FTOP: An Efficient Flow Table Overflow Preventing System for Switches in SDN

Dan Tang, Zhiqing Zheng, Keqin Li, *Fellow, IEEE*, Chao Yin, Wei Liang, and Jiliang Zhang, *Senior Member, IEEE*

***Abstract*—The Software-Defined Networking (SDN) is a new network framework widely adopted in data center networks that decouples the control plane from data plane to make network management easier. In SDN, OpenFlow is a mainstream southbound communication protocol for controllers and network devices. In an OpenFlow-supported SDN network, the control plane establishes connections with switches and installs flow entries in their flow tables to direct packet forwarding. Since the flow table built with the ternary content addressable memory (TCAM) has limited space, it is possible to overflow by Denial-of-Service attacks or Flash Crowds (FCs). In this article, we present FTOP, an eviction-based system to capture anomalies and prevent flow table overflow from Low-rate Flow Table Overflow (LFTO) attacks and FCs. FTOP has four modules: Predictor, Detector, Mitigator, and Preventer. Predictor monitors network traffic and produces estimation of the flow count. Detector calculates features of all flows and detects LFTO attacks. Mitigator calculates features of each flow and evicts malicious rules. Preventer calculates the significance score for each flow and evicts the low-scored flows. We introduce random forest classifiers in attack detection and mitigation. Simulation results demonstrate the effectiveness of FTOP in preventing flow table overflow, which proves FTOP a practical solution.**

***Index Terms*—Kalman filtering, low-rate flow table overflow attack, random forest, software-defined networking.**

I. Introduction

SOFTWARE-defined Networking (SDN) is a merging network framework that brings centralized control, direct programmability, and programmable configuration through making the control plane and data plane independent of each other [1]. In contrast to a traditional network architecture, SDN's data plane devices merely act on instructions from the control plane in order to forward packets, leaving sophisticated network operations like routing to the control plane.

To realize communications between the control plane and the data plane, many protocols have been proposed, such as the Extensible Message and Presence Protocol (XMPP) [2] and OpenFlow [3]. OpenFlow is now the most well-known southbound protocol that defines the flow table for packet forwarding and allows users to directly access and manipulate the network devices on the data plane.

SDN has benefited greatly from its innovation architecture, however, the OpenFlow could present a number of security risks [4]. Since OpenFlow utilizes flow tables to store essential information relevant to network configurations, the OpenFlow switch becomes a major target. Moreover, the network relies on the switches to make modifications on flow rules according to the controller's commands all the time, making switch security an even more important issue [5]. Physical switches typically store flow entries in the ternary content addressable memory (TCAM), which allows for wire-speed packet forwarding. The flow table can only install limited flow rules by virtue of the high cost of TCAM [6], rendering it vulnerable to Flow Table Overflow (FTO) attacks [7] and Flash Crowds (FCs).

In this article, we study two scenarios that lead to flow table overflow: the Low-rate FTO (LFTO) attacks and FCs. The LFTO attack is a low-volume attack that sends a small number of mismatched packets to trigger massive malicious rules installation and eventually leading to unavailable to install new rules on the flow table. Different from LFTO attacks, FCs refer to a variety of legitimate users accessing the service within a short period of time, resulting in an effect similar to Denial of Service attacks [8].

A multitude of proposed solutions by researchers aim to prevent flow table overflow, however, it is important to note that these solutions are not exempt from the following limitations:

- *Coarse-grained matching:* Most flow rule aggregation-based solutions assign specific rules for big flows while aggregating the rules for small flows into a default rule [9], leading to coarse-grained matching for packets of small flows [10].
- *Only Focus on a single scenario:* Most solutions only focus on a single scenario, as some of them work on improving flow table utilization under a normal network [11], [12] and they fail to prevent overflow under FTO attacks, while others only consider the attack scenario and may be unable to prevent overflow under normal network [13].

Manuscript received 12 January 2023; revised 11 June 2023; accepted 13 July 2023. Date of publication 21 July 2023; date of current version 30 April 2024. This work was supported in part by the National Key R&D Program of China under Grant 2020YFB1713400, in part by the National Natural Science Foundation of China under Grant 62122023, and in part by the Science and Technology Key Projects of Changsha City under Grant kq2208038. Recommended for acceptance by Dr. He Huang. *(Corresponding author: Jiliang Zhang.)*

Dan Tang, Zhiqing Zheng, and Jiliang Zhang are with the Hunan University, Changsha 410012, China (e-mail: dtang@hnu.edu.cn; zhengzq@hnu.edu.cn; zhangjiliang@hnu.edu.cn).

Keqin Li is with the State University of New York, New York, NY 10018 USA, and also with Hunan University, Changsha 410012, China (e-mail: lik@newpaltz.edu).

Chao Yin is with the Jiujiang University, Jiujiang 332006, China (e-mail: david_yin@jju.edu.cn).

Wei Liang is with the Hunan University of Science and Technology, Xiangtan 411199, China (e-mail: wliang@hnust.edu.cn).

Digital Object Identifier 10.1109/TNSE.2023.3297650

- *Unable to address the root causes of attacks:* Some solutions are based on attack flow migration, which routes the packets matched to flow rules stored in switches with insufficient memory resources to those switches with sufficient memory resources so that the flow entries will be stored in the switch which is not likely to overflow [14]. However, the migration mechanism is unable to fundamentally mitigate FTO attacks and may lead to multiple switches being overflowed [15].

In response to these issues, we propose FTOP, an online system to prevent flow table overflow in two scenarios: LFTO attacks and FCs. FTOP consists of four modules: *Predictor*, *Detector*, *Mitigator*, and *Preventer*. The core idea of FTOP is to prevent overflow with minimal evicting operations to the victim flow table. FTOP relies on *Mitigator* and *Preventer* to complete the eviction of malicious and less-important flow rules. Due to the high computation overhead of *Mitigator* and *Preventer*, we design *Predictor* and *Detector* to monitor the flow table and send alarms to activate *Mitigator* and *Preventer* only if the flow table may overflow. *Predictor* generates an estimate of the flow count for the next time stamp and activates *Detector* if the estimation exceeds the threshold. When *Detector* is activated, it polls the flow table and calculates features, and inputs them into the detection classifier. When the LFTO attack is confirmed, *Mitigator* will identify malicious rules and add these rules to an evict list. Otherwise, *Preventer* will be triggered to calculate significance score for each flow rule and evict the rules with a low significance score. With these four modules, FTOP can mitigate flow table overflow under the two scenarios effectively. We use Kalman filtering to predict the flow count for the next time stamp, and implement two random forest (RF) classifiers to achieve attack detection and malicious flow identification.

To assess the feasibility of FTOP, we performed four set of simulations on a virtual SDN network. For LFTO attack scenarios, deploying FTOP can detect attacks and identify malicious rules with an accuracy rate of up to 98%, and reduce the proportion of attack rules to 20.66%, which is more than 35% higher than existing methods. For FCs scenarios, deploying FTOP can reduce Table_Full messages to 411, which is more than 29% higher than existing methods. In brief, our contributions are as follows:

- We mitigate flow table overflow caused by both LFTO attacks and FCs, which appears to be the first solution of its kind.
- We propose an eviction-based system that only evicts malicious rules and a small number of small flows, the simulation results show that eviction operations do not add packet matching issues and maintain fine-grained packet matching.
- We design *Detector* and *Mitigator* to mitigate LFTO attacks and *Preventer* to prevent being overflowed by FCs, ensuring the security of the flow table under both attack scenarios and normal scenarios.
- We mitigate LFTO attacks by removing malicious flows from the flow table rather than aggregating or migrating, effectively reducing the proportion of malicious flows in the network.

This article is segmented into the following sections, arranged in the following manner. Section II provides background information on SDN and threat models. Section III discusses related work. In Section IV, we introduce techniques related to FTOP. Section V presents our countermeasure, FTOP, which mitigates LFTO attacks and prevents overflow caused by FCs. Section VI evaluates the offline and online performance of FTOP through simulations, and the last section, Section VII, provides the conclusion.

II. Background

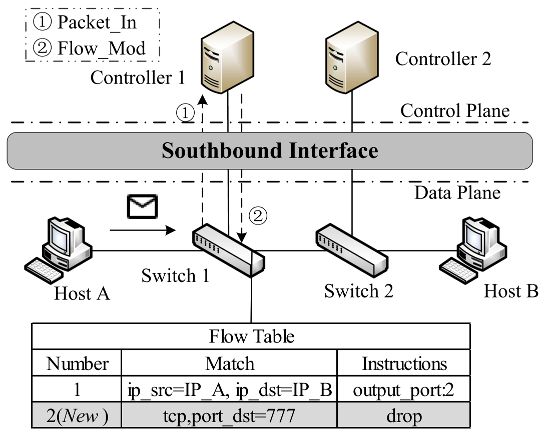
A. SDN Framework

SDN, which is originated from Standford University's Clean Slate project and was proposed by Professor N. Mckeow, revolutionizes network configuration and management by separating the tightly-coupled control and data plane in traditional networking devices, resulting in programmatically efficient network management. The application, control, and data plane are three vertically dividing planes that make up a typical SDN framework, according to ONF. Specifically, the control plane controls routing via OpenFlow and provides an interface to the north for developers to design and deploy their own functions and applications. The data plane, on the other hand, processes traffic solely in accordance with the instructions generated by the connected controller. Finally, the application plane comprises of applications that implement services through the network.

B. OpenFlow Protocol and the Data Plane

The OpenFlow protocol which is first proposed in 2008 and has now been updated to OpenFlow v1.5 is one of the most commonly used SDN southbound interface protocols. Through it, the control plane can direct data plane to perform packet forwarding and processing by a series of control events and decisions to add flow entries to the switches.

In the data plane, the flow table is a key data structure that enables the switches to forward and process incoming packets. A flow table consists of flows that direct the forwarding and processing of the matched packets. Each flow contains six components, which are match field, priority, instruction field, timeouts, and cookies. The most important parts are the match field and the instruction field. The match field specifies the packets that can match a flow, while the instruction field specifies how to handle matching packets. Fig. 1 illustrates how the data plane and control plane communicate with each other. When a data packet reaches a switch, the switch first parses its header and matches it with the flow entries in the flow table. If the packet matches a flow entry, the switch will execute the actions from the instruction field and specifies the output port for forwarding; Otherwise, the switch will send a Packet_In event to the control plane for further processing and routing instructions, and then the controller that is connected to the switch will reply with



Flow Table		
Number	Match	Instructions
1	ip_src=IP_A, ip_dst=IP_B	output_port:2
2(*New*)	tcp,port_dst=777	drop

Fig. 1. The communication between the control plane and the data plane.

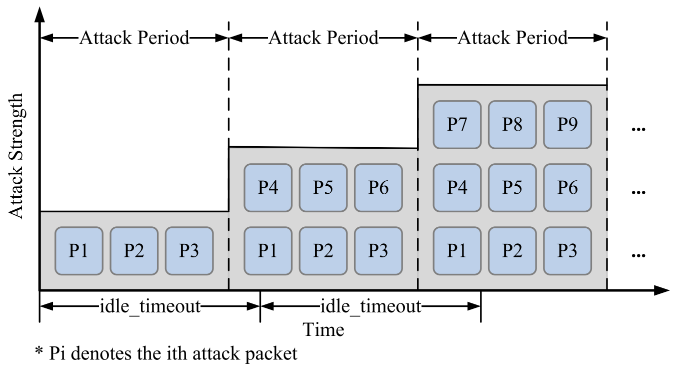


Fig. 2. The model of an LFTO attack.

a Flow_Mod event which defines the adding, modifying and removing of flow entries.

C. Threat Model

In this article, we discuss two kinds of data plane threats, which are LFTO attacks and FCs.

Different from the high-volume DDoS attack which overwhelm the resources of the switch's flow table immediately, the low-rate attack overflow the flow table in a more concealed way. The first low rate DoS attack targeting SDN switches is Slow-TCAM attack proposed by T. A. Pascoal et al. in 2017 [16]. Attackers recuit massive bots that each bot sends a well-constructed packet to the target switch without IP spoofing. In this article we study the LFTO attack proposed by Tang et al. [13]. The attacker sends specially designed data packets to the targeted switch and caused its flow table to install massive flow rules, and resends the packets during each *idle_timeout* to prevent being evicted. Fig. 2 depicts the model of the LFTO attack, which is characterized by three parameters: *period* (P), *step* (S), and *maximum attack strength* (MAS). P represents the attack period, which is intentionally set shorter than the *idle_timeout* to avoid eviction by the timeout mechanism. S indicates the growth of attack packets between adjacent periods (Ps). With the progression of the attack's duration, the number of malicious flow entries stored in the flow table rapidly increases, as determined by the parameter S. MAS represents the maximum quantity of malicious rules the attacker can generate. To overflow the flow table during each P, MAS must exceed the rest of the targeted flow table's resources. To make the attack more random and disordered, the P and S can be set to random. LFTO attacks cause the flow table to fail to install flow rules for legitimate new flows, so unmatched flows are forwarded to the controller, potentially causing saturation attacks to the control plane. In addition, due to the low attack rate of LFTO attack, the growth of attack flow entries is obviously lower than that of legitimate flow, which is difficult to be detected.

FCs refers to a phenomenon in which massive legitimate users access a service at the same time for a period of time, resulting in service performance degradation or even paralysis. In contrast to DDoS attacks, Flash Crowds traffic has a widely dispersed source IP distribution and packet size distribution. Different from the LFTO attack, the growth rate of flow rules is faster when there is flash congestion on the network, and the number of matched packets and bytes is larger, which does not have the effect of overflowing the flow table with small-sized data packets. Although Flash Crowd is non-malicious, it can still cause network performance degradation, so we hope to design an anti-overflow system that can mitigate the impact of it.

III. Related Work

In the past few years, there has been a growing body of research focused on investigating security issues within the SDN framework. In the section, we present relevant previous work that includes studies on attacks to the data plane and techniques for detecting and mitigating FTO attacks.

Attacks to the Data Plane: According to the literature [17], data plane attacks have three major types. The first one is the Denial-of-Service (DoS) attack. Attackers are able to monopolize the bandwidth between the controller and the switch [18], overflow the switch's flow table [19], occupy the bottleneck links in data plane [20], [21], [22], and consume the controller's available resources by Packet_In flooding [23]. The second one is topology poisoning attacks. The attackers can disrupt the controller's control ability of the entire network by constructing Link Layer Discovery Protocol (LLDP) packets to create fake links between SDN switches that are nonexistent to [24]. The third one is side-channel attacks. The attackers can estimate network configurations, such as flow table capacity [25] and communication records of switches [26] from the processing time of the controllers.

Mitigating Flow Table Overflow Attacks: Solutions to flow table overflow attacks can be categorized into four kinds, which are aggregation-based solutions, eviction-based solutions, timeout-based solutions, and migration-based solutions.

Aggregation refers to the use of a single flow rule to match multiple flows, effectively reducing the quantity of rules in the flow table. This kind of methods usually aggregate small flows and reserve specific rules for big flows [9], [27], some of them store important rules in TCAM while keeping less important rules in SRAM [11]. This kind of methods has lower fine-grained packet matching, and may lose small flows and increase the delay. In addition, this kind of methods are only practical for dealing with flow table overflow under normal network environments and are not capable of mitigating overflow from malicious users [28].

Timeout-based methods use adaptive algorithms to calculate the optimal idle_timeout [12], [29], or hard_timeout [30], [31] to evict the expired rules in the flow table. Similar to aggregation-based methods, the timeout-based methods can only prevent overflow under normal network conditions.

Migration-based methods need to adding routing rules to migrate the flow rules from a switch with sufficient resources [14], [32], [33]. This kind of methods may influence the routing of the entire network, leading to low versatility and deployability [28].

Eviction-based methods usually work by eliminating some chosen flow entries to release space for the installation of new rules. The eviction strategy of this type of methods can be divided into active eviction and passive eviction. Algorithms commonly used for passive eviction include least recently used (LRU) [34], first in first out (FIFO) [35], and random replacement [36]. Passive eviction fails to identify malicious rules, and therefore may mistakenly delete legitimate big flows causing higher network load. Active eviction often removes the flows with the least importance which are predicted to match the least packets in the future [37], [38], or the flows which are confirmed as malicious flows [39], [40], [41]. The current active eviction method mainly identifies small flows that match fewer packets and deletes them, and the accuracy is not high due to the dynamic network. In conclusion, existing eviction-based methods rarely target malicious rules, so most of the eviction rules are legitimate small flows, which cannot well mitigate flow table overflow attacks. In addition, eviction leads to higher communication overhead, which may influence traffic forwarding [42].

In this work, our goal is to develop a lightweight and effective system for preventing flow table overflow. We expect it can mitigate LFTO attacks and prevent being overflowed in a normal network, so we combine active and passive eviction strategies to achieve that goal with the least modifications on the switch. We simulated FTOP and evaluated its performance against the native overflow policies of OpenFlow (REFUSE and EVICT [43]), and three existing schemes from literature (SIFT [16], TableGuard [41], and SFTOGuard [13]). The simulation results demonstrate that FTOP outperforms all five methods in both LFTO attack and FCs scenarios.

IV. The Predicting and Attack Mitigation Techniques

This section introduces the techniques used in FTOP, namely Kalman Filtering and Random Forest.

A. Kalman Filtering

The Kalman filtering (KF) algorithm [44] proposed by Rudolf E. Kálmán in 1960, is a one-step state estimator generates estimates of unknown variables. Over these decades, the KF algorithm has been making contributions to numerous fields [45], and has been applied to attack detection problems [46]. Since

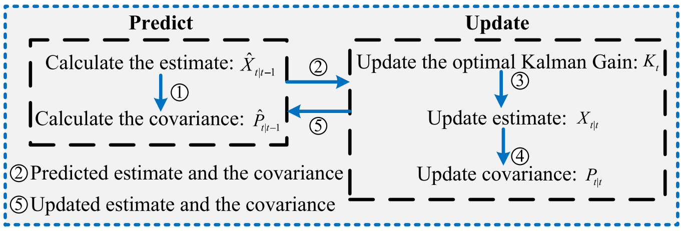

Predict
Update
Calculate the estimate: $\hat{X}_{t|t-1}$
Calculate the covariance: $\hat{P}_{t|t-1}$
Update the optimal Kalman Gain: K_t
Update estimate: $X_{t|t}$
Update covariance: $P_{t|t}$
①
②
③
④
⑤
②Predicted estimate and the covariance
⑤Updated estimate and the covariance

Fig. 3. The workflow of a KF estimator.

the KF algorithm has low memory consumption and fast calculation speed, we utilizes it for flow count prediction. The attack detection will not be carried out only if the prediction result indicates that the flow table may be overflowed, which effectively reduces the system overhead [47] while maintaining high-quality anti-overflow service for switches.

As shown in Fig. 3, the KF estimator consists of two phases, which are the predicting and updating phase.

The predicting phase utilizes the previously estimated state of the observed variable to estimate current state of the observed variable. Firstly, the estimator estimates the current state $\widehat{X}_{t|t-1}$ using the estimated previous state $X_{t-1|t-1}$ as follows:

$$\widehat{X}_{t|t-1} = F_t X_{t-1|t-1} + B_t u_t, \tag{1}$$

where F_t denotes the state transition model, u_t denotes the control vector, B_t denotes the control-input model applied to u_t. Then, the estimator generates the estimate covariance $\widehat{P}_{t|t-1}$ as follows:

$$\widehat{P}_{t|t-1} = F_t P_{t-1|t-1} F_t^T + Q_t, \tag{2}$$

where F_t^T is the transpose of F_t, and Q_t is the covariance of the process noise.

The updating phase updates estimates from the predicting phase once the measurements at time *t* are observed.

Firstly, the estimator updates the state estimate of time *t* the Kalman gain *K* as follows:

$$X_{t|t} = \widehat{X}_{t|t-1} + K_t \tilde{y}_t, \tag{3}$$

where $\tilde{y}_t$ is given by

$$\tilde{y}_t = z_t - H_t \widehat{X}_{t|t-1}, \tag{4}$$

where z_t is the observation of the true state X_t. Finally, the estimated covariance $P_{t|t}$ is obtained by

$$P_{t|t} = (1 - K_t H_t) \widehat{P}_{t|t-1}. \tag{5}$$

B. Random Forest Algorithm

For building our detection and mitigation models, we use the RF algorithm, which is an ensemble learning algorithm. Fig. 4 depicts a simplified RF workflow. The RF algorithm operates classification, regression, and other tasks by combining a multitude of individual decision trees (DTs). Each DT in the RF algorithm provides a class prediction, and the class with the majority vote is the prediction of the RF model. Due to the low correlation between DTs, the individual errors from the DTs

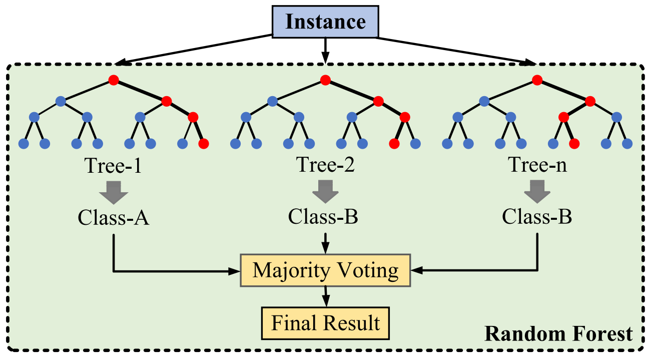


Fig. 4. The workflow of RF algorithm.

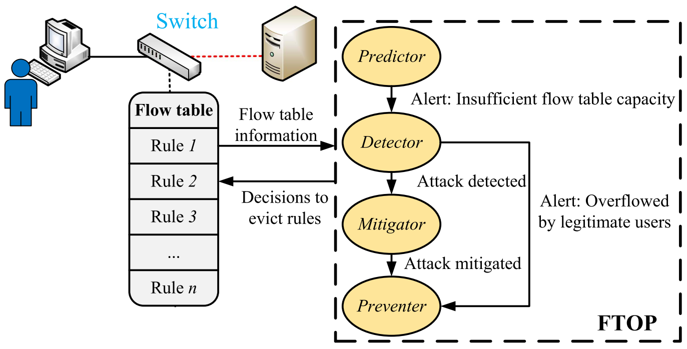


Fig. 5. System workflow.

will not influence the prediction result of the entire model, which brings high accuracy, good tolerance for outliers, and the ability to avoid overfitting.

V. System Design

A. System Overview

We first introduce the general overview of FTOP in this section, and then introduce each modules in brief. We define in this work that preventing overflow refers to preventing flow table usage from reaching more than 98%. As is shown in Fig. 5, FTOP has four modules, namely *Predictor*, *Detector*, *Mitigator*, and *Preventer*. *Predictor* polls the quantity of the flow entries and produces the estimated quantity of flow entries at time *t+1* with a KF estimator. If the estimate is larger than the threshold, *Detector* will be triggered. *Detector* first polls the flow table of the monitored switch and extracts its features for detection. Then, the module uses a trained RF classifier to confirm whether the monitored switch is been attacked. If the attacks occur, *Mitigator* is activated and traverses the flow entries. The flow rules which are identified as malicious rules by the mitigation model will be added to an evict list to be evicted. *Preventer* is active at all time to prevent flow tables overflowed by legitimate rules. With these four modules, FTOP can prevent flow table overflow under two circumstances, which are the LFTO attacks and FCs.

Algorithm 1: Predict the Number of Flow Rules.

```
Input: The size of the sliding window $Size$,the step of
       the window $step$, sampling interval
       $\Delta t$,threshold to activate $Detector$ $TH$,transition
       matrix of the KF estimator $mat_T$, observation
       matrix of the KF estimator $mat_O$
Output: Switch status $status$
1 Initialize $flowCount, kf$;
2 Function Predict($Size, step, \Delta t$):
3     $flowCount$ ← Collect flow count into an window
        of size $Size$; Window slides $step$ steps;
4     $kf$ ← KalmanFilter($mat_T$,$mat_O$,$flowCount$);
        $mean$, $covariance$ ← kf.filter($flowCount$);
        $Estimate$,$next\_covariance$ ←
        kf.filter_update($mean[-1]$,$covariance[-1]$);
5     if $Estimate > TH$ then
6         $status$ ← unsafe;
7     else
8         status ← safe;
9     $sleep(\Delta t)$;
```

B. Predictor Module

Algorithm 1 illustrates the workflow of *Predictor*. Firstly, the module polls the switch at Δt seconds interval, obtains the flow count and collects it into the sliding window *flowCount*. Secondly, it feeds *flowCount* to a KF estimator to produce estimations of the hidden state at the current time stamp. Then, the estimator predicts the state *Estimate* and calculate the covariance *next_covariance* at the next time stamp with the previous estimate and covariance. Finally, the status of the flow table is confirmed with a threshold *TH*. If the *Estimate* exceeds *TH*, *Predictor* considers that the flow table may overflow, and the *status* is set to unsafe to activate *Detector*. The *TH* is given by

$$TH = \begin{cases} \alpha TH & counter = 2 \\ TH0 & counter = 0, 1 \end{cases} \tag{6}$$

where $counter$ counts the times that *Detector* detects LFTO attacks and $counter = 2$ denotes *Detector* detects LFTO attacks twice continuously, α is set to 0.7, and $TH0$ is set to 80% of the flow table capacity.

We define the initial value for TH to 80% of the flow table is that it is likely to be overflowed with less than 20% available space. The sampling interval Δt is set to 1 s. To implement the KF estimator, we use *pykalman* [48]. With *Predictor*, the system turns to attack detection only if the flow table is likely to be overflowed, which effectively reduces the system overhead.

C. Detector Module

The workflow of *Detector* is shown in Algorithm 2. It detects whether flow table overflow attacks are launched and activates other modules to maintain flow table availability. To detect LFTO attacks, Detector first polls the switch's flow table to obtain the features of the flow table, which are packet numbers

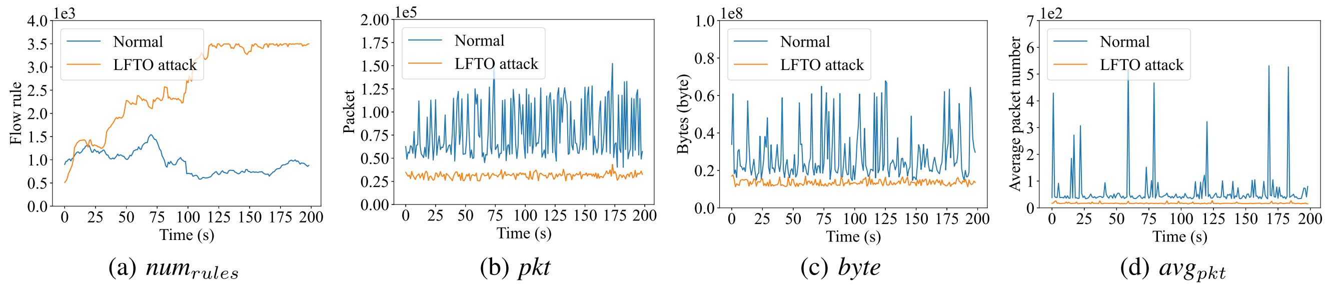


Fig. 6. Flow table features under normal conditions and LFTO attacks.

Algorithm 2: Detect LFTO Attacks.

```
    Input: $status$,$RF$,$counter$
    Output: detection result $res$, $counter$
1   Initialize $num_{rules}$, $pkt$, $byte$, $avg_{pkt}$;
2   Function Detect($status$, $RF$, $counter$):
3       $fc$ ← Collect flows with ovs-ofctl command;
        $num_{rules}$, $pkt$, $byte$ ← Extract detection features
        from $fc$;
4       $avg_{pkt}=\frac{pkt}{num_{rules}}$;
5       $res$=$RF$.predict([$num_{rules}$, $pkt$, $byte$, $avg_{pkt}$]);
6       if $res$==1 then
7           $counter$+=1; Activate Mitigator;
8       else
9           $counter$=0;
```

pkt, bytes *byte*, number of flow rules num_{rules}, and average packet number avg_{pkt}. The average packet number is given by

$$avg_{pkt} = \frac{pkt}{num_{rule}}, \tag{7}$$

where num_{rules} is the number of flow rules. Then, the features for detection are served as the input of the detection classification model to determine switch's status. If the model determines that LFTO attacks occur, the *counter* will be incremented by 1, and *Mitigator* will be activated to identify and evict malicious rules. Otherwise *counter* will be set to 0.

Fig. 6 shows the flow table features under flow table overflow attacks and without attacks. As shown in the figure, the above features are able to well distinguish between the flow table under normal conditions and LFTO attacks. The orange line denotes the displayed features under attacks while the blue line stands for normal conditions. As shown in Fig. 6(a), under normal network, num_{rules} does not exceed 1500 while that is obviously larger under LFTO attacks. As the attack lasting time grows, num_{rules} increases stepwise, which is in line with the attack model discussed in Section II. Since we modified the source port of attack packets to overflow the flow table with minimum packets, *pkt* and *byte* under LFTO attacks are much smaller than those under normal network, as shown in Fig. 6(b) and 6(c). In Fig. 6(d), under LFTO attacks, massive attack flows are added to the flow table that only match a minor quantity of packets and rules for legitimate packets can not be installed, leading to a low avg_{pkt} since *pkt* declines and num_{rules} increases. Therefore, the LFTO attacks can overflow the flow table with a small cost, which sends the least quantity of packets to construct a large quantity of flow entries.

With *Detector*, FTOP can confirm whether the flow table is overflowed by legitimate users or malicious attacks. If it is overflowed by legitimate flow entries, *Preventer* will be triggered to evict small flow rules which are less important to free the flow table space. If the flow table is overflowed by attackers, *Mitigator* will be triggered to identify malicious rules and evict them.

D. Mitigation Module

Mitigator is activated when *Detector* sends the being attacked signal to it. Firstly, this module requests the switch for detailed flow table information, which includes every flow entry in the switch's flow table. Then, it traverses the obatined flow rules, extracts the features of each rule, and enters the features into the RF classifier which is trained to identify malicious rules. If a flow rule is identified as a malicious rule, it will be added to an evict list and later deleted. Finally, if the quantity of flow rules still exceeds *TH*, *Preventer* will be activated.

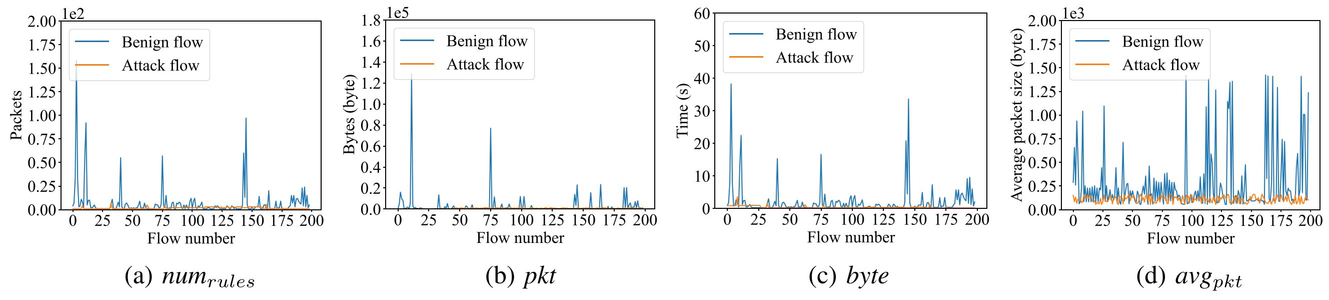
The features extracted are the number of packets *pkt*, bytes *byte*, source and destination ports, source and destination IPs, average packet arrival interval (*APAI*), and average packet size (*APS*). *APAI* is given by

$$APAI = \frac{pkt}{duration}, \tag{8}$$

where *duration* is the valid time of the flow entry.

Fig. 7 shows the above features of attack rules and benign rules. Since the LFTO attacks have a short attack period, the *APAI* of a malicious rule is much larger than that of a benign rule. To overflow the flow table, the attacker sends a major quantity of unmatched data packets with randomized ports to the targeted switch to trigger the installation of massive flow rules corresponding to the malicious packets. We can see that the selected features can well distinguish between the attack flows and the legitimate flows.

With *Mitigator*, FTOP can identify malicious flows with high accuracy, which enables FTOP to achieve precise mitigation from attackers and has the least impact on the matching of legitimate packets.



(a) num_{rules} (b) *pkt* (c) *byte* (d) avg_{pkt}

Fig. 7. Mitigation features of the attack flows and benign flows.

E. Preventer Module

Preventer is activated under two circumstances, which is 1) *Predictor* predicts that the estimated quantity of flow entris at the next time stamp will exceed the threshold *TH* and *Detector* confirms no LFTO attacks, and 2) *Mitigator* evicts the malicious flow entries identified by our mitigation model and after this process, the quantity of flow entries still exceeds *TH*. This module has two main steps. Firstly, the module traverses all the flow entries in the switch's flow table, calculate the significance score (*ss*) of each rule. The formula of *ss* is given by

$$ss_i = \frac{\theta}{\alpha+\beta} \times \frac{nx_i}{\sum_{i=1}^{n} x_i} + \frac{\beta}{\theta+\beta} \times \frac{ny_i}{\sum_{i=1}^{n} y_i}, \tag{9}$$

where x_i is the number of packets that the flow rule *i* matches, y_i is the quantity of bytes that $rule_i$ matches, θ and β are the coefficient of variation of *x* and *y*.

The coefficient of variation (CV) is a statistical measure that expresses the degree of variation relative to its mean value. It is calculated as the ratio of the standard deviation to the average value of the data and has been frequently utilized for weight assignment. The larger CV of a feature indicates that this feature carries more information than other features and is of greater significance, and thus the weight of the feature is given a greater value. For the two features, the quantity of *pkt* and the quantity of *byte*, we do not manually set the fixed weights, but use their CVs to produce self-adaptive weights, which makes the *ss* always useful to measure whether the flow rule is taking up significant amounts of network capacity and can not be evicted.

Fig. 8 shows the distribution of *ss*. It is evident that the flows which match more packets and bytes have higher *ss*, with an average of 64.72, and we regard these flows as big flows that take on heavier data transfer task that should not be evicted. For legitimate small flows and attack flows, the average *ss* is 0.52. These flows are with less significance and have little influence on the network if they are evicted. Once the significance score (*ss*) of all flow entries have been calculated, the module proceeds by sorting the rules according to their respective scores and subsequently removing the ones with the lowest scores. This process is aimed at ensuring that the flow table usage after the mitigation is reduced to a level that is below the predetermined threshold *TH*, thereby effectively managing the excess flow entries and optimizing the switch's performance and overall performance of the SDN network.

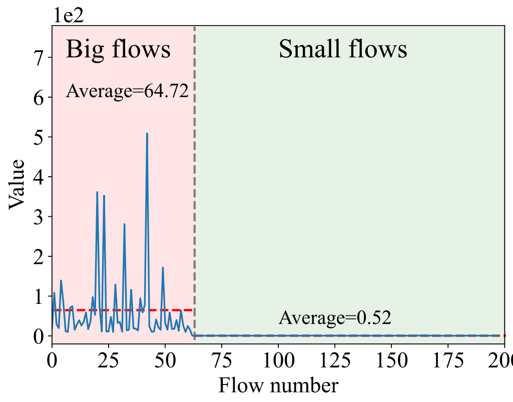


Fig. 8. Distribution of *ss* of the flow rules.

VI. Evaluation

A. Simulations Setup

The simulation environment specifications for our experiments were an Intel Core i5-7500 CPU @ 3.40 GHz, 64-bit operating system with 32 GB RAM. We installed Mininet [49] version 2.3.0d6 and Ryu controller [50] version 4.34 on Ubuntu version version 18.04.3 LTS. The switches in the topology are all OpenvSwitch version 2.5.9, and we followed the OpenFlow version 1.3 protocol.

We tested our system with the dumbbell topology, which is shown in Fig. 9. The topology contains 2 switches, 4 hosts, and 5 links. The link bandwidth between switches is 1 Gbps, and the link bandwidth between switches and hosts is 10 Gbps. And the flow table space of S1 is set to 3500, which denotes S1 can install no more than 3500 flow rules. The idle_timeout of all flow rules is set to 10 s, which means a flow rule will be evicted if it matches no packet after the given idle_timeout. To ensure interconnection between hosts, IP addresses of switches and hosts are all under the 10.0.0.0/12 network segment. Additionally, all the switches are connected to an Ryu controller. h3 is an attacker and sends malicious packets with spoofed IPs to h4 the server. h1 sends legitimate packets to the network.

To evaluate the effectiveness of FTOP, four groups of simulations were conducted using various parameters, as shown in Table I. The parameter *T* of every attack launched in each group is a random value within the range in the cells, and *t* is the average time from the initiation of attacks to the overflow

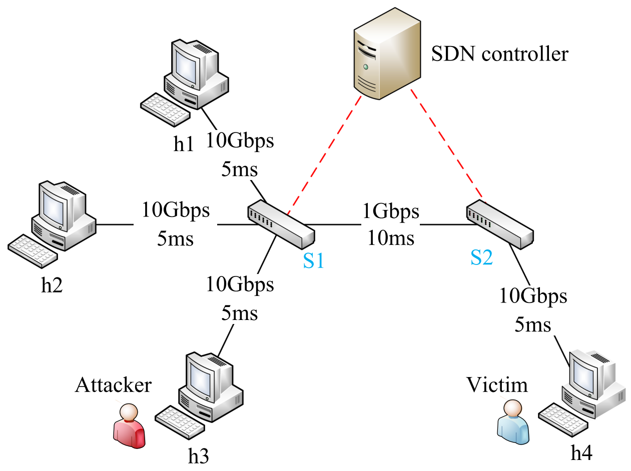


Fig. 9. Network topology for evaluation.

TABLE I
PARAMETERS SETTING IN EACH SIMULATION GROUP

Group Number	Attack type	T (s)	S (pkt/T)	M (pkt)	t (s)
1	LFTO	[5.5,9]	75	1700	60.5
2	LFTO	[6,9.5]	100	1700	41.1
3	LFTO	[6.5,10]	125	1700	27.4
4	Flash Crowd	1	2000	-	18.4

of the switch's flow table in each set of simulations. The first three groups of simulations are used to verify whether FTOP is effective for LFTO attacks and the fourth group of simulations is used to confirm whether the method is effective for FTO attacks from FCs.

B. Dataset

Attack traffic: We use *Python Scapy* to generate the LFTO attack on h3 and send the packets to h4 to overflow the flow tables of S1.

Background traffic: Real-world network traffic trace IMC-10 [51], [52] is used for evaluation. The benign traffic for our experiments was obtained from the *pt1* of *univ2* trace from the IMC dataset, while we filled the remaining load of the packets with the anonymous trace. To replay the trace, we utilized *tcpreplay* [53]. The legitimate packets were sent from *h1* to *h4*. For each trace, the playback time is 3600 s, and the speed is 1000 pps. To simulate FTO attacks from FCs, we replayed the trace from *h2* to *h4*, the packet replay rate of which is 2000pps.

Dataset: To build a dataset for detection, we polled *S1*'s flow table every 1 s and collected the number of flow rules to build a raw dataset. Then, to label the raw dataset, we defined the label of positive samples as "1" and negative samples as "0". The detection dataset contains 17838 samples, where 8980 of them are positive samples. To build a dataset for mitigation, we first sampled the flow table of *S1* every 10 s and collected each flow rule in the flow table to build a raw dataset, where 10 s is the idle timeout of the flow rules. Then, we labeled the data, where "1" stands for positive sample data, and "0" stands for negative

TABLE II
RESULTS OF OFFLINE DETECTION

Modol	Accuracy	Precision	Recall	F-score	FPR	MCC
AdaBoost	0.9717	0.9614	0.9745	0.9679	0.0306	0.9426
GB	0.9719	0.9670	0.9697	0.9684	0.0264	0.9431
XGBoost	0.9727	0.9617	0.9747	0.9682	0.0308	0.9427
SVM	0.8961	1.0000	0.8106	0.8954	0.0	0.8117
KNN	0.7383	0.5875	0.7694	0.6662	0.2778	0.4685
RF	**0.9866**	**0.9765**	**0.9932**	**0.9848**	**0.0119**	**0.9729**

sample data. The mitigation dataset has 237143 samples in total, where 162005 of them are positive.

C. Evaluation Metric

So as to appraise the efficiency of our designed system, we adopt six metrics, which are Accuracy, Precision, Recall, F-score, False Positive Rate (FPR), and Matthews correlation coefficient (MCC). The formulas to calculate these six metrics are given by

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}. \tag{10}$$

$$Precision = \frac{TP}{TP + FP}. \tag{11}$$

$$Recall = \frac{TP}{TP + FN}. \tag{12}$$

$$F\text{-}score = \frac{Recall \times Precision}{Recall + Precision} \times 2. \tag{13}$$

$$FPR = \frac{FP}{TN + FP}. \tag{14}$$

where TP, FP, TN, FN represent true positives, false positives, true negatives, and false negatives, respectively.

It can be inferred from their equations that higher Accuracy, Recall, Precision, and MCC values correspond to superior classification performance, while a lower FPR value denotes a lower probability of error.

D. Performance of the Proposed System

Offline Detection Evaluation: We conducted offline detection evaluation before we deployed FTOP to SDN switches. To show the effectiveness of the detection method utilized in FTOP, we compared it with five machine learning algorithms implemented with *scikit-learn*, which are Adaptive Boosting (AdaBoost), Gradient Boosting (GB), Extreme Gradient Boosting (XGBoost), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and. All the parameters of these machine learning algorithms are default, and the results are shown in Table II. It can be seen from the table that RF has the best classification performance and performs well among the six metrics, and can better identify whether the flow table is overflowed by malicious flows.

Offline Malicious Flows Identification: We also conducted offline malicious flows identification before we deployed FTOP to switches. It evaluates that the selected flow rule features and the RF classifier can effectively identify malicious flow rules. We

TABLE III
OFFLINE MALICIOUS FLOW CLASSIFICATION

Model	Accuracy	Precision	Recall	F-score	FPR	MCC
AdaBoost	0.9568	0.9294	0.9986	0.9895	0.0474	0.9424
GB	0.9744	0.9677	0.9747	0.9711	0.0258	0.9481
XGBoost	0.9704	0.9722	0.9620	0.9671	0.0227	0.9403
SVM	0.8135	1.0000	0.7948	0.8857	0.0000	0.5105
KNN	0.8991	0.8026	0.9994	0.8903	0.1978	0.4685
RF	**0.9889**	**0.9921**	**0.9895**	**0.9908**	**0.0118**	**0.9896**

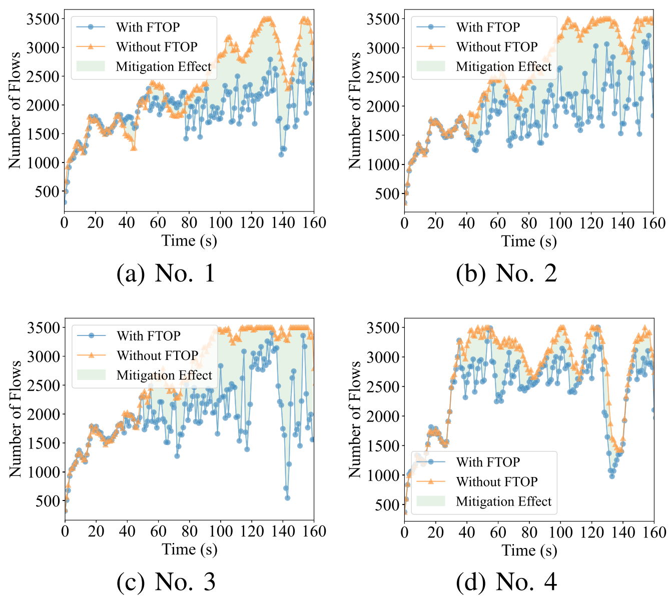


Fig. 10. The number of flow rules under four set of simulations.

compared the RF classifier with five machine learning methods implemented using *scikit-learn*, all parameters are default values, the results are displayed in Table III. It is evident from the table that the RF model outperforms other classification models in offline mitigation evaluation.

Online Mitigation Effect: We evaluated the online mitigation effect of FTOP from three perspectives: the quantity of all the flow entries, controller events, and the proportion of malicious rules.

1) num_{rules}: We recorded the distribution of the quantity of all flow entries in the four simulation groups, both with and without FTOP deployed. The results are shown in Table I and Fig. 10. Since the Detector module is activated only if the estimate produced by the Predictor module exceeds the threshold *TH*, the num_{rules} of the flow table with and without FTOP deployed shows no difference at the early stage of being overflowed. When the predicted value exceeds *TH*, *Detector* is triggered, and *Mitigator* evicts the identified malicious rules to achieve mitigation for LFTO attacks. However, because of the delay of eviction rules, the number of evicted rules is less than the newly added malicious rules, leading to a certain increase of num_{rules}. Although the flow table may still overflow with FTOP deployed, in the two circumstances of LFTO attacks and FTO attacks from FCs, the average num_{rules} is significantly lower than without

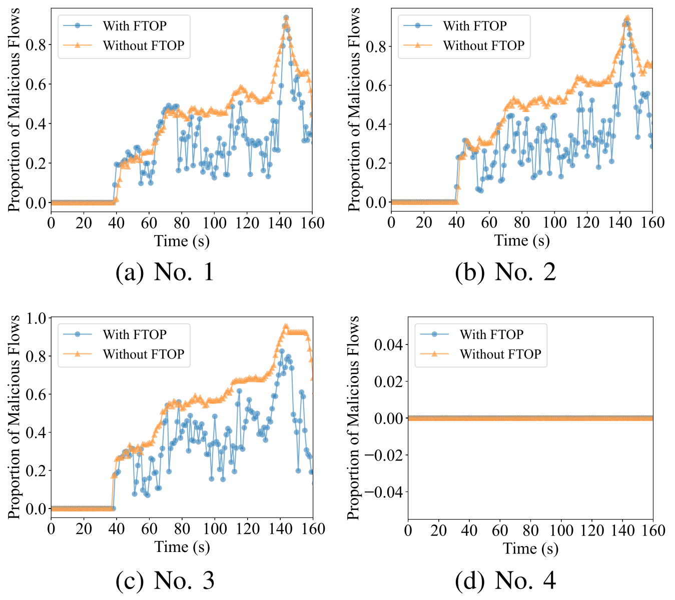
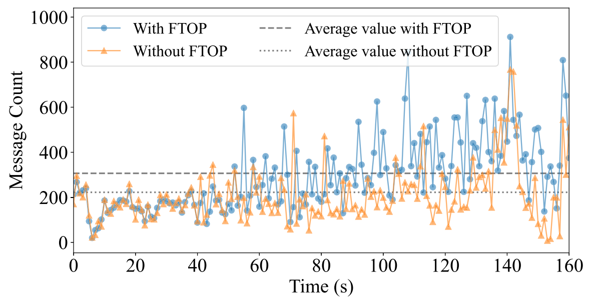


Fig. 11. Proportion of P_m in each simulation group.

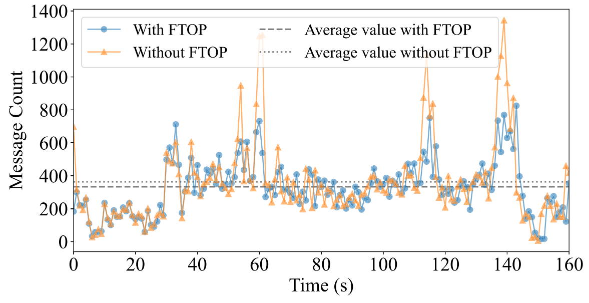
FTOP, indicating that FTOP can effectively alleviate flow table overflow caused by attacks and benign users.

2) *Malicious flows proportion:* The malicious flow entries proportion P_m describes the ratio of malicious rules to the overall flow entries. We compared the P_m with and without FTOP deployed under four sets of simulation parameters and the results are shown in Fig. 11. When LFTO attacks occurred, P_m increases sharply. With FTOP deployed, the rate of evicting rules is lower than that of adding new malicious rules, leading to an increase of P_m, but it is still significantly lower than that without FTOP, which provides strong evidence that FTOP can effectively identify malicious rules, evict the identified rules to reduce the impact of LFTO attacks. Under FCs, P_m is zero, and *Mitigator* cannot identify malicious rules. The mitigation of overflow is completed by the Preventer module.

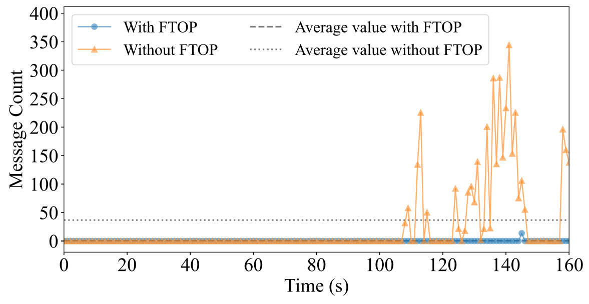
3) *The number of control messages:* Control messages are a good reflection of network status. We use two types of control messages to represent flow table status, which are Packet_In message and Table_Full message. Packet_In messages can well represent the matching of the packets. If the packet does not have a matching flow entry, the switch will generate a Packet_In event and send it to the control plane. The number of Table_Full messages can well reflect the extent of switch's flow table overflow. When the switch is adding flow rules according to the Flow_Mod events from the controllers, if the flow table is overflowed and cannot provide space for new entries, the switch will send Table_Full messages to controllers, indicating that the flow table has no space to install a single flow rule and forward the corresponding packets. We compared the quantity of the two types of control messages mentioned above with and without FTOP deployed under four sets of simulation parameters and the results are shown in Fig. 12. We can see that the quantity of Packet_In messages does not increase significantly with FTOP deployed in the two cases, indicating that the eviction of malicious rules does not affect the matching of legitimate packets



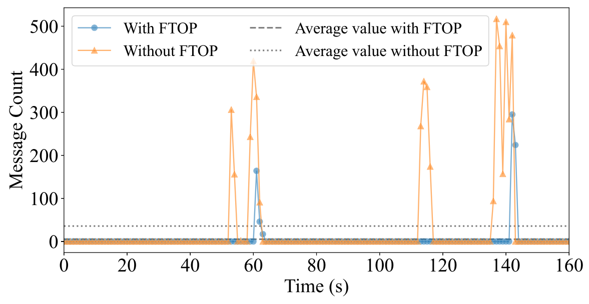
(a) Packet_In under LFTO attacks



(b) Packet_In under FCs



(c) Table_Full under LFTO attacks



(d) Table_Full under FCs

Fig. 12. The number of control messages under LFTO attacks and FCs.

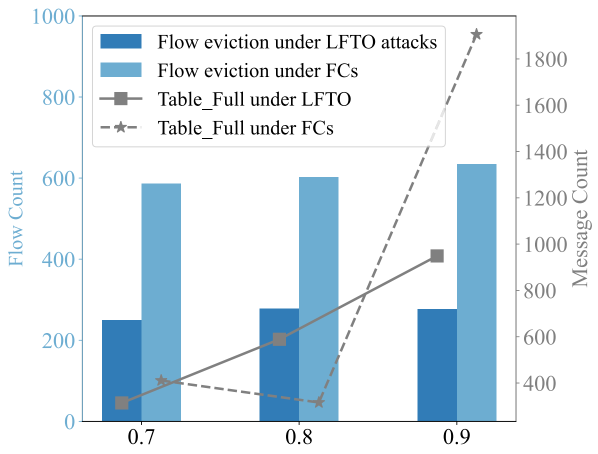


Fig. 13. The comparison of mitigate effect under LFTO attacks with different thresholds.

and the number of Table_Full messages is significantly reduced, it can be seen that FTOP can effectively alleviate the flow table overflow while affecting packets matching as little as possible.

Impact of the threshold: Our system includes one key parameter: α, which adjusts the threshold TH to determines when the system will starts detection and mitigation. We use $counter$ to record the the urgency of mitigation. If the counter is greater than 2, it means that the Detector has determined that the switch is attacked by LFTO twice in a row. Since Mitigator will expel all the attack flow entries which are stored chronically in the switch's flow table every time it's determined that an attack has been detected, two consecutive detection results of 1 indicate that the attack has entered the outbreak stage, and the threshold needs to be lowered for earlier detection.

To know how α influences the performance of FTOP, we set three values for it, which are 0.7, 0.8 and 0.9, and we evaluate the performance from two aspects, which are number of eviction rules and the number of Table_Full message count.

Fig. 13 shows the mitigation effect of the three α values in LFTO attack FCs scenarios. The dark blue bar symbolizes the mean number of rule evictions in the LFTO attack scenario, and the light blue bar symbolizes the mean number of rule evictions in the FCs scenario, where the mean rule eviction refers to the mean number of rules deleted by a single eviction operation. It can be seen that the bigger the α, the bigger the average quantity of evictions. This is because the greater the threshold, the higher the P_m during mitigation, and the more rules need to be evicted, but the overflow is also easier. As the alpha increases, the number of Table_Full messages also increases, which means that the number of overflows has increased. Therefore, we set alpha to 0.7, which can enable detection and mitigation earlier and effectively prevent flow table overflows.

System latency: The latency of FTOP includes four parts, the processing time of *Predictor*, *Detector*, *Mitigator*, and *Preventer*. We recorded the processing time of these four modules with the attack parameters shown in Table I, and the processing time of each module are illustrated in Fig. 14.

Fig. 14(a) displays the processing time of *Predictor* and *Detector*. Fig. 14(b) shows the processing time of *Mitigator* and *Preventer*. *Predictor* can mostly complete the predicting on flow count of the next time stamp in 0.061 s, with a median of 0.053 s. *Detector* can complete attack detection in 0.047 s, with a median of 0.042 s. For flow eviction, *Mitigator* completes mitigation with an average of 0.543 s, and *Preventer* processes with an average time of 1.680 s. The reason why *Mitigator* and *Preventer* takes longer to process is that the *ovs* command needs to be operated once for each flow to be deleted, and after an attack is detected, thousands of flows need to be deleted each time

TABLE IV
RESULTS OF COMPARATIVE EVALUATION UNDER LFTO ATTACKS

Method	Deploy Plane	Average Flow Table Usage (%)	Total Overflow Count (s)	Earliest Overflow Time (s)	Average P_m	Max P_m	Packet_In Count	Table_Full Count
REFUSE [43]	Data	66.54	170	55	59.64	96.30	232427	42150
EVICT [43]	Data	72.48	274	59	52.54	95.55	314696	NA
SIFT [16]	Control	68.17	211	101	37.25	96.43	284876	37510
TableGuard [41]	Control	66.88	217	98	34.63	95.46	242532	80322
SFTOGuard [13]	Data	63.10	**3**	**162**	32.21	93.97	240052	**103**
FTOP	Data	**50.56**	10	106	**20.66**	**83.79**	**187048**	672

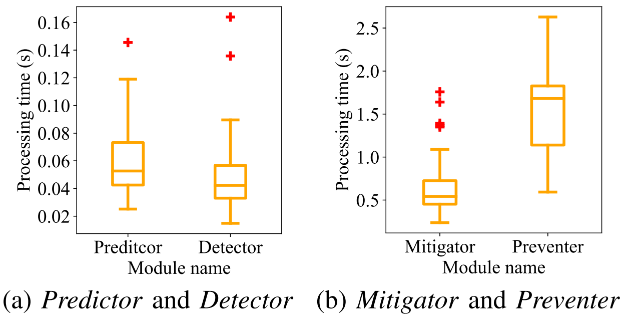


Fig. 14. Processing time of each module of FTOP.

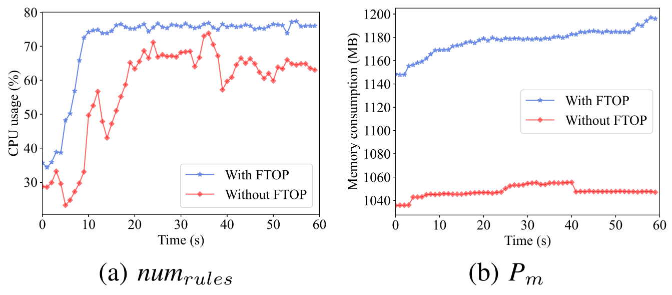


Fig. 15. The comparison of system overhead.

Mitigator is activated. Consequently, the latency of the entire system is between 1 3 seconds, with an average of 2.297 s.

System consumption: We tested the overhead of FTOP, and the results are presented in Fig. 15. With LFTO deployed, the mean CPU usage increased by only approximately 18% within one minute (the remaining CPU consumption in the figure is attributed to network simulation scripts to fill the background traffic and software), and the memory usage was as low as 129 MB. As a result, the system consumption of FTOP is accepctable.

E. Performance Comparison

We compared FTOP with OpenFlow native overflow policies: EVICT and REFUSE, and three relevant methods from the literature.The methods selected for comparative experiments include SIFT [16] (source code at [54]), TableGuard [41] and SFTOGuard [13]. Table IV displays the experimental results under LFTO attacks scenario and Table V displays the results under FCs. The duration of each experiment is 600 s, with a

TABLE V
RESULTS OF COMPARATIVE EVALUATION UNDER FCS

Method	Average Flow Table Usage (%)	Total Overflow Count (s)	Earliest Overflow Time (s)	Packet_In Count	Table_Full Count
REFUSE [43]	77.85	62	22	276623	20994
EVICT [43]	81.60	181	20	247774	NA
SIFT [16]	**62.62**	**17**	41	384894	1371
TableGuard [41]	66.80	85	37	287919	25049
SFTOGuard [13]	83.96	142	37	311063	28243
FTOP	67.18	18	**56**	269538	**411**

table size of 3500. The total overflow count calculates how many times the flow table has been overflowed during each simulation process; the earliest overflow time is the shortest time that is required for the first flow table overflow to occur.

It can be seen from Table IV that FTOP effectively mitigates LFTO attacks, mainly reflected in significantly reducing the quantity of malicious attack flow entries stored in the flow table. Specifically, FTOP reduced the average P_m to 20.66%, outperforming other five methods. This also significantly alleviate the average flow table usage to 50.56%. For control messages, the number of Table_Full messages generated by using SFTOGuard is less than that of FTOP, that is, using SFTOGuard can better prevent overflow, but the proportion of attack flow is higher when using SFTUGuard, indicating that SFTOGuard drives out a major quantity of benign flow entries to prevent overflow, which tremendously affects normal user behaviors.

Table V shows the comparative simulation results under FCs. From the Table, we can see that FTOP can significantly mitigate the overflow caused by benign users. FTOP significantly reduces the flow table occupation to 67.18%. It also has excellent performance in preventing overflow, and the total overflow time is 18 s, which is lower than other methods. In addition, the earliest overflow time of FTOP is 56 s and the number of Table_Full messages is much smaller than other methods, which means FTOP outperforms other methods in mitigating flow table overflow caused by legitimate users.

VII. CONCLUSION

In this article, we propose FTOP to mitigate flow table overflow in both LFTO attack and FCs scenarios. FTOP can significantly protect SDN switches against FTO attacks from attackers and FCs by continuously estimating flow rule count using

Kalman Filtering, detecting LFTO attacks with random forest model, mitigating LFTO attacks by removing malicious flows, removing less important flows to prevent being overflowed by FCs. Our simulation results indicate that FTOP successfully mitigates flow table overflow by 65% compared to OpenFlow's native strategies and 35% compared to three existing schemes.

However, there are still questions that require further investigation. The first question is how to detect and mitigate flow table overflow in large-scale SDN networks with high-levels of traffic and complex network topologies. The second is the coordination and communication problems that exist in distributed mitigation. Switches on a link may have different prediction or detection results. If there is no unified control logic, agreed flow rules may be repeatedly deleted and installed. In the future, we will implement LFTO attacks and FCs in large-scale network to verify the effectiveness of FTOP and design a centralized mitigation control module to process the flow table information of each switch, and provide mitigation strategies for switches that need mitigation.

References

[1] 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, Firstquarter 2015.

[2] P. Saint-Andre, "Extensible messaging and presence protocol (XMPP): Core," 2011. [Online]. Available: http://xmpp.org/rfcs/rfc6120.html

[3] N. McKeown et al., "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[4] W. Li, W. Meng, and L. F. Kwok, "A survey on OpenFlow-based software defined networks: Security challenges and countermeasures," *J. Netw. Comput. Appl.*, vol. 68, pp. 126–139, 2016.

[5] B. Yuan, C. Lin, D. Zou, L. T. Yang, and H. Jin, "Detecting malicious switches for a secure software-defined tactile internet," *ACM Trans. Internet Technol.*, vol. 21, no. 4, pp. 1–23, 2021.

[6] R. Biswas and J. Wu, "Traffic engineering to minimize the number of rules in SDN datacenters," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 2, pp. 1467–1477, Apr.-Jun. 2021.

[7] R. Kandoi and M. Antikainen, "Denial-of-service attacks in OpenFlow SDN networks," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, 2015, pp. 1322–1326.

[8] S. Yu, W. Zhou, W. Jia, S. Guo, Y. Xiang, and F. Tang, "Discriminating DDoS attacks from flash crowds using flow correlation coefficient," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 6, pp. 1073–1080, Jun. 2012.

[9] Y. Zhou et al., "Raze policy conflicts in SDN," *J. Netw. Comput. Appl.*, vol. 199, 2022, Art. no. 103307.

[10] B. Isyaku, M. S. Mohd Zahid, M. Bte Kamat, K. Abu Bakar, and F. A. Ghaleb, "Software defined networking flow table management of OpenFlow switches performance and security challenges: A survey," *Future Internet*, vol. 12, no. 9, 2020, Art. no. 147.

[11] 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, 2018.

[12] F. Haq, A. Naaz, T. P. K. Bantupalli, and K. Kataoka, "DRL-FTO: Dynamic flow rule timeout optimization in SDN using deep reinforcement learning," in *Proc. Asian Internet Eng. Conf.*, 2021, pp. 41–48.

[13] D. Tang, D. Zhang, Z. Qin, Q. Yang, and S. Xiao, "SFTO-guard: Real-time detection and mitigation system for slow-rate flow table overflow attacks," *J. Netw. Comput. Appl.*, vol. 213, 2023, Art. no. 103597.

[14] R. Al-quraan and A. Alma'aitah, "A secure switch migration scheduling based on prediction for load balancing in SDN," in *Proc. IEEE 12th Int. Conf. Inf. Commun. Syst.*, 2021, pp. 364–370.

[15] J. Xu, L. Wang, C. Song, and Z. Xu, "Proactive mitigation to table-overflow in software-defined networking," in *Proc. IEEE Symp. Comput. Commun.*, 2018, pp. 00719–00725.

[16] T. A. Pascoal, Y. G. Dantas, I. E. Fonseca, and V. Nigam, "Slow TCAM exhaustion DDoS attack," in *Proc. 32nd IFIP Int. Conf. ICT Syst. Secur. Privacy Protection*, 2017, pp. 17–31.

[17] S. Gao, Z. Li, B. Xiao, and G. Wei, "Security threats in the data plane of software-defined networks," *IEEE Netw.*, vol. 32, no. 4, pp. 108–113, Jul./Aug. 2018.

[18] M. Rahouti, K. Xiong, N. Ghani, and F. Shaikh, "SYNGuard: Dynamic threshold-based SYN flood attack detection and mitigation in software-defined networks," *IET Netw.*, vol. 10, no. 2, pp. 76–87, 2021.

[19] Y. Qian, W. You, and K. Qian, "OpenFlow flow table overflow attacks and countermeasures," in *Proc. IEEE Eur. Conf. Netw. Commun.*, 2016, pp. 205–209.

[20] 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.

[21] D. Tang, S. Zhang, Y. Yan, J. Chen, and Z. Qin, "Real-time detection and mitigation of LDoS attacks in the SDN using the HGB-FP algorithm," *IEEE Trans. Serv. Comput.*, vol. 15, no. 6, pp. 3471–3484, Nov./Dec. 2022.

[22] D. Tang, S. Wang, B. Liu, W. Jin, and J. Zhang, "GASF-IPP: Detection and mitigation of LDoS attack in SDN," *IEEE Trans. Serv. Comput.*, early access, Apr. 13, 2023, doi: 10.1109/TSC.2023.3266757.

[23] Y. Yu, L. Guo, Y. Liu, J. Zheng, and Y. Zong, "An efficient SDN-based DDoS attack detection and rapid response platform in vehicular networks," *IEEE Access*, vol. 6, pp. 44570–44579, 2018.

[24] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *Proc. Netw. Distrib. Syst. Secur.*, 2015, pp. 8–11.

[25] F. Shoaib, Y.-W. Chow, and E. Vlahu-Gjorgievska, "Preventing timing side-channel attacks in software-defined networks," in *Proc. IEEE Asia-Pacific Conf. Comput. Sci. Data Eng.*, 2021, pp. 1–6.

[26] S. Liu, M. K. Reiter, and V. Sekar, "Flow reconnaissance via timing attacks on SDN switches," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 196–206.

[27] N. Saha, S. Misra, and S. Bera, "Q-Flag: QoS-aware flow-rule aggregation in software-defined IoT networks," *IEEE Internet Things J.*, vol. 9, no. 7, pp. 4899–4906, Apr. 2022.

[28] D. Tang, Y. Yan, C. Gao, W. Liang, and W. Jin, "LtRFT: Mitigate the low-rate data plane DDoS attack with learning-to-rank enabled flow tables," *IEEE Trans. Inf. Forensics Secu.*, vol. 18, pp. 3143–3157, 2023.

[29] S. K. Noh, M. Kang, and M. Park, "Protection against flow table overflow attack in software defined networks," in *Proc. IEEE Int. Conf. Inf. Netw.*, 2021, pp. 486–490.

[30] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *Proc. IEEE Conf. Comput. Commun.*, 2013, pp. 545–549.

[31] A. Panda, S. S. Samal, A. K. Turuk, A. Panda, and V. C. Venkatesh, "Dynamic hard timeout based flow table management in OpenFlow enabled SDN," in *Proc. IEEE Int. Conf. Vis. Towards Emerg. Trends Commun. Netw.*, 2019, pp. 1–6.

[32] N. Aljeri and A. Boukerche, "An efficient heuristic switch migration scheme for software-defined vehicular networks," *J. Parallel Distrib. Comput.*, vol. 164, pp. 96–105, 2022.

[33] B. Yuan, D. Zou, S. Yu, H. Jin, W. Qiang, and J. Shen, "Defending against flow table overloading attack in software-defined networks," *IEEE Trans. Serv. Comput.*, vol. 12, no. 2, pp. 231–246, Mar./Apr. 2019.

[34] E.-D. Kim, Y. Choi, S.-I. Lee, and H. J. Kim, "Enhanced flow table management scheme with an LRU-based caching algorithm for SDN," *IEEE Access*, vol. 5, pp. 25555–25564, 2017.

[35] F. Stajano et al., "Controlling your neighbour's bandwidth for fun and for profit (transcript of discussion)," in *Proc. 25th Int. Workshop Secur. Protoc. XXV*, 2017, pp. 224–231.

[36] B. Isyaku, K. B. A. Bakar, F. A. Ghaleb, and S. Sulaiman, "Performance evaluation of flowtable eviction mechanisms for software defined networks considering traffic flows variabilities," in *Proc. IEEE 12th Symp. Comput. Appl. Ind. Electron.*, 2022, pp. 71–75.

[37] B.-S. Lee, R. Kanagavelu, and K. M. M. Aung, "An efficient flow cache algorithm with improved fairness in software-defined data center networks," in *Proc. IEEE 2nd Int. Conf. Cloud Netw.*, 2013, pp. 18–24.

[38] G. Huang and H. Y. Youn, "Proactive eviction of flow entry for SDN based on hidden Markov model," *Front. Comput. Sci.*, vol. 14, no. 4, pp. 1–10, 2020.

[39] M. Zhang, J. Bi, J. Bai, Z. Dong, Y. Li, and Z. Li, "FTGuard: A priority-aware strategy against the flow table overflow attack in SDN," in *Proc. SIGCOMM Posters Demos*, 2017, pp. 141–143.

[40] H. Luo, W. Li, Y. Qian, and L. Dou, "Mitigating SDN flow table overflow," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf.*, 2018, pp. 821–822.
[41] D. Kong, C. Wu, Y. Shen, X. Chen, H. Liu, and D. Zhang, "TableGuard: A novel security mechanism against flow table overflow attacks in SDN," in *Proc. IEEE Glob. Commun. Conf.*, 2022, pp. 4167–4172.
[42] R. Challa, Y. Lee, and H. Choo, "Intelligent eviction strategy for efficient flow table management in OpenFlow switches," in *Proc. IEEE NetSoft Conf. Workshops*, 2016, pp. 312–318.
[43] "Open vswitch," 2022. [Online]. Available: https://www.openvswitch.org/support/dist-docs/ovs-ofctl.8.html
[44] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Trans. ASME- J. Basic Eng.*, vol. 82, pp. 35–45, 1960.
[45] W. Liang, J. Long, K.-C. Li, J. Xu, N. Ma, and X. Lei, "A fast defogging image recognition algorithm based on bilateral hybrid filtering," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 17, no. 2, pp. 1–16, 2021.
[46] A. S. L. V. Tummala and R. K. Inapakurthi, "A two-stage kalman filter for cyber-attack detection in automatic generation control system," *J. Modern Power Syst. Clean Energy*, vol. 10, no. 1, pp. 50–59, Jan. 2022.
[47] X. Li et al., "Online internet anomaly detection with high accuracy: A fast tensor factorization solution," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1900–1908.
[48] "pykalman," 2012. [Online]. Available: https://pykalman.github.io/
[49] "Mininet," 2022. [Online]. Available: http://mininet.org/
[50] "Ryu controller," 2011. [Online]. Available: https://github.com/faucetsdn/ryu/
[51] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, 2010, pp. 267–280.
[52] "Data set for IMC 2010 data center measurement," 2010. [Online]. Available: http://pages.cs.wisc.edu/tbenson/IMC10_Data.html
[53] "Tcpreplay," 2014. [Online]. Available: https://github.com/appneta/tcpreplay/
[54] "sift defense code," 2017. [Online]. Available: https://github.com/tuliopascoal/sift-defense

Dan Tang received the B.S., M.S., and Ph.D. degrees from the Huazhong University of Science and Technology, Wuhan, China. He is currently an Associate Professor with Hunan University, Changsha, China. His research interests include network security, denial-of-service attack detection and prevention, intrusion detection, and prevention system, with a focus on developing innovative solutions to address the challenges facing these fields.

Zhiqing Zheng received the B.S. degree in information security from Hunan University, Changsha, China, in 2021. She is currently a Postgraduate Student majoring in computer science and technology with Hunan University and her supervisor is Dr. Dan Tang. Her research interests include SDN data plane security, cyber-attack detection, and Internet of Vehicles. She is currently working on online system for detecting and mitigating low-rate denial-of-service attacks under SDN environment.

Keqin Li (Fellow, IEEE) is currently a SUNY Distinguished Professor with the State University of New York, Albany, NY, USA, and National Distinguished Professor with Hunan University, Changsha, China. His research focuses on advanced computing systems. He is actively involved in research related to Big Data computing, energy-efficient computing and communication, CPU-GPU hybrid and cooperative computing, heterogeneous computing systems, high-performance computing, computer architectures and systems, computer networking, and machine learning. Through his innovative research, he is dedicated to advancing the field of computer science and contributing to the development of cutting-edge technologies.

Chao Yin received the bachelor's, master's, and doctoral degrees from the Huazhong University of Science and Technology, Wuhan, China. He is currently an Associate Professor with Jiujiang University, Jiujiang, China. His research interests include information security, Big Data analysis, cloud storage, and erasure codes. With his expertise in these areas, Dr. Yin is committed to advancing the field of computer science and contributing to the development of novel solutions to address the challenges in these fields.

Wei Liang received the Doctor of Philosophy degree from Hunan University, Changsha, China. From 2014 to 2016, he was a Postdoctoral Scholar with Lehigh University, Bethlehem, PA, USA. He is currently a Professor with the Hunan University of Science and Technology, Xiangtan, China. His research interests include embedded system protection, blockchain security, wireless sensor networks, and network security protection.

Jiliang Zhang (Senior Member, IEEE) is currently a Full Professor with the College of Integrated Circuits, Hunan University, Changsha, China. He is the Vice Dean of the College of Integrated Circuits, Hunan University and the Secretary-General of CCF Fault-Tolerant Computing Professional Committee. He has authored more than 80 technical papers in leading journals and conferences. His research interests include hardware security, integrated circuit design, and intelligent system. He was the recipient of the CCF Integrated Circuit Early Career Award and CCF Distinguished Speaker, and the winner of Excellent Youth Fund of the NSFC. He is the Program Committee Member for a number of well-known conferences such as DAC, ASP-DAC, GLSVLSI, and FPT.