can increase the value of R. The maximum number of attack flows that can be sent within a single attack cycle is limited by the maximum flow transmission rate $\tilde{v}_{max}$, where $C_M^{max} = \tilde{v}_{max} \cdot T$ represents the maximum number of attack flows that can be sent within a single attack cycle, considering the constraint imposed by the application. The maximum value of the objective function is achieved when $C_M = C_M^{max}$.

$$P_f = \frac{C_{max} - 1}{C_{max}} \tag{8}$$

3) When LRU (Least Recently Used) is deployed, the flow entries with lower activity are preferentially evicted, as referred to [25]. A flow entry's activity corresponds to the flow's transmission rate. When the flow table overflows, the flows with higher transmission rates are more likely to survive, while those with lower transmission rates are more likely to be evicted. Therefore, we define the average transmission rate of attack flows $\widetilde{v_M}$ as shown in (9) and the survival rate of an attack flow during flow table eviction P_f as shown in (10). In this equation, $\check{v}$ represents the threshold transmission rate for flow survival during flow entry eviction, and β is a scaling factor with $\beta > 1$. For analysis purposes, let's consider β approaching infinity, which transforms P_f into I_f as shown in (11). Here, v_f represents the transmission rate of attack flow f. When $\check{v} > \{v_M^j\}_{j=1}^{C_M}$, employing a greedy approach, the attack flows can occupy at most $C_{max} - \sum_{i=1}^{C_N} v_N^i > \check{v}$ flow entries. This is because flow entries corresponding to flows with rates lower than $\check{v}$ will be immediately evicted. In this case, the maximum value of R is given by $\frac{C_{max} - \sum_{i=1}^{C_N} v_N^i > \check{v}}{C_{max}}$. When $\check{v} \leq \{v_M^j\}_{j=1}^{C_M}$, the attack flows can occupy exactly C_M flow entries. In this case, $\max R = C_M$. Hence, the maximum value of the objective function is achieved when $\widetilde{v_M} = \check{v}$.

$$\widetilde{v_M} = \frac{\sum_{j=1}^{C_M} v_M^j}{C_M} \tag{9}$$

$$P_f = \frac{1}{1 + e^{(\check{v} - \widetilde{v_M}) \cdot \beta}} \tag{10}$$

$$I_f = \begin{cases} 1, & v_f \geq r \\ 0, & v_f < r \end{cases} \tag{11}$$

Based on the above analysis, it can be inferred that attackers can adopt different attack modes to achieve maximum attack effect under different flow entry eviction mechanisms. Consequently, attack flows exhibit distinct features under different flow entry eviction mechanisms, making existing methods ineffective in mitigating ALFO.

IV. ALFO-GUARD FRAMEWORK DESIGN

A. SYSTEM OVERVIEW

As shown in Fig. 3, the ALFO-Guard comprises three components: the flow monitoring mechanism, the attack detection mechanism, and the attack mitigation mechanism. The overall process is as follows: Firstly, the controller queries the information about flow entries and switch ports for the attack detection mechanism (Step ①). Subsequently, the controller receives and parses the information provided by the switches, updating the dataset associated with each switch based on the parsing results (Step ②). Next, the attack detection mechanism constructs a current-time graph model for the switch based on the dataset (Step ③). Besides, Graph

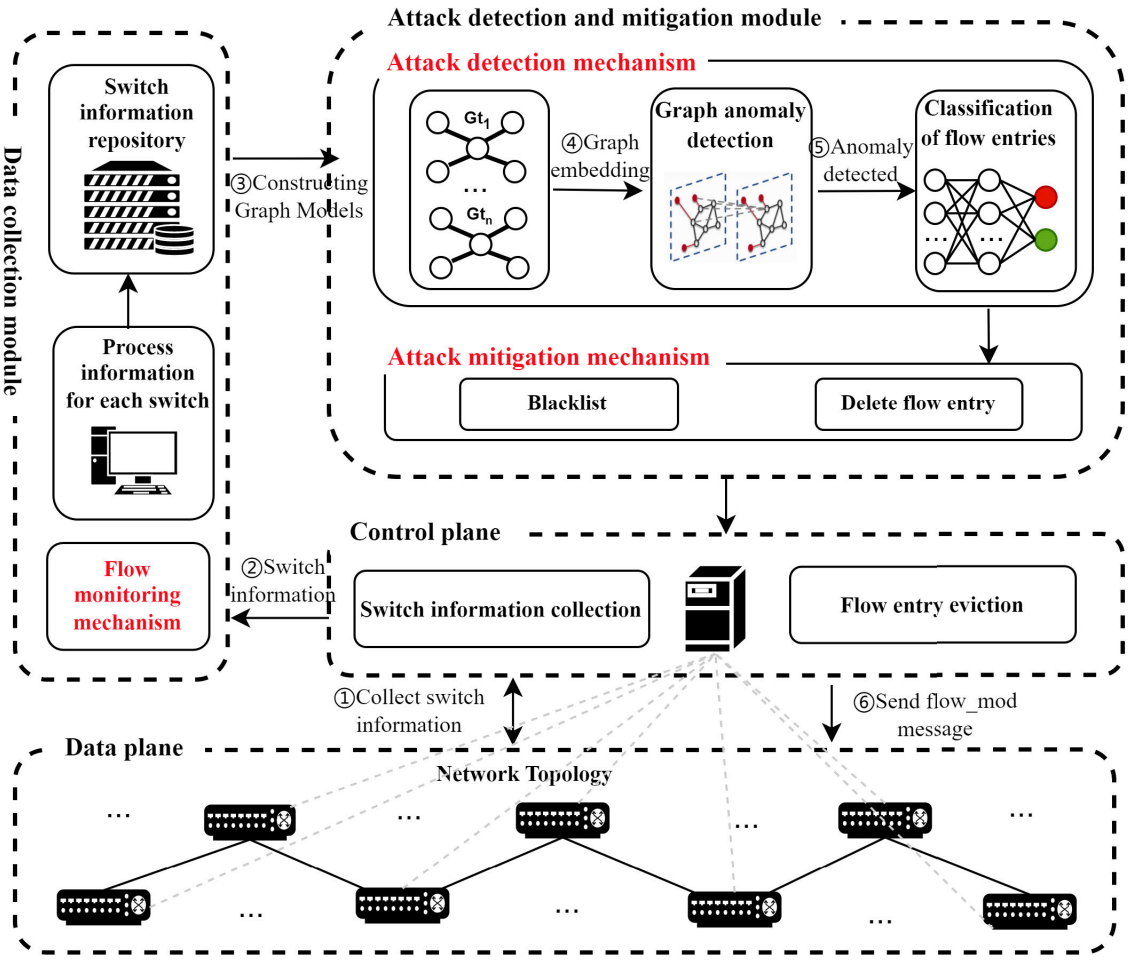
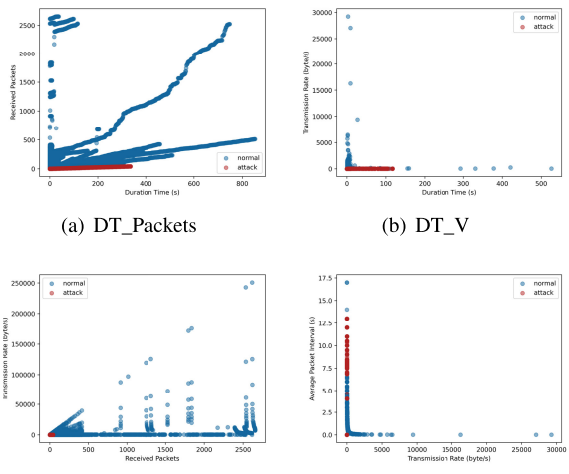


FIGURE 3. The framework of ALFO-Guard.

embedding vectors are generated using Graph Convolutional Networks (GCN) and attention mechanisms. The vectors are then input into the pre-trained graph anomaly detection model to detect anomalies (Step ④). If anomalies are detected, the flow entries classification model is triggered to identify attack flow entries (Step ⑤). Finally, the attack mitigation mechanism is triggered to delete the attack flow entries and add them to the blacklist (Step ⑥).

B. CONSTRUCTION OF GRAPH MODEL

To construct the current-time graph model, it is essential to extract eight features from each flow entry, as shown in Table 3. These features include duration time, received bytes, received packets, average packet bytes, the coefficient of variation of average packet bytes, average packet interval, the coefficient of variation of average packet interval, and transmission rate. It should be noted that apart from the observed variations in features between attack flows and normal flows (as shown in Fig. 4), there are additional variations that depend on the specific attack modes, which can affect the detection performance of flow feature-based detection methods under different attack modes [14].



(a) DT_Packets (b) DT_V

(c) Packets_V (d) V_API

FIGURE 4. Differences in features of normal flow and attack flow entries.

Therefore, when conducting attack detection, it is crucial to consider the flow features and the structural information among flow entries as a fundamental detection criterion. To achieve this, we extract flow features to construct graph

TABLE 3. Extracted flow entry features.

Feature Names	Description	Significance of Selection
DT (Duration Time)	Duration of Flow Entry Occupying Flow Table	The higher the value, the higher the likelihood that it is an attack flow
Bytes (Received Bytes)	Cumulative Received Bytes of the Flow Entry	The lower the value, the higher the likelihood that it is an attack flow
Packets (Received Packets)	Cumulative Matched Packets of the Flow Entry	The lower the value, the higher the likelihood that it is an attack flow
AB (Average Packet Bytes)	Average Packet Bytes of the Flow Entry Matching. AB=Bytes/Packets (AB=0 if Packets=0)	Attackers often aim to minimize their attack costs by creating attack packets that are as small as possible. Therefore, the lower the value of the average packet size, the higher the likelihood that it is an attack flow
BCV (The Coefficient of Variation of Average Packet Bytes)	Adding the newly calculated average packet bytes to the sequence each time a data packet is matched by the flow entry. The coefficient of variation of this sequence represents the BCV	Attackers often employ techniques such as randomizing packet lengths to evade detection. Relying solely on average packet size may not be sufficient for effective detection. However, it is important to note that attackers typically aim to minimize their attack costs, and once an attack is initiated, they are unlikely to reconstruct attack packets. Therefore, a lower BCV indicates a higher likelihood of being an attack flow
API (Average Packet Interval)	The average packet transmission interval for matching flow entries. API=DT/Packets (API=DT if Packets=0)	The closer the value is to the default idle timeout of the flow entry, the higher the possibility that it is an attack flow
ICV (The Coefficient of Variation of Average Packet Interval)	Adding the newly calculated average interval to the sequence each time a data packet is matched by the flow entry. The coefficient of variation of this sequence represents the ICV	When it comes to attack traffic, the intervals between the transmission of attack packets tend to be periodic. On the other hand, for normal traffic, the packet intervals are typically random. Therefore, if the ICV exhibits periodic patterns, it indicates a higher likelihood of being an attack flow
V (Transmission Rate)	Received bytes per unit time of the flow entry. V=Bytes/DT	The lower the value of the transmission rate, the higher the likelihood that it is an attack flow

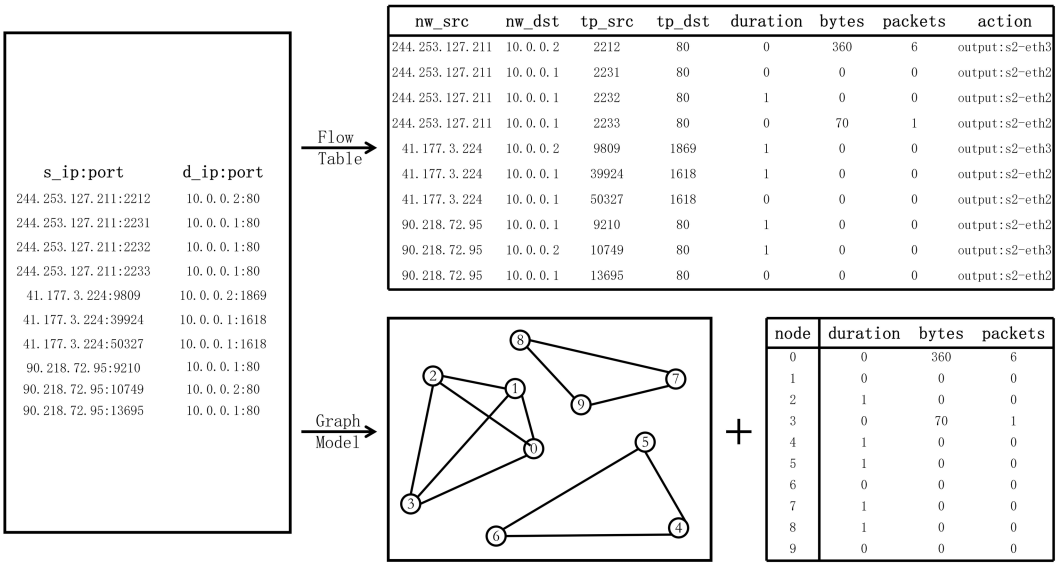


FIGURE 5. The structure diagram of the current-time graph model.

models for the switches based on the granularity of flow entry deployment by the controller.

The undirected graph $Gt = (V_t, E_t, L_G, L_N)$ represents the graph model of switch S at time t. Here, $V_t = \{v : X(v)\}$ denotes the set of flow entries, which includes all the flow entries in the switch S at the current time. $X(v) = [x1, x2, \ldots, x8]$ represents the extracted feature vector of each flow entry. $E_t = e_{ij}, i, j \in V_t$ represents the set of edges, where $e_{ij} = (v_i, v_j)$. $L_G = \{0, 1\}$ represents the set of graph labels. $L_N = \{L(v), v \in V_t\}$ represents the set of node labels. Fig. 5 illustrates the process of constructing the current-time graph model. When dealing with different flows, the switch matches them to different flow entries. Consequently, the current-time graph model adds new nodes and stores its flow feature accordingly. In Fig. 5, assuming there are ten flows, *FlowTable* represents the flow aggregation process in a switch, while *GraphModel* represents the process of graph model construction. The corresponding analysis of these two processes reveals that the number of flow entries in *FlowTable* is equal to the number of nodes in *GraphModel*, and the information of each flow entry in *FlowTable* corresponds to the information of each node in *GraphModel*. Therefore, the current-time graph model closely resembles the flow table of switches in both information and structure, enabling it to detect attacks effectively.

The construction process of the graph model is described in Algorithm 1. As shown in lines 5–9, the algorithm collects the flow table information from the switch at the current time. Each flow entry is treated as a node, and the algorithm reads the information of each flow entry to construct the node feature matrix and edge index matrix, thereby generating the current-time graph model. The construction method for the node feature matrix is presented in lines 10–21. Based on the information collected at the current time, the algorithm extracts the eight features for each flow entry and generates the feature matrix. Then, the algorithm normalizes the feature matrix by scaling each column using the minimum and maximum values, resulting in the node feature matrix of the current-time graph model. The construction method for the edge index matrix is shown in lines 23–35. Specifically, utilizing the information at the current time, the algorithm identifies flow entries with the same source IP address. Subsequently, it creates undirected edges for all nodes with the same source IP address and stores them in the edge index matrix.

Algorithm 1 Graph Model Construction

Require: The information about flow entries collected at the current time f.
Ensure: Current-time graph model Gt.

1: Initialize the network graph Gt
2: Initialize the flow entry feature list x
3: Initialize the dictionary ip_dict
4: Initialize the list of edges of the graph $edge\_index$
5: **function** main(f)
6: $Gt.x \leftarrow$ get_node_feature(f)
7: $Gt.edge\_index \leftarrow$ get_edge_index(f)
8: Save Gt
9: **end function**
10: **function** get_node_feature(f)
11: **for** p in Switch flow table information f **do**
12: $duration, bytes, bytes\_list, pakets, interval\_list \leftarrow$ Parse the flow table information p
13: $average\_bytes \leftarrow$ bytes divided by packets
14: $average\_time\_interval \leftarrow$ duration divided by packets
15: $rate \leftarrow$ bytes divided by duration
16: $bytes\_cv \leftarrow$ The coefficient of variation of $bytes\_list$
17: $interval\_cv \leftarrow$ The coefficient of variation of $interval\_list$
18: Encapsulate data such as duration, bytes, packets, average_bytes, rate, average_time_interval, bytes_cv, interval_cv to obtain node_features.
19: Add node_features to the flow entry feature list x
20: **end for**
21: Normalize the list x of flow entry feature by column
22: Save the list x of flow entry feature
23: **end function**
24: **function** get_edge_index(f)
25: **for** p in Switch flow table information f **do**
26: The source node ip src is obtained by parsing the flow entry p
27: **if** src not in ip_dict **then**
28: Initialize a list as the key of the src dictionary's value
29: **end if**
30: Binds the index of the current flow entry to the key-value pair
31: **end for**
32: **for** src in ip_dict **do**
33: **for** i in $ip\_dict[src]$ **do**
34: Other flow entries with the same source ip are obtained, according to the index i of the current src.
35: The indexes of flow entries with the same source ip are formed as edges and stored in edge set of the graph $edge\_index$
36: **end for**
37: **end for**
38: Saves a list of edge sets of the graph $edge\_index$
39: **end function**

C. ATTACK DETECTION MECHANISM

The attack detection mechanism can be divided into three stages: node embedding, graph embedding, and attack detection. Next, we will provide a detailed description of each stage.

1) NODE EMBEDDING STAGE

In this study, the graph convolutional networks (GCN) [39] are utilized to encode each node's neighborhood features and structural properties in the graph model, enabling node embedding. The aggregation function is defined as shown in (12). Here, $N(n)$ represents the set of first-order neighbors of node n, including node n itself. d_n is equal to the degree of node n plus 1. $W^{(l)} \in R^{D^l \times D^{l+1}}$ denotes the weight matrix associated with the lth GCN layer. $b^{(l)} \in R^{D^{l+1}}$ represents the bias. The activation function $f(\cdot)$ is applied, such as $ReLU(x) = \max(0, x)$.

$$conv(u_n) = f\left(\sum_{m \in N(n)} \frac{1}{\sqrt{d_n d_m}} u_m W^{(l)} + b^{(l)}\right) \tag{12}$$

2) GRAPH EMBEDDING STAGE

Firstly, based on the node feature matrix generated in the node embedding stage, we compute the weighted average of each node on each feature and generate a graph context vector through a non-linear transformation, as shown in (13). The graph context vector provides information about the structure and features of the graph. Next, using the graph context vector, we calculate attention weights for each node and apply the sigmoid function to ensure that the attention weights are within a range. Nodes that are similar to the global context will generate higher attention weights. Finally, we perform a weighted sum of the node feature matrix to obtain the graph embedding vector, as shown in (14). Where N represents the number of nodes in the graph. $u_n \in R^D$ is the embedding vector of node n, where D is the dimensionality of the embedding. $W \in R^{D \times D}$ is a learnable weight matrix. $\sigma(\cdot)$ denotes the sigmoid function.

$$c = \tanh\left(\frac{1}{N}\sum_{n=1}^{N} u_n W\right) \tag{13}$$

$$\begin{aligned} h &= \sum_{n=1}^{N} {u_n}^T \sigma(u_n c) \\ &= \sum_{n=1}^{N} {u_n}^T \sigma\left(u_n \tanh\left(\frac{1}{N}\sum_{m=1}^{N} u_m W\right)\right) \end{aligned} \tag{14}$$

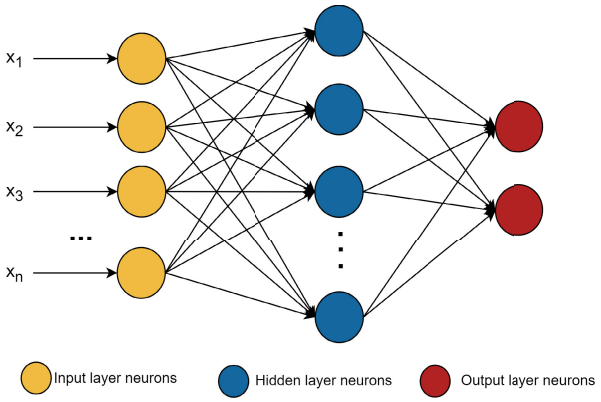


FIGURE 6. Two-layer fully connected neural network.

3) ATTACK DETECTION STAGE

In this stage, the graph embedding vectors generated in the graph embedding stage are used for classification, aiming to determine whether an attack has occurred at the current time. Specifically, we utilize cross-entropy as the loss function, as shown in (15). Here, $p(x_t)$ represents the predicted value at the current time t, and $q(x_t)$ represents the ground truth at the current time t.

$$\text{CELoss} = -\sum_{i=1}^{n} p(x_t) \bullet \log q(x_t) \tag{15}$$

If an attack occurs at the current time, we feed the node embedding matrix generated in the node embedding stage into a two-layer fully connected neural network to classify the flow entries. Based on the classification results, we can identify attack flow entries, as illustrated in Fig. 6. The two-layer fully connected neural network in Fig. 6 employs a non-linear activation function to enhance the expressive power of the model and models the relationships between nodes through fully connected layers.

D. ATTACK MITIGATION MECHANISM

The attack mitigation mechanism primarily consists of two operations: deleting attack flow entries and adding them to a blacklist. Upon detecting an attack, the mitigation module deletes each identified attack flow entry by sending OFPFlowMod messages to the switches. However, simply deleting the attack flow entries does not eliminate the effect of the attack, as the attack flows can continue to consume flow entries. Therefore, while deleting the attack flow entries, the mitigation module also keeps track of the number of times each flow entry is deleted within a certain period. If the deletion count exceeds a threshold, the flow entry is added to the blacklist to block the attack source.

V. EXPERIMENTS AND ANALYSIS

This paper introduces a novel adaptive low-rate flow table overflow attack (ALFO). To counter this attack, we propose a detection and mitigation framework called ALFO-Guard. In this section, we validate the effectiveness of ALFO, the effectiveness of ALFO-Guard, and the performance overhead through experiments. Code and data are available at https://github.com/446571357/ALFO

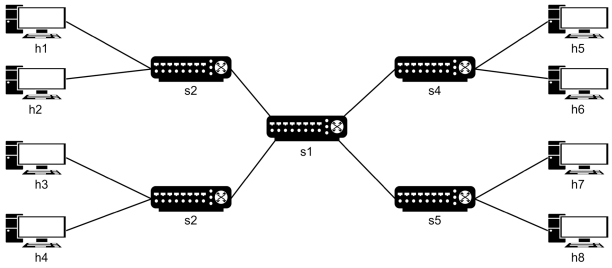


FIGURE 7. Experimental topology.

TABLE 4. Number of normal and attack flows in dataset.

Eviction Mechanism	Normal Flow	Attack Flow	Total Flow
FIFO	1202097	2773613	3975710
Random	1159647	2951133	4110780
LRU	1095370	2723557	3818927

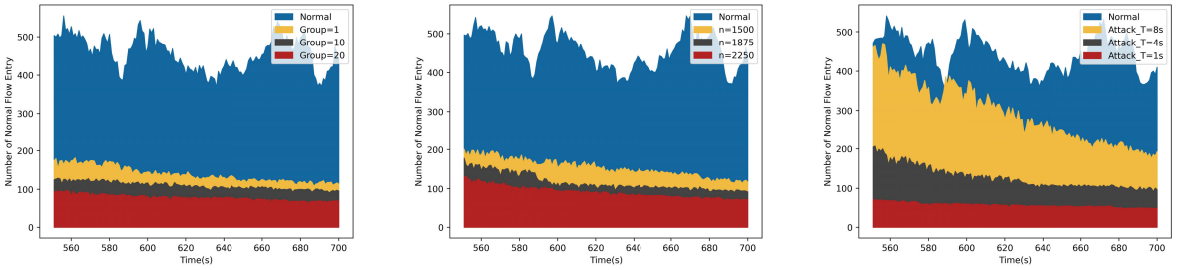
A. EXPERIMENTAL ENVIRONMENT

In our experimental setup, we utilize Mininet and Ryu controllers to simulate the SDN environment within a virtual machine. The virtual machine operates on Ubuntu 20.04 with a 4-core CPU and 8 GB of memory. The versions of Mininet and Ryu controller are 2.3.1b1 and 4.3.4, respectively. The software switch used is OpenvSwitch. On the Ryu controller, we run a custom controller program that implements the functionality of installing flow entries based on IP and port matching. The matching fields include *protocol*, *port_src*, *port_dst*, *ip_src*, and *ip_dst*. Since the proposed method in this paper relies solely on the flow features of a single switch and is independent of the network topology, we adopt the topology from reference [21] as our experimental topology, as depicted in Fig. 7.

In this paper, the capacity of the flow table is set to 1500, with a default idle timeout of 10 seconds [24]. To generate background traffic, we use the Tcpreplay tool [40] on host h8 to replay the IMC-10 data center network trace [41] dataset. For generating the attack traffic, we implement an attack program based on scapy [42] and launch the attack on host h1. Specifically, we simulate several groups of ALFO attacks under different flow entry eviction mechanisms: FIFO, Random, and LRU. We collect information about the flow entries and port status of the targeted switch $S2$ every second. Each group of attacks lasts for 700 seconds, with the attack initiated at the 150th second. The presence of attack flow entries in the flow table is used to set the graph labels and node labels are assigned based on whether the source IP is the attacking IP. The specific information of dataset collection is shown in Table 4, with a training set to a testing set ratio of 7:3.

B. VALIDATION OF THE EFFECTIVENESS OF ALFO

Firstly, we validate the proposed attack model by examining the effect of ALFO on the number of normal flow entries under different flow entry eviction mechanisms. (1) For the



(a) FIFO (b) Random (c) LRU

FIGURE 8. Attack effect verification under different parameters.

FIFO eviction mechanism, the attack parameter is the number of batches sent in a single attack cycle. We set it to be 1, 10, and 20. (2) For the Random eviction mechanism, the attack parameter is the number of attack flows. We set it to be 1500, 1875, and 2250. (3) For the LRU eviction mechanism, the attack parameter is the attack cycle. We set it to be 8 second, 5 second, and 1 second.

Specifically, we collect the number of normal flow entries after launching the attack for 400 second, which serves as the evaluation metric for Section III-C. The results are shown in Fig. 8. Under each flow entry eviction mechanism, regardless of the chosen attack parameters, the number of normal flow entries of the target switch is relatively small compared with the case of no attack, indicating that the attack decreases the forwarding efficiency of normal flows.

Furthermore, we observe that in the FIFO eviction mechanism (Fig. 8(a)), the attack has a more pronounced effect with a higher number of batches sent within a single cycle. In the Random eviction mechanism (Fig. 8(b)), the attack has a more pronounced effect with a higher number of attack flows. In the LRU eviction mechanism (Fig. 8(c)), the attack has a more pronounced effect with a shorter attack cycle. These observations align with the results derived in Section III-C.

Next, to demonstrate the effectiveness of ALFO, we compare it with two representative low-rate flow table overflow attack methods, Slow-TCAM [18] and LOFT [19]. Based on the results of the previous experiments, we set the attack parameters for ALFO as follows: For the FIFO eviction mechanism, the number of batches sent by ALFO within a single cycle is set to 20. For the Random eviction mechanism, the number of attack flows for ALFO is set to 2250. For the LRU eviction mechanism, the attack cycle for ALFO is set to 1 second.

To compare the effectiveness of the three attacks under different flow entry eviction mechanisms, we collected the number of flow entries and normal flow entries in the switch over time before and after the attacks. Additionally, we used D-ITG on host h2 to send 50 new flows, with each flow transmitting 1 MB of data. This operation was repeated ten times to calculate the average round-trip time (RTT) from sending to receiving a response for all flows, evaluating the effect of different attacks on the delay of normal applications. The results are shown in Figs. 9 to 11.

Figs. 9 and 10 demonstrate that under each flow entry eviction mechanism, all three attacks decrease the number of normal flow entries in the target switch and cause the flow table to reach its capacity limit compared to the typical scenario. Fig. 11 shows that all three attacks significantly increase the average RTT of new flows. These observations indicate that the attacks effectively degrade the quality of service for normal applications. Furthermore, compared to the attacks Slow-TCAM and LOFT, which do not consider flow entry eviction mechanisms, ALFO has a higher effect on the number of flow entries occupied by expected flows and significantly increases the forwarding delay of new flows. This suggests that ALFO outperforms Slow-TCAM and LOFT in terms of attack effectiveness.

C. VALIDATION OF ATTACK DETECTION MECHANISM

We utilize the following metrics to assess the effectiveness of detection methods:

Accuracy (Acc): The proportion of correctly classified samples. The calculation formula is defined as shown in (16).

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{16}$$

False Positive Rate (FPR): The proportion of misclassified samples among all normal samples. The calculation formula is defined as shown in (17).

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{17}$$

False Negative Rate (FNR): The proportion of misclassified samples among all attack samples. The calculation formula is defined as shown in (18).

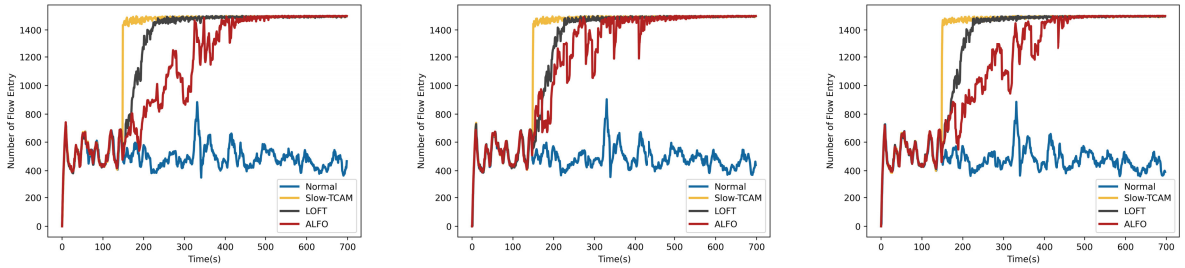
$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}} \tag{18}$$

F1-score: The harmonic mean of precision and recall. The calculation formula is defined as shown in (19).

$$\text{F1 - Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} \times \text{Recall}} \tag{19}$$

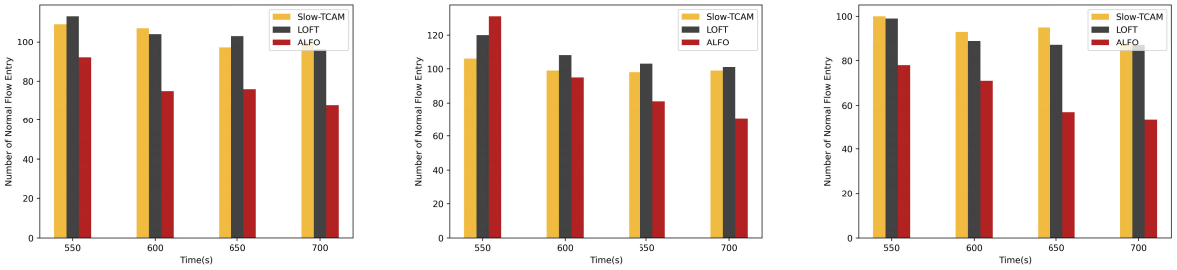
Precision: The proportion of true attack samples among samples classified as attack. The calculation formula is defined as shown in (20).

$$\text{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{20}$$



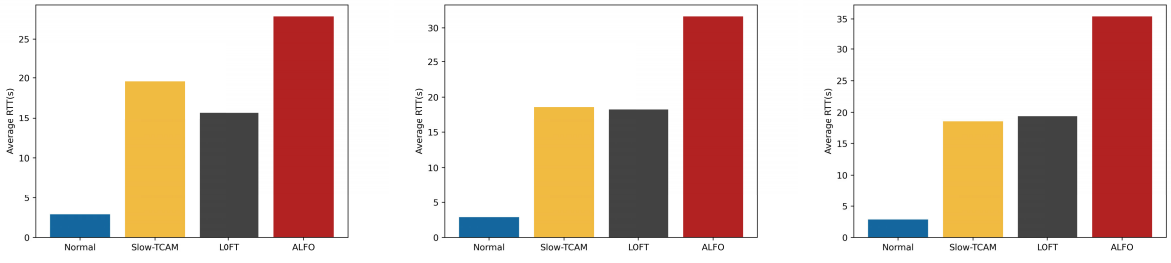
(a) FIFO (b) Random (c) LRU

FIGURE 9. Changes in the number of flow entries for different attack methods over time.



(a) FIFO (b) Random (c) LRU

FIGURE 10. Comparison of the number of normal flow entries under different attack methods (t = 550, 600, 650, 700s).



(a) FIFO (b) Random (c) LRU

FIGURE 11. Comparison of average RTT under different attack methods.

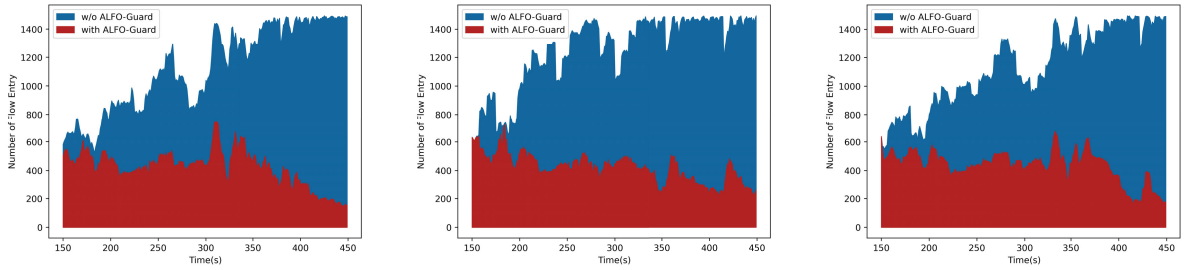
Recall: The proportion of correctly classified attack samples among all attack samples. The calculation formula is defined as shown in (21).

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{21}$$

To validate the detection performance of ALFO-Guard, this study trained the graph anomaly detection model and flow entries classification model under different flow entry eviction mechanisms. The detection results are shown in Tables 5 to 8. In Table 5, regardless of the flow entry eviction mechanism, the graph anomaly detection model achieves a 99% accuracy and a near-zero FNR, indicating its high accuracy in identifying attack situations. The graph anomaly detection model aims to determine whether the current situation is under attack. We aim to improve the sensitivity of the model to attack scenarios rather than misclassifying attack situations as usual. Therefore, FNR is a crucial detection metric.

As shown in Tables 6 to 8, the flow entries classification model exhibits an accuracy and F1 score of around 99%, indicating its accurate identification of attack flow entries. In terms of false positive rates, it can be observed that the flow entries classification model has a low misclassification rate for normal flow entries. Since the identified attack flow entries by the flow entries classification model will be deleted and blocked by the attack mitigation mechanism, a lower misclassification rate for normal flow entries implies minimal influence on normal flows. Furthermore, regardless of the flow entry eviction mechanism, the flow entries classification model shows minimal differences in its performance metrics. This indicates that the proposed method in this study is less susceptible to environmental influences and possesses high robustness.

To evaluate the capability of ALFO-Guard in detecting ALFO, we compare its detection performance with LOFT-Guard [23] and FTMaster [24]. Specifically, this study trains



(a) FIFO (b) Random (c) LRU

FIGURE 12. Comparison of flow entry numbers with and without ALFO-Guard.

TABLE 5. Graph anomaly detection model.

	Acc	FPR	FNR	F1-score
FIFO	99.619%	1.869%	0.0%	99.761%
Random	99.047%	4.386%	0.0%	99.395%
LRU	99.619%	0.897%	0.242%	99.758%

TABLE 6. Flow entries classification model (FIFO).

	Acc	FPR	FNR	F1-score
LOFTGuard	85.124%	42.844%	2.773%	90.122%
FTMaster	93.841%	6.69%	5.929%	95.52%
ALFO-Guard	98.993%	0.651%	1.161%	99.376%

TABLE 7. Flow entries classification model (Random).

	Acc	FPR	FNR	F1-score
LOFTGuard	87.812%	39.451%	1.476%	92.068%
FTMaster	95.179%	5.529%	4.543%	96.602%
ALFO-Guard	99.338%	0.8%	0.609%	99.54%

TABLE 8. Flow entries classification model (LRU).

	Acc	FPR	FNR	F1-score
LOFTGuard	97.847%	4.812%	1.084%	98.496%
FTMaster	98.924%	0.77%	1.2%	99.242%
ALFO-Guard	99.084%	0.928%	0.911%	99.357%

the detection models using the same dataset under different flow entry eviction mechanisms, and the detection results are shown in Tables 6 to 8. LOFTGuard and FTMaster exhibit significant differences in their detection results under different flow entry eviction mechanisms. This suggests that detection methods that overly rely on the features of attack flows are more susceptible to the influence of flow entry eviction mechanisms, resulting in lower robustness when dealing with ALFO as proposed in our study. On the other hand, ALFO-Guard constructs a current-time graph model based on the current moment, which not only collects flow entry information but also captures structural information embedded in flow entries. This captured structural information is not affected by flow entry eviction mechanisms. Therefore, regardless of the flow entry eviction mechanism, ALFO-Guard achieves better detection performance than other models in various metrics.

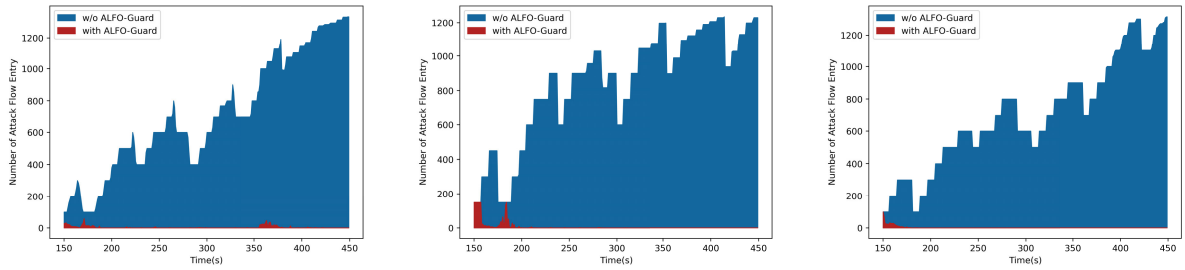
D. VALIDATION OF ATTACK MITIGATION MECHANISM

The attack was initiated at 150 seconds, and approximately 200 seconds later, the flow table started to overflow in the absence of ALFO-Guard deployment. To verify the mitigation performance of ALFO-Guard, this study deployed the number of flow entries and attack flow entries in the flow table of targeted switches over time under different flow entry eviction mechanisms with and without ALFO-Guard, as shown in Figs. 12 and 13.

In Fig. 12, without the deployment of ALFO-Guard, the utilization of the switch's flow table significantly increased shortly after the attack initiation. After a certain period, the number of flow entries reached the maximum limit of the switch's flow table, indicating overflow of the flow table. However, in the case of ALFO-Guard deployment, the number of flow entries remained lower, indicating that ALFO-Guard can effectively mitigate the occupancy of the switch's flow table by the attack flows.

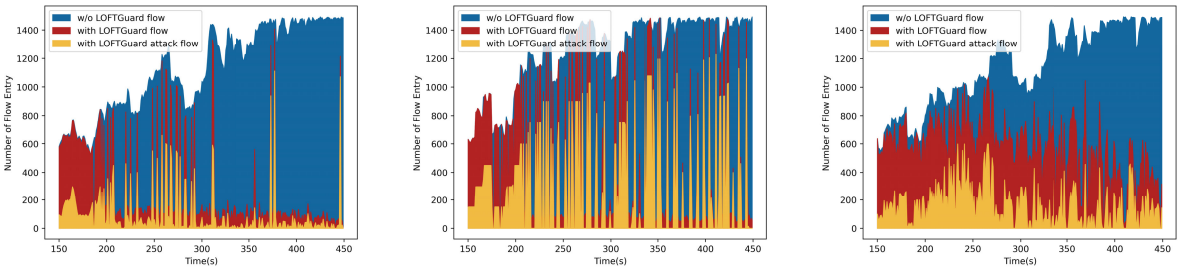
In Fig. 13, without the deployment of ALFO-Guard, the number of attack flow entries gradually increased, eventually reaching 1200 entries, occupying 80% of the flow table, greatly impacting the forwarding efficiency of normal flows. On the other hand, with ALFO-Guard deployed, the number of attack flow entries only existed in the flow table during the initial stages of the attack. After a certain period, their count approached zero. This indicates that the flow entries classification model can identify the attack flow entries promptly, and the attack mitigation mechanism can effectively delete and block the identified attack flow entries, restoring the flow table to its normal state.

To evaluate the mitigation capabilities of ALFO-Guard against ALFO, we compared the changes over time in the number of flow entries and the number of attack flow entries in the flow table after deploying ALFO-Guard, LOFTGuard, and FTMaster. The results are shown in Fig. 14. From the perspective of the number of flow entries, LOFTGuard exhibits significant fluctuations and has some impact on the number of normal flow entries. Besides, both ALFO-Guard and FTMaster maintain the number of flow entries at a normal level. This indicates that LOFTGuard and FTMaster methods can prevent flow table overflow and mitigate the effect of ALFO to some extent, which aligns with the detection results shown in Tables 5 to 7.

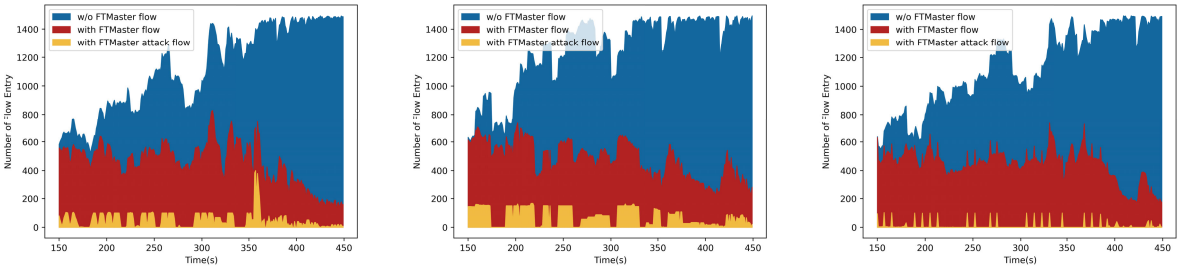


(a) FIFO (b) Random (c) LRU

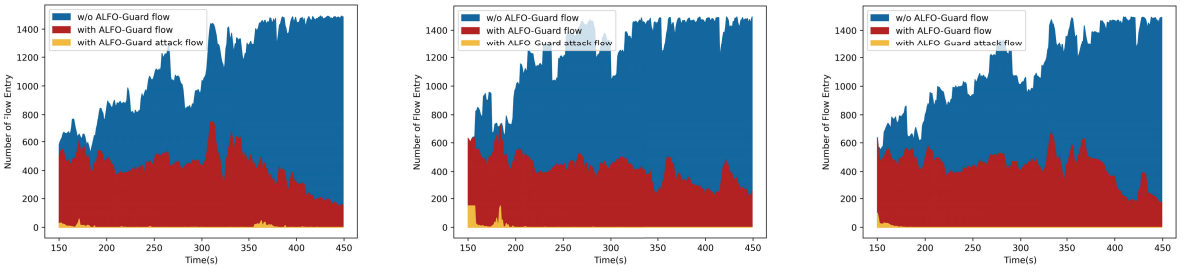
FIGURE 13. Comparison of attack flow entry numbers with and without ALFO-Guard.



(a) LOFTGuard-FIFO (b) LOFTGuard-Random (c) LOFTGuard-LRU



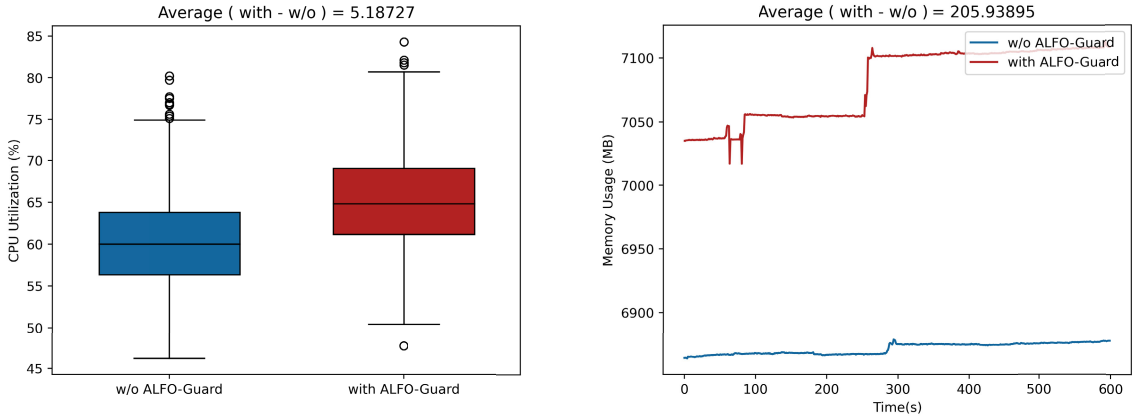
(d) FTMaster-FIFO (e) FTMaster-Random (f) FTMaster-LRU



(g) ALFOGuard-FIFO (h) ALFOGuard-Random (i) ALFOGuard-LRU

FIGURE 14. Comparison of flow entry numbers with different method.

Regarding the number of attack flow entries, LOFTGuard shows significant fluctuations, and FTMaster still has a certain number of attack flow entries. Additionally, LOFTGuard and FTMaster yield significantly different results under different flow entry eviction mechanisms, which is consistent with the detection results shown in Tables 5 to 7. In contrast to LOFTGuard and FTMaster, the number of attack flow entries for ALFO-Guard exists only during the initial stages of the attack. After a certain period, their count approaches zero, regardless of the flow entry eviction mechanism. This indicates that ALFO-Guard is less susceptible to environmental influences and more robust. Since ALFO-Guard has the lowest number of attack flow entries and provides a consistently stable mitigation effect under different flow entry eviction mechanisms, it demonstrates superior mitigation effectiveness compared to other methods.



(a) CPU Utilization

(b) Memory Usage

FIGURE 15. Performance overhead of ALFO-Guard.

E. PERFORMANCE OVERHEAD

We evaluated the CPU utilization and memory usage for ALFO-Guard, as shown in Fig. 15. The figure shows that compared to the scenario without ALFO-Guard deployment, the average CPU utilization only increases by approximately 5.19% when ALFO-Guard is deployed. However, the average memory usage increases by approximately 206 MB after deploying ALFO-Guard, suggesting the need for further reduction.

VI. CONCLUSION

This study has identified ALFO as a significant threat to the security of SDN. ALFO considers the influence of flow entry eviction mechanisms on attack effectiveness and adjusts the attack modes under different flow entry eviction mechanisms, thereby amplifying the effect of an attack. To defend against this attack, we designed the detection and mitigation framework ALFO-Guard. It extracts flow features from flow entries information in the switch and aggregates them into a current-time graph model. Then, combining graph neural networks, it performs graph anomaly detection and flow entry classification to identify attack flow entries. Finally, the attack can be eliminated by deleting the identified attack flow entries and blocking the attack flows. The effectiveness of ALFO and ALFO-Guard is validated through extensive experiments, and the experimental results demonstrate that ALFO-Guard can effectively defend against ALFO under different flow entry eviction mechanisms.

However, ALFO-Guard does have certain limitations. In future work, the focus will be on reducing the system overhead of ALFO-Guard and deploying it in real-world environments. Additionally, there are plans to explore and discuss more attack scenarios to enhance the applicability of ALFO-Guard against other types of attacks. Simultaneously, integrating data plane programmable technologies, such as P4, to optimize switches is a worthwhile research direction. For instance, by writing custom flow table operations, switches can perform real-time monitoring and analysis of network flows, enabling faster matching and filtering of attack traffic to ensure the availability of network services.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] N. McKeown, "Software-defined networking," *INFOCOM Keynote Talk*, vol. 17, no. 2, pp. 30–32, 2009.
[2] 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.
[3] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 623–654, 1st Quart., 2016.
[4] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Flow wars: Systemizing the attack surface and defenses in software-defined networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3514–3530, Dec. 2017.
[5] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful SDN data planes," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1701–1725, 3rd Quart., 2017.
[6] Y. Tzang, H. Chang, and C. Tzang, "Enhancing the performance and security against media-access-control table overflow vulnerability attacks," *Secur. Commun. Netw.*, vol. 8, no. 9, pp. 1780–1793, Jun. 2015.
[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.
[8] X. Shengxu, X. Changyou, Z. Guomin, S. Lihua, and H. Guyu, "Survey of openflow switch flow table overflow mitigation techniques," *J. Comput. Res. Develop.*, vol. 58, no. 3, pp. 586–603, 2021.
[9] R. Mohammadi, M. Conti, C. Lal, and S. C. Kulhari, "SYN-guard: An effective counter for SYN flooding attack in software-defined networking," *Int. J. Commun. Syst.*, vol. 32, no. 17, p. e4061, Nov. 2019.
[10] M. Zhang, J. Bi, J. Bai, Z. Dong, and Z. Li, "FTGuard: A priority-aware strategy against the flow table overflow attack in SDN," in *Proc. SIGCOMM Posters Demos*, Los Angeles, CA, USA, Aug. 2017, pp. 141–143, doi: 10.1145/3123878.3132015.
[11] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite cacheflow in software-defined networks," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 175–180.
[12] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Cacheflow: Dependency-aware rule-caching for software-defined networks," in *Proc. Symp. SDN Res.*, 2016, pp. 1–12.

[13] S. Xie, C. Xing, G. Zhang, and J. Zhao, "Research on table overflow Ldos attack detection and defense method in software defined networks," in *Big Data and Securit* (Communications in Computer and Information Science), vol. 1210, Y. Tian, T. Ma, and M. Khan, Eds. Singapore: Springer, 2020, doi: 10.1007/978-981-15-7530-3_7.

[14] S. Y. Khamaiseh, I. Alsmadi, and A. Al-Alaj, "Deceiving machine learning-based saturation attack detection systems in SDN," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2020, pp. 44–50.

[15] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2008.

[16] W. Jiang, "Graph-based deep learning for communication networks: A survey," *Comput. Commun.*, vol. 185, pp. 40–54, Mar. 2022.

[17] J. Leng, Y. Zhou, J. Zhang, and C. Hu, "An inference attack model for flow table capacity and usage: Exploiting the vulnerability of flow table overflow in software-defined network," 2015, *arXiv:1504.03095*.

[18] T. A. Pascoal, Y. G. Dantas, I. E. Fonseca, and V. Nigam, "Slow TCAM exhaustion DDoS attack," in *ICT Systems Security and Privacy Protection: 32nd IFIP TC 11 International Conference, SEC 2017, Rome, Italy, May 29-31, 2017, Proceedings 32*. Springer, 2017, pp. 17–31, doi: 10.1007/978-3-319-58469-0_2.

[19] J. Cao, M. Xu, Q. Li, K. Sun, Y. Yang, and J. Zheng, "Disrupting SDN via the data plane: A low-rate flow table overflow attack," in *Security and Privacy in Communication Networks: 13th International Conference, SecureComm 2017, Niagara Falls, ON, Canada, October 22–25, 2017, Proceedings 13*. Springer, pp. 356–376, doi: 10.1007/978-3-319-78813-5_18.

[20] W. Zhijun, X. Qing, W. Jingjie, Y. Meng, and L. Liang, "Low-rate DDoS attack detection based on factorization machine in software defined network," *IEEE Access*, vol. 8, pp. 17404–17418, 2020.

[21] S. Xie, C. Xing, G. Zhang, and J. Zhao, "A table overflow LDoS attack defending mechanism in software-defined networks," *Secur. Commun. Netw.*, vol. 2021, pp. 1–16, Jan. 2021.

[22] C. Xingshu, H. Qiang, W. Yitong, G. Long, and Z. Yi, "Research on low-rate DDoS attack of SDN network in cloud environment," *J. Commun./Tongxin Xuebao*, vol. 40, no. 6, pp. 210–222, 2019.

[23] J. Cao, M. Xu, Q. Li, K. Sun, and Y. Yang, "The attack: Overflowing SDN flow tables at a low rate," *IEEE/ACM Trans. Netw.*, vol. 31, no. 3, pp. 1416–1431, Jun. 2023.

[24] D. Tang, C. Gao, W. Liang, J. Zhang, and K. Li, "FTMaster: A detection and mitigation system of low-rate flow table overflow attacks via SDN," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 4, pp. 5073–5084, Dec. 2023.

[25] Y. Liu, Y. Wang, and H. Feng, "POAGuard: A defense mechanism against preemptive table overflow attack in software-defined networks," *IEEE Access*, vol. 11, pp. 123659–123676, 2023.

[26] M. Zhang, J. Bi, J. Bai, and G. Li, "FloodShield: Securing the SDN infrastructure against denial-of-service attacks," in *Proc. 17th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./12th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2018, pp. 687–698.

[27] S. Gao, Z. Peng, B. Xiao, A. Hu, Y. Song, and K. Ren, "Detection and mitigation of DoS attacks in software defined networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1419–1433, Jun. 2020.

[28] Z. Liu, Y. He, W. Wang, and B. Zhang, "DDoS attack detection scheme based on entropy and PSO-BP neural network in SDN," *China Commun.*, vol. 16, no. 7, pp. 144–155, Jul. 2019.

[29] S. Khamaiseh, E. Serra, and D. Xu, "VSwitchGuard: Defending OpenFlow switches against saturation attacks," in *Proc. IEEE 44th Annu. Comput., Softw., Appl. Conf. (COMPSAC)*, Jul. 2020, pp. 851–860.

[30] A. Ahalawat, K. S. Babu, A. K. Turuk, and S. Patel, "A low-rate DDoS detection and mitigation for SDN using Renyi entropy with packet drop," *J. Inf. Secur. Appl.*, vol. 68, Aug. 2022, Art. no. 103212.

[31] K. Hong, Y. Kim, H. Choi, and J. Park, "SDN-assisted slow HTTP DDoS attack defense method," *IEEE Commun. Lett.*, vol. 22, no. 4, pp. 688–691, Apr. 2018.

[32] R. Xie, M. Xu, J. Cao, and Q. Li, "Softguard: Defend against the low-rate tcp attack in SDN," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Oct. 2019, pp. 1–6.

[33] L. Zhou, M. Liao, C. Yuan, and H. Zhang, "Low-rate DDoS attack detection using expectation of packet size," *Secur. Commun. Netw.*, vol. 2017, pp. 1–14, Jan. 2017.

[34] D. Tang, Y. Yan, S. Zhang, J. Chen, and Z. Qin, "Performance and features: Mitigating the low-rate TCP-targeted DoS attack via SDN," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 1, pp. 428–444, Jan. 2022.

[35] L. Xiangju, L. Xiaohao, F. Xianjin, and S. Linsong, "Low-rate denial-of-service attack detection method under software defined network environment," *J. Comput. Appl.*, vol. 42, no. 4, p. 1301, 2022.

[36] D. Tang, X. Wang, Y. Yan, D. Zhang, and H. Zhao, "ADMS: An online attack detection and mitigation system for LDoS attacks via SDN," *Comput. Commun.*, vol. 181, pp. 454–471, Jan. 2022.

[37] M. Yu, T. Xie, T. He, P. McDaniel, and Q. K. Burke, "Flow table security in SDN: Adversarial reconnaissance and intelligent attacks," *IEEE/ACM Trans. Netw.*, vol. 29, no. 6, pp. 2793–2806, Dec. 2021.

[38] Y. Zhou, K. Chen, J. Zhang, J. Leng, and Y. Tang, "Exploiting the vulnerability of flow table overflow in software-defined network: Attack model, evaluation, and defense," *Secur. Commun. Netw.*, vol. 2018, pp. 1–15, Jan. 2018.

[39] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.

[40] *Tcpreplay*. Accessed: Nov. 18, 2023. [Online]. Available: https://tcpreplay.appneta.com/

[41] T. Benson. *Data Set for IMC 2010 Data Center Measurement*. Accessed: Nov. 18, 2023. [Online]. Available: https://pages.cs.wisc.edu/tbenson/IMC10_Data.html

[42] *Scapy*. Accessed: Nov. 18, 2023. [Online]. Available: https://scapy.net/

YING ZENG received the B.S. degree from the Institute of Information Technology, Guilin University of Electronic Technology (GUET), China, in June 2020, where she is currently pursuing the master's degree with the School of Computer Science and Information Security. Her current research interests include software-defined networking and network security.

YONG WANG received the Ph.D. degree from the East China University of Science and Technology, in 2005. He is currently a Full Professor with the School of Computer Science and Information Security, Guilin University of Electronic Technology. His current research interests include distributed storage systems, cloud computing, and information security.

YUMING LIU received the M.S. degree in computer science and technology from Guilin University of Electronic Technology, in 2017, where he is currently pursuing the degree with the School of Computer and Information Security. His current research interests include software-defined networking and network security.