The DUDFTO Attack: Towards Down-to-UP Timeout Probing and Dynamically Flow Table Overflowing in SDN

Jiasong Li, Yunhe Cui, Yi Chen, *Member, IEEE*, Guowei Shen, Chun Guo, and Qing Qian, *Member, IEEE*

***Abstract*—As a new network structure, the decoupling of the control plane and forwarding plane makes Software-Defined Networking (SDN) widely used in large-scale network scenarios. However, the decoupling network architecture also brings new vulnerabilities. The flow table overflow attack is an attack strategy that can overwhelm SDN switches. Nevertheless, the existing flow table overflow attacks may fail in probing timeouts and match fields of flow entries, due to link failure, measurement of the round-trip time (RTT) of different packets, interference of hard-timeout and idle-timeout. Meanwhile, the stealthiness of the existing attacks may also reduce, as these attacks use fixed attack rate. To improve the timeout probing accuracy and the stealthiness of attack, a new flow table overflow attack strategy, DUDFTO, is proposed to accurately probe timeout settings and match fields, then stealthily overflow SDN flow tables. Firstly, it probes the match fields by measuring the one-sided transmission delay of the packets. After that, DUDFTO designs a down-to-up feedback-based timeout probing algorithm to eliminate the issues caused by high RTT, link failure, interference between hard-timeout and idle-timeout. Then, DUDFTO designs a dynamic attack packets sending algorithm to improve its stealthiness. Finally, DUDFTO probes the flow table state to stop sending new attack packets. The evaluation results demonstrate that DUDFTO outperforms the existing attacks in terms of match fields probing ability, timeout probing relative error, number of packet_in and flow_mod messages generated by the attack, rate distribution of packet_in and flow_mod messages generated during the attack, and number of detected attack packets.**

***Index Terms*—SDN switchs, flow table overflow attack, information probing.**

Received 21 August 2024; revised 21 March 2025; accepted 23 May 2025. Date of publication 27 May 2025; date of current version 7 October 2025. This work was supported by the National Natural Science Foundation of China (NSFC) under the Grant Nos. 62462010 and 62102111, the Guizhou Provincial Science and Technology Plan, China under the Grant No. Qian Ke He Zhongda Zhuanxiang Zi[2024]003 and [2024]014, the Science and Technology Support Program of Guizhou Province, China under Grant No. [2022]071, and the Big Data Security and Network Security Innovation Team of Guizhou Provincial High Education Institution, China under Grant No. [2023]052. The associate editor coordinating the review of this article and approving it for publication was J.-F. Botero. *(Corresponding author: Yunhe Cui.)*

Jiasong Li, Yunhe Cui, Yi Chen, Guowei Shen, and Chun Guo are with the State Key Laboratory of Public Big Data, the Engineering Research Center of Text Computing & Cognitive Intelligence, Ministry of Education, the College of Computer Science and Technology, and the Guizhou Provincial Characteristic Key Laboratory of Software Engineering and Information Security, Guizhou University, Guiyang 550000, China (e-mail: lijiasong1213@163.com; yhcui@gzu.edu.cn; yichen.research@gmail.com; gwshen@gzu.edu.cn; cguo@gzu.edu.cn).

Qing Qian is with the School of Information, Guizhou University of Finance and Economics, Guiyang 550000, China (e-mail: qqian2018_p@163.com).

Digital Object Identifier 10.1109/TNSM.2025.3574260

I. Introduction

As a new network structure, SDN is increasingly used in large-scale network scenarios, such as data centers, Internet of Things, and cloud networks [1]. Although the decoupling of the control plane and forwarding plane brings SDN network programmability, it also introduces new security challenges to SDN. When processing packets, the SDN switch will initially check if there is a flow entry which can match these packets. If there is a corresponding flow entry, the switch will forward, edit or drop these packets. Otherwise, the switch will send a packet_in message to the controller to request instructions on how to process these packets. Upon receiving and processing the packet_in message, the controller will send a flow_mod message to the switch. Subsequently, the switch will install a new flow entry to handle these packets. However, the SDN switches have limited capacity to store flow entries [2], [3]. The attacker can exploit this vulnerability to launch flow table overflow attack targeted at the SDN switches, by continuously sending attack packets and installing new flow entries. As a result, the flow table storage resource will be overwhelmed by these malicious flow entries, making the SDN switches cannot process normal packets.

SDN has designed a flow entry expiry mechanism to remove flow entries by the switches, which consists of hard-timeout and idle-timeout [4]. The hard-timeout indicates that the flow entry will be deleted if its existence time exceeds the preset hard-timeout. The idle-timeout indicates that the flow entry will be deleted if the flow entry does not match any packets in the given idle-timeout period. The hard timeout and idle-timeout can be used separately or simultaneously.

There are some related attack strategies that can probe the information of SDN and overflow flow table [13], [15], [16], [17], [18]. However, some of these studies face with the following issues.

- *ISSUE 1: High RTT, link failure, interference between hard-timeout and idle-timeout may lead to low probing accuracy of some attacks.* For instance, Ahmed et al. first probe a timeout value (hard-timeout or idle-timeout), then probe another timeout value based on the probed timeout value [14]. But they do not consider the high Round-Trip Time (RTT) caused by link delay. Cao et al. probe hard-timeout first to eliminate the interference

between hard-timeout and idle-timeout. But they also do not consider the high RTT [17]. As a result, the timeout probing accuracy of these two methods will be affected by high RTT. Yiğit et al. first probe the hard-timeout [15]. But when probing the idle-timeout, it seems that they do not add RTT when calculating the total sleep time. Hence, in some cases, their method will consider the flow entry deletion caused by the hard-timeout as flow entry deletion caused by the idle-timeout. Therefore, the probed idle-timeout may be significantly smaller than the true idle-timeout in high RTT environments. Moreover, none of the above methods consider the case where the RTT of the probing packet increases or becomes unmeasurable due to link failure. Consequently, they may mistakenly attribute such RTT increases to idle or hard timeout, thereby rendering their probed results inaccurate or entirely incorrect.

- *ISSUE 2: Some existing attacks may not successfully probe all the fields when probing match fields.* The reason is that these attacks infer the match fields by measuring the RTT of different packets [14], [17]. But they may fail to infer some special fields by measuring the RTT of the probing packets. For instance, whether the source IPv4 address is contained in the match fields cannot be probed. Because when the attackers forge the source IPv4 address of the probing packets, the receiver cannot find the sender due to it does not have the ARP mapping of the forged source IPv4 address. Therefore, the attacks cannot measure the RTT of these packets.
- *ISSUE 3: The fixed attack rate reduces the stealthiness of the existing attacks.* When attacking the switch, the existing attacks usually send the flow table overflow packets at a fixed rate [17], [19]. However, the fixed attack rate causes its traffic to lose its volatility and present a periodicity characteristic. Hence, these attacks maybe easily found out by the attack detection methods [5], [6], [7], [8], [28], [29], [30], [31], [32].

To overcome the above issues, this paper proposes the DUDFTO attack, a **D**own-to-**U**p timeout probing and **D**ynamic **F**low **T**able **O**verflowing attack. Here, it should be noted that DUDFTO is designed for SDN used out-band communication mode. In summary, this work makes the following contributions:

- This paper proposes DUDFTO, a down-to-up timeout probing and dynamic flow table overflow attack. DUDFTO contains the following four phases: probing match fields, probing timeouts, launching flow table overflow attack, and probing flow table state.
- To overcome issue 1, DUDFTO designs a down-to-up feedback-based timeout probing algorithm that can probe the timeout settings more accurately.
- To address issue 2, unlike the existing attacks, DUDFTO infers the match fields by measuring the one-sided transmission delay of packets.
- To overcome issue 3, DUDFTO designs a dynamic attack packets sending algorithm to improve its stealthiness.
- DUDFTO is evaluated and compared with LOFT [17] and RAFA [15]. The evaluation results show that DUDFTO outperforms LOFT in terms of match fields probing ability, and number of detected attack packets. Meanwhile, the timeout probing relative error comparison in challenging network environment also shows that DUDFTO outperforms LOFT and RAFA.

II. Related Work

Shin et al. used fingerprint technology to identify network categories [9]. Unlike that work, our work focuses on probing the matching fields and timeout settings of the flow entry and launching a dynamic attack to improve its stealthiness. In [10] and [11], Sonchack et al. proposed a probing attack to infer match fields. The attack simultaneously sends timing probing packets and test probing packets to the target network. The timing probing packets probe the load of the controller by measuring RTT, and the test probing packets probe the match fields by sending packets with different head fields. If sending a test probing packet causes the RTT of the timing probing packets increase, the attack infers that the controller is processing the test probing packet. By sending test probing packets with different head fields, the attack can infer the match fields. Lin et al. also studied how to probe match fields [12]. Unlike [10] and [11], their work consists of three steps: probing, clustering and rule inference. After sending different packets and measuring their delays, they use packet delays and k-means method to divide packets into multiple classes. Then a prior algorithm is utilized to find common fields in the classes to infer the match fields.

Azzouni et al. probe which kind of SDN controller is managing the network by probing the timeout settings [13]. However, they probe timeout settings under the default timeout settings, so their probing will not be successful when controller changes the timeout setting. Meanwhile, the combined existence of idle-timeout and hard-timeout is not considered. In addition, their timeout probing algorithm does not consider the interference between hard-timeout and idle-timeout. Ahmed et al. probes match fields by measuring RTT, but it does not consider that RTT may not be measured when the source IPv4 or IPv6 address is forged and the receiver does not have the corresponding ARP mapping [14]. Moreover, the accuracy of their timeout probing algorithm may be low in high RTT environments. Yiğit et al. studied how to infer network types and timeout settings [15]. They employ statistical methods to calculate the average and standard deviation of RTT when probing timeout values. Then they use the mean value plus multiple standard deviations as the threshold to judge whether a flow entry is installed in the switch. Additionally, the work analyzes the probing accuracy under different network conditions. However, this work first probes hard-timeout and then probes idle-timeout. When probing the idle-timeout, it does not consider adding RTT when calculating the total sleep time. Therefore, the probed idle-timeout may be significantly smaller than the true idle-timeout when the RTT is high. Zhou et al. studied how to infer the flow table storage capacity and different replacement algorithms (LRU, FIFO) [16]. The inference algorithm has two parts, flow table state detection and flow table state control. They probe the

TABLE I
MAIN DIFFERENCES BETWEEN DUDFTO AND THE EXISTING METHODS

Ref.	Technique	Comments
[9]	T-test, Time inference	Fingerprint attack is carried out by measuring RTT.
[10], [11]	Periodic probing, Test streams	The match fields are inferred by periodically sending probing packets and test streams.
[12]	K-means, Prior algorithm	The match fields are probed using K-means and prior algorithm. But it only probes the IP, protocol and port fields.
[13]	Time-Analysis, Packet-Analysis	The controller type is probed by inferring the default timeout settings. The combined existence of idle-timeout and hard-timeout is not considered.
[14]	Statistical methods, Quantitative analysis	The hard-timeout, idle-timeout, match field, controller reaction at table full event and information about the topology are probed. The RTT of a forge packet without a corresponding ARP mapping may not be measured when probing matching field.
[15]	Statistical methods, Quantitative analysis	The network type and timeout setting are probed. When probing the timeout setting, it first probes the hard-timeout and then probes the idle-timeout. The probed idle-timeout may be significantly smaller than the true idle-timeout when the RTT is large.
[16]	Reconnaissance, Inference algorithms	It infers the table capacity and flow entry replacement algorithms.
[17]	Quantitative analysis, Binary search	It launches attack based on inferred information. The RTT of a forge packet without a corresponding ARP mapping may not be measured when probing a matching field. When probing the timeout settings, the probing accuracy may be low. Meanwhile, its attack rate is constant.
[18]	Finite-state machine, Adaptive timeout strategy	It infers dynamic timeout settings by using finite-state machine and launches attack based on the inferred information. But only dynamic and fix idle-timeout settings can be probed.
[19]	Reconnaissance, Inference algorithms	It infers the internal parameters of the flow table (size, policy, and load) by using the inference algorithm and launches attack based on the inferred information. But the rate of attack is constant.
[20]	Timing probes, Test streams	Same as [10] and [11], it infers match fields by sending timing probes and test streams simultaneously. Meanwhile, it launches attack using a constant rate.
DUDFTO	Feedback mechanism, Uniform distribution, One-sided transmission delay	DUDFTO probes match fields based on the one-sided transmission delay. Meanwhile, it uses a down-to-up feedback mechanism to improve the timeout probing accuracy. It also utilizes a dynamical attack packets sending algorithm to improve its stealthiness.

different states of flow table and flow entries by flow table state detection, and manipulate the state of flow entries by flow table state control.

Cao et al. proposed LOFT, a flow table overflow attack that uses the minimum attack packet rate to overwhelm the SDN switches [17]. LOFT first probes match fields and timeout settings of flow entries. Then it computes the minimum attack rate and launches the flow table overflow attack. Similar to [14], they also do not consider the scenario that there is no corresponding ARP mapping in the destination host when probing matching fields. Additionally, their timeout probing accuracy may be low in high RTT environments. Moreover, they launch an attack at a fixed rate, which may decrease the stealthiness of the attack. Shen et al. proposed an advanced flow table overflow attack [18]. Unlike other attacks, this attack is able to probe dynamic timeout settings. However, it can only probe dynamic and fixed idle-timeout settings. Meanwhile, they also launch the attack at a fixed rate. Yu et al. developed an intelligent attack model that uses the specific cache-like behavior of a flow table to infer its internal configuration and state [19]. Their work is similar to [16], it can also probe the flow table storage capacity and flow entry replacement method. They designed the attack for different replacement methods, but their attack rate was also fixed. Zhang et al. proposed a control plane reflection attack [20]. It exploits direct and indirect data plane events to force the control plane to send massive expensive control messages to the SDN switch. Their attack is launched using the minimum fixed attack rate. Pascoal et al. studied the Slow-TCAM and Slow Saturation attacks [21]. Their experiments prove that these attacks can cause great damage to SDN.

Table I lists the main differences between DUDFTO and the existing attack methods. Compared with the existing flow table overflow attacks, DUDFTO designs a down-to-up feedback-based timeout probing algorithm, addressing the issue of inaccurate probing caused by the interference between the hard-timeout and idle-timeout. Meanwhile, DUDFTO infers the match fields by measuring the one-sided transmission delay of packets, to address the issue of failing to probe all match fields of the existing attacks. Finally, DUDFTO designs a dynamical attack packets sending algorithm to improve its stealthiness.

III. DESIGN OF DUDFTO

In this section, we elaborate DUDFTO, a down-to-up timeout probing and dynamic flow table overflow attack method. DUDFTO contains four phases, including 1) probing match fields, 2) probing timeouts, 3) launching flow table overflow attack, and 4) probing flow table state, as shown in Fig. 1. The following is a detailed description of its phases. Table II explains the meaning of the symbols used in this paper.

A. Probing Match Fields

To overflow the flow table storage space, DUDFTO needs to install sufficient flow entries on the switch and make the flow entries exist for a long time. Therefore, DUDFTO has to first probe which match fields can be used to trigger new flow entry installation. There are 40 kinds of match field available in OpenFlow1.3. The controller can arbitrarily choose some of them to generate the match fields of a flow entry. Hence, the attacker can find out how the match fields of the flow entry are

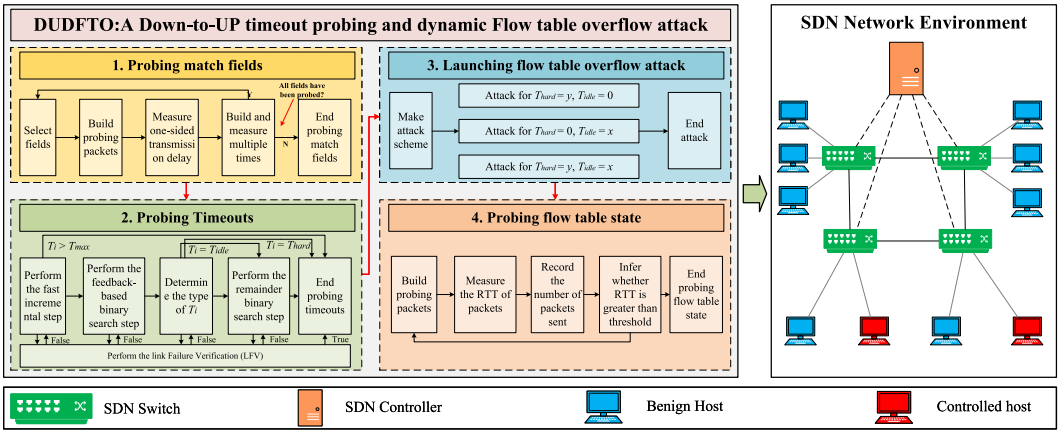


Fig. 1. Overview of DUDFTO.

TABLE II
SYMBOL DEFINITION

Symbol	Explanation
T_i	The time interval between the i-1th probing packet and the ith probing packet
T_k	The true timeout value (idle-timeout or hard-timeout)
T_{max}	The maximum hard-timeout
T_{idle}	The probed idle-timeout
T_{hard}	The probed hard-timeout
T_{over}	The upper bound value of the probed timeout
λ	The minimum incremental of the probing interval
δ	The maximum number of cycles
η	The RTT threshold used to determine whether the flow entry is deleted
α	The number of incremental reductions
β	The maximum value of the number of incremental reductions
σ	The final probed hard-timeout value
D	The measured RTT
χ	The coefficient that denotes how many times V_{upper} surpassed than V_{lower}

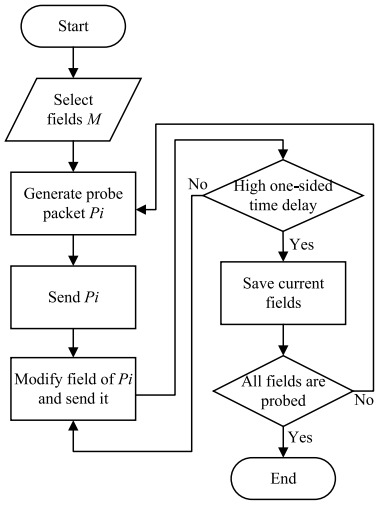


Fig. 2. The illustration of probing Match Fields.

formed after probing all the fields. After probing the match fields, the attacker can generate attack packets by changing the match fields that can be forged. For example, when the attacker probes that the match fields contains $ipv4\_src$ field, he can construct attack packets to install a large number of malicious flow entries in flow table by spoofing the source IP address of these packets. DUDFTO probes the match fields (such as source IP, source TCP port, etc.) used in the flow entries of a switch. More importantly, these probed match fields are used to induce the switch to install a large number of useless flow entries. This can be done by changing the value of a match field. Meanwhile, DUDFTO does not specify that a match field must be used to launch a flow table overflow attack. Therefore, DUDFTO does not consider the scenario described in [34], where SDN controller remembers that a particular IP address is attached to which port of the switch.

For this purpose, DUDFTO constructs match field probing packets and sends these packets to the SDN switch. By measuring the one-sided transmission delay of these match field probing packets and changing the header of these match field probing packets, DUDFTO can obtain the match fields used in the flow entries.

Before probing the match fields, DUDFTO has calculated the one-sided transmission delay threshold which is used to determine if a new flow entry is installed. Similar to previous works [14], [15], DUDFTO first sends a packet to the network. Then, it continuously sends the same packets and measures the one-sided transmission delays. After that, the average value and standard deviation of these one-sided transmission delays are calculated. Finally, the average value plus standard deviation is used as the threshold to judge whether the one-sided transmission delay is high. The above process is also adopted in calculating the RTT threshold.

As shown in Fig. 2, assuming DUDFTO controls two hosts in SDN, DUDFTO first builds a set of probing packets. Subsequently, it sends the probing packets from one controlled host to another. Then, the one-sided transmission delay of the probing packets is measured at another host. If the one-sided transmission delay of the first probing packet is significantly greater than the one-sided transmission delay of the subsequent packets, it indicates that a new flow entry is installed.

The basis principle of that process is that when a traffic flow does not match any flow entries, the switch will spend more time on handling its first packet. Specially, when the

first packet does not match any flow entry in the switch, the switch will send a packet_in message to the controller to ask how to handle the packet. After receiving and processing packet_in message, the controller will send packet_out and flow_mod messages to instruct the switch to install flow entries to process that packet and the subsequent packets of that traffic flow. Therefore, if the one-sided transmission delay of the first probing packet is significantly higher than the subsequent probing packets, it indicates that a new flow entry has been installed. Therefore, DUDFTO can infer that the field is contained in the matching fields.

After probing a match field, DUDFTO generates a new set of probing packets by adding, deleting, and modifying a certain field of the last probing packets. Repeating this process, DUDFTO can infer all the fields of the match fields.

B. Probing Timeouts

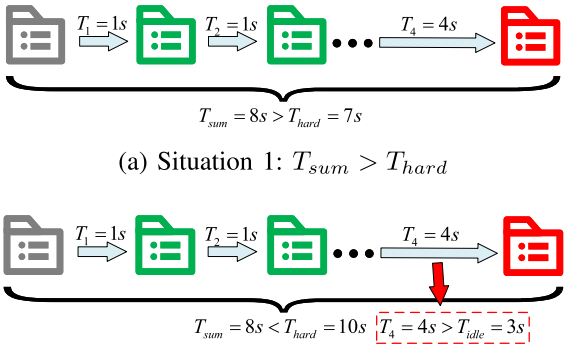
After probing the match fields, DUDFTO continues to probe the timeout settings to ensure that the installed malicious flow entries exist for a long time.

DUDFTO proposes a down-to-up feedback-based timeout probing algorithm. The algorithm is divided into three steps, including 1) **fast increment step**, 2) **feedback-based binary search step** and 3) **remainder binary search step**. In the fast increment step, DUDFTO roughly infers the range of timeout value by rapidly increasing the probing time interval. In the feedback-based binary search step, DUDFTO uses the binary search and feedback mechanism to precisely infer the timeout value. Meanwhile, in this step, DUDFTO also designs a method to judge whether the probed timeout value is hard-timeout or idle-timeout. When the timeout is hard-timeout, the probing timeouts phase ends. When the timeout is idle-timeout, the remainder binary search step utilizes the probed idle-timeout value with binary search to find the hard-timeout value quickly. In addition, we design a link failure verification method named LFV to infer the cause of the increased or unmeasurable RTT, further mitigating the interference of link failures on timeout probing accuracy.

1) Fast Increment Step: The initial sending interval of probing packet is recorded as T_1. The probing time of the i round is T_i. T_k is the timeout value (hard-timeout or idle-timeout). When T_i does not exceed T_k or $\sum_{i=1} T_i$ does not exceed the hard-timeout value, the time interval T_{i+1} for the next probe is the sum of the time intervals of all previous probes. Therefore, the time interval T_i is calculated as:

$$T_i = \begin{cases} \acute{\eta}, & \text{if } i = 1 \quad or \quad 2 \\ \sum_{i=1} T_{i-1}, & \text{if } i >= 3 \end{cases} \tag{1}$$

According to the existing timeout settings [22], [23], [24], [25], [26], [27], the maximum hard-timeout value is set as T_{max}. If the time interval T_i is larger than T_{max}, it means that the switch dose not set hard-timeout or idle-timeout. Hence, DUDFTO finishes the timeout probing. On the contrary, if a significant increase in the measured RTT occurs, it indicates that the switch has set idle-timeout or hard-timeout. In this case, there are probably two reasons for the significant increase in the RTT. The first reason is that T_i exceeds the true timeout value T_k, which is shown as Eq. (2). Another reason may be that the sum of the time spent T_{sum} from round *1* to round *i* exceeds the hard-timeout value T_N, which is shown as Eq. (3). Next, DUDFTO tries to find the exact reason of the significant increase in the measured RTT. It sends two probing packets (p_1 and p_2) with the time interval T_i and measures the RTT.



(a) Situation 1: $T_{sum} > T_{hard}$

(b) Situation 2: $T_i > T_{idle}$

Fig. 3. Two situations that occur in the fast increment step.

$$T_i > T_k \tag{2}$$

$$T_N < \sum_{j=1}^{N} T_j \tag{3}$$

$$T_{over} = \begin{cases} T_i, & \text{if } T_i > T_k \\ T_{i+1}, & \text{if } T_N < \sum_{j=1}^{N} T_j \end{cases} \tag{4}$$

$$\tau = \begin{cases} T_{i-1}, & \text{if } T_i > T_k \\ T_i, & \text{if } T_N < \sum_{j=1}^{N} T_j \end{cases} \tag{5}$$

If the measured RTT of p_2 is small, it means that T_i has not exceeded T_k. So the reason for the increase in RTT is that the sum of the time spent from round 1 to round *i* exceeds the hard-timeout value, as shown in Eq. (3). In this case, it can be concluded that T_{i+1} is greater than T_N. So at this time, DUDFTO assigns T_{over} as T_{i+1}, as shown in Eq. (4). And it assigns the increment τ as T_i, as shown in Eq. (5). Meanwhile, DUDFTO no longer increases the probing time interval according to Eq. (1). So far, the fast increment step is completed, and DUDFTO continues to carry out the feedback-based binary search step.

If the measured RTT of p_2 is still high, it means that the time interval T_i has exceeded T_k, as shown in Eq. (2). Subsequently, DUDFTO assigns T_{over} as T_i, as shown in Eq. (4). And it assigns the increment τ as T_{i-1}, as shown in Eq. (5). Meanwhile, DUDFTO no longer increases the probing time interval according to Eq. (1). So far, the fast increment step is completed. DUDFTO start to carry out the feedback-based binary search step to achieve a more accurate probing of the timeout setting.

Example: To make the readers better understand DUDFTO, we give an example which contains two situations that occur in the fast increment step to explain how it probes timeouts. As shown in Fig. 3, a green packet indicates that its RTT is less than the threshold, while a red packet indicates that its RTT is greater than the threshold. The gray packet is used to trigger flow entry installation, and its RTT does not need to

be measured. In Fig. 3 (a), assuming that T_{hard} is 7s, T_{idle} is 0s, and the initial probing interval T_1 is 1s. According to Eq. (1), T_4 is 4s. T_4 is less than T_{hard}. However, in this case, it can be seen that the RTT of the packet is greater than the threshold. Therefore, DUDFTO determines that the increase in RTT is caused by sending two probing packets at T_4 intervals. When the RTT of the second probing packet is less than the threshold, it can be found that the reason for the increase in RTT is that the sum of probing time T_{sum} from round 1 to round 4 exceeds T_{hard}. At this point, T_{over} is denoted as 8s. τ is denoted as 4s. Subsequently, DUDFTO will further execute the feedback-based binary search step.

In Fig. 3 (b), assuming that T_{hard} and T_{idle} are 10s and 3s. According to the same probing process, DUDFTO infers that the reason for the increase of RTT is that T_4 is greater than T_{idle}. At this point, T_{over} is denoted as 4s. τ is denoted as 2s. DUDFTO will execute the feedback-based binary search step. DUDFTO can quickly infer the upper bound of the pre-probe timeout value T_k using the fast increment step without pre-probing hard or idle-timeout in advance, which is different from [15] and [17].

2) Feedback-Based Binary Search Step: At the feedback-based binary search step, DUDFTO starts to continuously send a set of probing packets until the measured RTT increases. Here, Eq. (6) is used to calculate the time interval T_i of these probing packets. In Eq. (6), λ is the minimum value of the probe interval increment. α is the number of incremental reductions. Meanwhile, β is the maximum value of the number of incremental reductions.

After sending each probing packet, DUDFTO measures its RTT. If there is no increase in the measured RTT, the current increment is unchanged. However, when the measured RTT increases, DUDFTO record the number of sent probing packets as *M*. Subsequently, DUDFTO again sends two probing packets (p_{M+1} and p_{M+2}) with time interval T_i and measures the RTT of these two probing packets.

After that, DUDFTO executes the following processing based on the measured RTT of p_{M+2}. According to whether the measured RTT of p_{M+2} is high, DUDFTO records two cases: Case 1 and Case 2.

Case 1: If the measured RTT of p_{M+2} is still high, it means that the time interval T_i has exceeded T_k. At this time, DUDFTO continues to use Eq. (6) to calculate the probing interval T_i and continues to send probing packets. It sets T_i back to T_{i-1}. Hence, the increment continues to decrease by half and DUDFTO notes the interval T_i as the new T_{over}. When the increment is less than or equal to λ, the probing accuracy has reached its maximum. So DUDFTO records the number of sent probing packets as *N*. Meanwhile, DUDFTO notes the current T_i as T_x. T_x is the undetermined idle-timeout or hard-timeout value.

$$T_i = T_{i-1} + \lim_{\frac{\tau}{2^\beta}\to\lambda}\left(\sum_{\alpha=1}^{\beta}\frac{\tau}{2^\alpha}\right), \left(\frac{\tau}{2^\beta} \geq \lambda, T_i < T_{over}\right) \quad (6)$$

Case 2: If the measured RTT of p_{M+2} is small, it means that T_i has not exceeded T_k. In this case, DUDFTO continues to send probing packets. Meanwhile, it starts to use Eq. (7) to calculate the probing interval T_i. That is because the increase of RTT is not caused by T_i exceeding T_k. T_i should not be reduced at this time.

The new T_i cannot be greater than T_{over}. When T_i plus the increment of the next round is greater than T_{over}, then the increment is reduced by half. After that, DUDFTO re-adds T_i with the new increment. Likewise, α is the number of incremental reductions, and β is the maximum value of the number of incremental reductions in Eq. (7). When the increment is less than or equal to λ, the probing accuracy has reached its maximum at this point. Here, DUDFTO records the number of sent probing packets as *N*. So DUDFTO notes the current T_i as T_x. T_x is the undetermined idle-timeout or hard-timeout values.

$$T_i = T_i + \lim_{\frac{\tau}{2^\beta}\to\lambda}\left(\sum_{\alpha=1}^{\beta}\frac{\tau}{2^\alpha}\right), \left(\frac{\tau}{2^\beta} \geq \lambda, T_i < T_{over}\right) \quad (7)$$

After probing T_x, DUDFTO needs to determine whether T_x is idle-timeout or hard-timeout values. It uses the following method to achieve that. Firstly, it no longer increases the probed time interval according to Eqs. (6) or (7). Subsequently, DUDFTO reduces T_x by Δt, and sends two packets (p_{N+1} and p_{N+2}) separately with $T_x - \Delta t$ as the sending interval. Here, $T_x - \Delta t$ must not exceed T_k. The reason is that, according to the Eqs. (6) and (7), it can be seen that T_x must be smaller than T_k.

After sending and measuring the RTT of p_{N+1} and p_{N+2}, DUDFTO sends another packet p_{N+3} with $2\Delta t$ as the time interval and measures its RTT.

If the measured RTT of p_{N+3} is significantly higher than that of p_{N+1} and p_{N+2}, it means that only hard-timeout is set in the switch and T_x is the hard-timeout value.

The reason is that if T_x is idle-timeout value, $T_x - \Delta t$ is less than the idle-timeout value. In that case, the RTT will not increase, which is inconsistent with the increase of the RTT of p_{N+3}. Similarly, when sending p_{N+3} using an interval of $2\Delta t$, the RTT will not increase since $2\Delta t$ must be less than the idle-timeout value. So if the RTT of p_{N+3} has increased significantly, it means that $T_x + \Delta t$ must greater than the hard-timeout value at this time, and no idle-timeout is set in the switch. In conclusion, in this case, T_x must be the hard-timeout value. Therefore, we can conclude that the idle-timeout value is 0, and the final probing result is shown in Eq. (8). In this case, the timeout probing is finished. However, the link delay and background traffic may affect the timeout probing accuracy, because they may increase the RTT. So the probed T_x needs to add the threshold of the measured RTT, denoted as η. It helps mitigate the impact of the high RTT caused by link delay when probing timeout.

$$T_{idle} = 0, \qquad T_{hard} = T_x + \eta \quad (8)$$

In contrast, if the measured RTT of p_{N+3} remains small, it indicates that T_x is not the hard-timeout value. It can only be the idle-timeout value. Whether there is hard-timeout in the flow entry needs to be further determined. So in this case, the feedback-based binary search step is finished. $T_x + \eta$ is the idle-timeout value, as shown in Eq. (9). Subsequently,

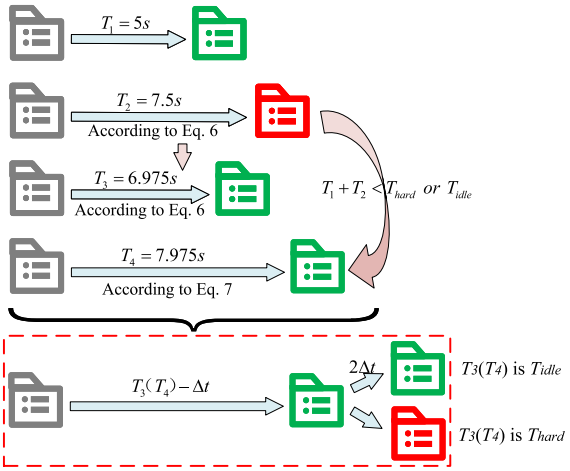


Fig. 4. The process of the feedback-based binary search step probing timeouts.

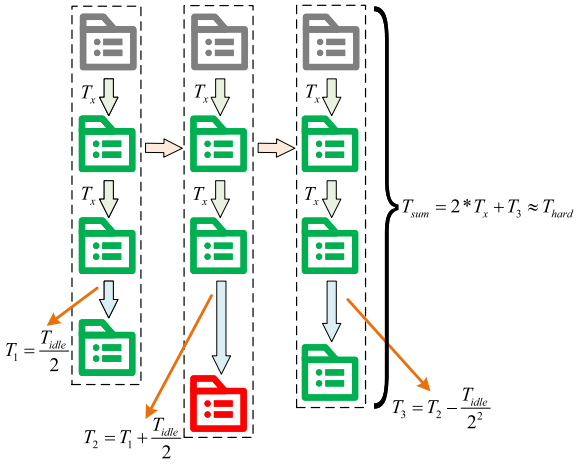


Fig. 5. The process of the remainder binary search step probing timeouts.

DUDFTO further performs the remainder binary search step to complete the hard-timeout probing.

$$T_{idle} = T_x + \eta, \qquad T_{hard} = TBD \tag{9}$$

Example: An example of how the feedback-based binary search step probes timeouts is shown in Fig. 4. Assuming that the initial sending interval T_1 between two packets is 5s. Based on this, DUDFTO begins to calculate the sending interval T_2 between two packets based on Eq. (6) until the RTT of packet increases. After that, DUDFTO infers whether T_2 is greater than T_k. If yes, DUDFTO continues to calculate the sending interval T_3 using Eq. (6). If $T_1 + T_2$ is greater than T_k, DUDFTO uses Eq. (7) to calculate the sending interval T_4.

When the RTT of a packet is bigger than the pre-defined threshold, DUDFTO records the current T_3 or T_4. Since the timeout type of T_3 (T_4) probed is unknown, DUDFTO further probes its type. Eventually, DUDFTO infers T_3 (T_4) is hard-timeout or idle-timeout. When T_3 (T_4) is idle-timeout, DUDFTO further performs the remainder binary search step. DUDFTO can infer the timeout value without setting the timeout value type of probing in advance, which is different from [15] and [17]. Furthermore, the way of determining why the RTT of the probing packet exceeds the threshold makes DUDFTO has ability to probe the timeout value T_k more accurately.

3) Remainder Binary Search Step: In this step, DUDFTO further probes the hard-timeout value based on the probed idle-timeout value. In this case, DUDFTO first periodically sends probing packets with T_x as interval. When DUDFTO continues to send packets for a duration greater than T_{max} and there is no increase in the measured RTT, it means that switch has not set hard-timeout.

If the measured RTT of any probing packets increases, the number of cycles is noted as *R*. Then, the hard-timeout can be expressed as in Eq. (10).

$$T_{hard} = (R - 1) \times T_{idle} + \Delta x + \eta \tag{10}$$

In this case, Δx is less than or equal to the idle-timeout value. In order to quickly calculate the value of Δx, DUDFTO uses the binary search, to step by step calculate Δx. Firstly, DUDFTO continuously sends probing packets until the number of transmissions is equal to $R - 1$. The sending interval of these probing packets is set as T_{idle}. After that, DUDFTO sends two probing packets (p_{R+1} and p_{R+2}) with time interval Δx and measures the RTT of these two probing packets. When the RTT of p_{R+2} is greater than η, DUDFTO uses Eq. (11) to reduce Δx. When the RTT of p_{R+2} is less than η, Eq. (11) is used to increase Δx. DUDFTO repeats the process until $T_{idle}/2^i <= \lambda$. η denotes the RTT threshold used to determine whether the flow entry expires. *D* is the RTT of a probing packet. *i* means the *i*th round probing. Finally, when $T_{idle}/2^i <= \lambda$, the value of Δx is calculated using Eq. (12).

$$\Delta x = \begin{cases} T_{idle}/2, & i = 1 \\ \Delta x + T_{idle}/2^i, & i > 1 \text{ and } D <= \eta \\ \Delta x - T_{idle}/2^i, & i > 1 \text{ and } D > \eta \end{cases} \tag{11}$$

$$\Delta x = \Delta x + T_{idle}/2^i \tag{12}$$

Example: An example of how the remainder binary search step probes timeouts is shown in Fig. 5. Assuming that the idle-timeout value probed in the feedback-based binary search step is T_x. $T_{hard} = 2 * T_x + \Delta x$. Before each time DUDFTO starts to probe timeouts and calculates the sending interval Δx of two packets, DUDFTO wills send three packets firstly, two of which are sent at T_x interval. The initial $\Delta x = T_1 = T_{idle}/2$. If RTT does not exceed the threshold at this time, $\Delta x = T_2 = T_1 + T_{idle}/2$. Otherwise, $\Delta x = T_3 = T_2 + T_{idle}/2^2$. When the increment is less than λ, DUDFTO can infer $T_{hard} = 2 * T_x + T_3$. Based on this, DUDFTO can infer the hard-timeout value with a lower bound, which makes its hard-timeout probing accuracy maintain a high level.

4) Link Failure Verification: DUDFTO effectively mitigates issues caused by high RTT and interference between hard-timeout and idle-timeout, thereby improving probing accuracy. However, it remains susceptible to link failure [35], which can disrupt timeout probing. Unlike the high RTT caused by the link delay, the link failure may result in a

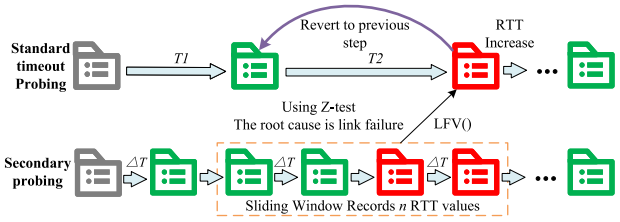


Fig. 6. The link failure verification.

sudden increase or unmeasurable RTT for the probe packets. This can cause timeout probing algorithms to misattribute the increased or unmeasurable RTT to either a hard timeout or idle timeout, leading to erroneous results. While DUDFTO's feedback-based timeout probing algorithm helps mitigate this issue, it does not fully eliminate the impact of link failure.

To further enhance the robustness of DUDFTO in probing timeout values, we introduce a link failure verification method, `LFV`. This method determines whether an increase or unmeasurable RTT of probing packets is caused by the link failure. Specifically, in addition to DUDFTO's standard timeout probing process, we introduce a secondary probing process that continuously sends probing packets at fixed small intervals. After that, using a sliding window, we record the *n* most recent RTT values linked to the increased or unmeasurable RTT observed during DUDFTO's standard probing. For a probing packet with an unmeasurable RTT, we set its RTT as the fixed maximum value t_{up}.

The reason for this is that, in the presence of the link failure, the RTTs of multiple probing packets in the secondary probing process will also increase or become unmeasurable. In contrast, if the increased or unmeasurable RTT is caused by the idle timeout or hard timeout, only one RTT will change drastically. Accordingly, considering that difference, we apply a Z-test to determine whether the link failure is the reason of the increased or unmeasurable RTT. When DUDFTO probes an increased or unmeasurable RTT, `LFV` evaluates the root cause using the collected *n* RTT values and Z-test. If the root cause is not attributed to the link failure, DUDFTO continues timeout probing as usual. Conversely, if the Z-test confirms the link failure, DUDFTO reverts to the previous step to re-evaluate the timeout value and reapply `LFV`. If both tests confirm the link failure, DUDFTO halts probing and waits for a stable network state before resuming timeout probing.

Fig. 6 illustrates the link failure verification process. When the link failure is confirmed, DUDFTO re-evaluates the timeout and performs additional checks before proceeding. This mechanism minimizes the impact of link failure on timeout probing. In scenarios with frequent link failure, it also allows DUDFTO to exit probing in a timely manner, preventing incorrect results.

The pseudocode for timeout probing is shown in Algorithm 1. The input consists of the destination *dst* and match fields *F*. In addition, in the algorithm, ϕ represents the number of probing cycles. φ represents whether Eq. (1) is used. τ represents the increment of the probing interval. λ represents the minimum increment of the probing interval. *dst* represents destination IP. *F* represents probed match fields. *out* represents whether *Feedback-based binary search step* is completed. *PUSH* represents whether DUDFTO continues to send probing packets in cycles with T_i intervals. σ is the probed hard-timeout. η is the RTT threshold. `LFV()` denotes the link failure verification method.

In Algorithm 1, DUDFTO first performs the fast increment step. DUDFTO uses Eq. (1) to roughly probe the timeout value in lines 1 to 8. When the probed timeout value is greater than T_{max} and the RTT is less than η, DUDFTO infers that both the hard-timeout and idle-timeout are 0. Otherwise, DUDFTO continues to perform the feedback-based binary search step and probes timeout value using Eqs. (6) and (7) from lines 9 to 43. During probing, DUDFTO will terminate probing when the link failure verification method proves that the increased or unmeasurable RTT is caused by link failure. In contrast, DUDFTO will continue with timeout probing. If the probed timeout value is hard-timeout, DUDFTO infers that the hard-timeout is $T_i + \eta$ and the idle-timeout is 0. If the probed timeout value is idle-timeout, DUDFTO infers the idle-timeout is $T_i + \eta$. In this time, DUDFTO will continue to perform the remainder binary search step and probe the hard-timeout value using Eq. (11) from lines 44 to 65. Similarly, in that step, DUDFTO will perform the link failure verification method. When no link failure occurs, DUDFTO infers that the hard-timeout is $\sigma + \eta$. If there is a link failure, the probing is stopped.

When both the hard and idle-timeout are set, the time complexity of DUDFTO is the highest. In this case, the time for DUDFTO to perform the fast increment step will be greater than or equal to T_{hard}. Therefore, the time complexity of DUDFTO to perform the fast increment step is $O(T_{hard})$. For the feedback-based binary search step, DUDFTO will perform it repeatedly to probe the idle-timeout value. Assuming that the feedback-based binary search step is executed *C* times, the time complexity of carrying out the feedback-based binary search step is $O(C * T_{idle})$. For the remainder binary search step, DUDFTO will also perform it repeatedly to probe the hard-timeout value. Assuming that the remainder binary search step is executed *K* times, the time complexity of carrying out the remainder binary search step is $O(K * T_{hard})$. The timeout probing of DUDFTO is divided into the above three steps, so the time complexity of Algorithm 1 is $O(K * T_{hard} + C * T_{idle})$.

C. Launching Flow Table Overflow Attack

DUDFTO can launch the flow table overflow attack through the probed match fields and timeout settings. DUDFTO forged *C* flow table overflow attack packets by using the probed match fields, where *C* represents the maximum capacity of the flow table [17]. For example, for the flow table that uses the source IP address, destination IP address, and protocol as the match fields, DUDFTO can generate different flow table overflow attack packets by modifying the source IP address of the packets.

After that, DUDFTO calculates the flow table overflow attack packet sending rate based on the probed timeout settings and the maximum flow table capacity. It ensures that attack

Algorithm 1 Timeout Probing Algorithm

Require: Destination *dst*, Feature set F
Ensure: Hard timeout T_{hard}, Idle timeout T_{idle}
Send initial packet pkt_0 ▷ Fast increment step
while out = True **do**
  Measure RTT $D_0[i] \leftarrow \text{send\_packet}(pkt)$
  if $(D_0 < \eta \vee \phi = 1) \wedge \varphi = 1$ **then**
    Set τ = Period_Time, calculate T_i via Eq. (1), $\phi \leftarrow \phi + 1$
    if $T_i \geq T_{\max}$ **then return** $(0, 0)$ ▷ No T_{hard}, T_{idle}
    end if
    sleep(T_i)
  else if $D_0 > \eta \wedge \phi \neq 1$ **then** ▷ Feedback-based binary search
    Update $Fail\_num$ via LFV(), sleep(T_i)
    Measure $D_1[i] \leftarrow \text{send\_packet}(pkt)$
    if $D_1 > \eta$ **and** LFV()=1 **then**
      $Fail\_num \leftarrow Fail\_num + 1$
      if $Fail\_num \geq 2$ **then** ▷ Double failure
        return Frequent link failure
      else
        Measure $D_1[i] \leftarrow \text{send\_packet}(pkt)$
        if $D_1 > \eta$ **and** LFV()=1 **then** ▷ Double failure
          return Frequent link failure
        else if $D_1 > \eta$ **then**
          $T_{\text{over}} \leftarrow$ Period_Time, $\psi \leftarrow$ False, $Fail\_num \leftarrow 0$
        else
          Calculate T_i, τ via Eq. (7), sleep(T_i), $Fail\_num \leftarrow 0$
        end if
      end if
    else if $D_1 > \eta$ **and** LFV()=0 **then**
      Set $T_{\text{over}} \leftarrow$ Period_Time, $\psi \leftarrow$ False, $Fail\_num \leftarrow 0$
    else
      Eq.(7): T_i, τ & sleep(T_i)
    end if
    $\varphi \leftarrow 0$
    if ψ = False **then**
      Calculate T_i, τ via Eq. (6), update ϕ
    end if
    if $\tau \leq \lambda$ **then** $out \leftarrow$ False ▷ Probing accuracy maximized
    end if
  else
    Calculate T_i and τ via Eq. (7), sleep(T_i)
    if $\tau \leq \lambda$ **then** $out \leftarrow$ False ▷ End binary search
    end if
  end if
end while
Measure $D_{2:4}[i]$ with intervals $T_i - \Delta t$ and $2\Delta t$
if Secondary LFV() = *True* **then** ▷ Remainder binary search step
  return Frequent link failure
else if $D_4 < \eta$ **then** ▷ Idle-timeout exists
  send_packet(pkt_0), initialize $nums \leftarrow 0$
  while Push = True **do** ▷ PUSH loop
    sleep(T_i), Measure $D_5[i] \leftarrow \text{send\_packet}(pkt)$
    if Secondary LFV() = *True* **then**
      return Frequent link failure
    else if $D_5 > \eta$ **then** Push = Flase ▷ Terminate the loop
    end if
    if $nums \times T_i > T_{\max}$ **then**
      return $(0, T_i{+}\eta)$ ▷ Max cycle reached, only T_{idle}
    end if
    $nums \leftarrow nums + 1$
  end while
  sleep(T_i), Measure $D_6[i] \leftarrow \text{send\_packet}(pkt)$
  Compute T_i, τ via Eq. (11)
  if $\tau \leq \lambda$ **then return** $(\sigma{+}\eta, T_i{+}\eta)$ ▷ Out T_{hard}, T_{idle}
  end if
else
  return $(T_i{+}\eta, 0)$ ▷ No T_{idle}
end if

packets can be refreshed regularly, so that flow entries can survive in the flow table for a long time. Here, DUDFTO calculates different attack rates based on different timeout settings.

According to the probed timeout settings utilized in the flow entries, four combinations of hard-timeout and idle-timeout can be summarized as follows. (I) No hard-timeout and no idle-timeout are set. (II) Only hard-timeout is set. (III) Only idle-timeout is set. (IV) Both idle-timeout and hard-timeout are set.

Due to the constant attack rate of the existing attacks, the attacks will show obvious features, which makes the detection system easy to detect them. Therefore, in order to stealthily overflow the switch, DUDFTO designs a dynamical attack packets sending algorithm. The attack rate is randomly adjusted by uniform distribution based on timeout settings.

Since the flow entries expire in category (I) is independent of the timeout setting, and it is not regularly used, this paper does not consider category (I).

In **category (II)**, when a flow entry reaches the specified hard-timeout, the flow entry is forcibly deleted. Therefore, in order to overflow the flow table, it is necessary to ensure that the flow entries remain in the flow table. DUDFTO utilizes Eq. (13) to calculate the sending rate of attack packets. C denotes the maximum flow table capacity, and T denotes the probed hard-timeout. When a flow entry has been deleted, DUDFTO uses Eq. (13) to make that flow entry to be reinstalled. Finally, the purpose of the attack is achieved.

$$V_{min} = \frac{C}{T} \tag{13}$$

For example, assuming the hard-timeout value is 60s, and the flow table capacity is 1200, in this case, the corresponding sending rate of attack packets is 20pps. It guarantees that the flow table is always in an overflow state. Meanwhile, DUDFTO keeps the sending time interval between two identical attack packets at about 60s. This allows flow entries to be quickly reinstalled when they are forced to be deleted due to hard-timeout. And during a hard-timeout, each flow entry produces only one packet. Thus, DUDFTO generates the least amount of communication messages during the whole attack process and achieves the purpose of overflowing the flow table.

In **category (III)**, the flow entry will only be deleted when no corresponding packet is received within the idle-timeout period. So when overflowing flow table, DUDFTO only needs to send the same attack packet again within the idle-timeout period. It will ensure the long-term survival of the flow entry corresponding to the attack packet.

Here, DUDFTO utilizes Eqs. (14), (15) and (16) to calculate the sending rate of attack packets. V_{upper} denotes the upper bound of the send rate. V_{lower} denotes the lower bound of the send rate. *Random* is a random function. χ is the coefficient that denotes how many times V_{upper} is surpassed

than V_{lower}. Using χ, we can adjust the range of rate fluctuations. Meanwhile, to ensure the long-term survival of flow entries without excessively high packet transmission rates, χ is defined as a real number ranged from 1 to 2. When χ is set as 2, it means V_{upper} is twice of V_{lower}, allowing the rate V_i to fluctuate between V_{upper} and V_{lower} according to Eq. (14).

$$V_i = Random\big(V_{upper}, V_{lower}\big) \tag{14}$$

$$V_{lower} = \frac{C}{T_{idle}} \tag{15}$$

$$V_{upper} = \frac{\chi \times C}{T_{idle}} \tag{16}$$

For example, assuming the idle-timeout is 60s, and the flow table capacity is 1200, and χ is 2, at this point, the upper bound on the sending rate is 40pps and the lower bound is 20pps. Finally, the rate used to send attack packets varies randomly between 20 and 40pps.

In **category (IV)**, even the flow entries generated by the attack packets does not expire due to the exist of idle-timeout, they will still be deleted when their survival time reach the hard-timeout. Therefore, in addition to gradually overflowing the flow table during the idle-timeout and periodically refreshing the flow entries, an attacker also needs to reinstall the flow entries after they are deleted due to hard-timeout.

To solve this problem, DUDFTO calculates the attack rates based on the relationship between the probed hard-timeout and the idle-timeout. If the hard-timeout is an integer multiple of the idle-timeout, DUDFTO continues to use Eqs. (14), (15), and (16) to calculate the sending rate of attack packets.

For the case that the hard-timeout is not an integer multiple of the idle-timeout, we divided it into the divisible part and the remainder part. We separately calculate different packet sending rates for the divisible part and the remainder part. For the divisible part, we calculate the rate using Eqs. (14), (15), and (16). Here, the denominator is still the idle-timeout. For the remainder of the hard-timeout divided by the idle-timeout (the remainder part), we treat it as a complete packet sending cycle. We still calculate the rate based on Eqs. (14), (15), and (16). But for Eqs. (15) and (16), the denominator is the remainder. Through these two parts, the hard-timeout which is not an integer multiple of the idle-timeout is converted to the hard-timeout that is "an integer multiple" of the idle-timeout. It ensures that flow entries will not be deleted due to idle-timeout.

For example, suppose that the hard-timeout is 65s, the idle-timeout is 30s, flow table capacity is 1200 and χ is 2, the hard-timeout is not an integer multiple of the idle-timeout. In this case, we separately calculate different packet sending rates for the divisible part and the remainder part. For the divisible part, the upper bound on the sending rate is 80pps and the lower bound is 40pps. For the remainder part, The remainder of the hard-timeout divided by the idle-timeout is 5s. So when DUDFTO calculates the sending rate based on Eqs. (14), (15), and (16) for the remainder part, the denominator of Eqs. (15) and (16) becomes 5s. Therefore, the upper bound of the sending rate is 480pps and the lower bound is 240pps.

We notice that the rate of the remainder part gets larger, because the denominator gets smaller and the numerator stays the same for Eqs. (15) and (16). So we further consider the following method to reduce the rate. When the remainder of the hard-timeout divided by the idle-timeout is quite smaller than the idle-timeout value (etc. the remainder is 1s and the idle-timeout is 30s), we do not process the remainder part because the interval between the current remainder and the next send cycle is small. After a short time, the flow table can still be overflow.

In Eqs. (15) and (16), the attacker considers overflowing the flow table without other flows. However, in real network, there are benign flow entries in the flow table, and the flow table may also use the replacement algorithms. Therefore, we analyze the possible impact of the replacement algorithm on DUDFTO. We mainly analyze the possible influence of FIFO and LRU. For FIFO, it will replace the flow entry which is installed in the switch firstly when the flow table is full. In this case, the likelihood of malicious flow entries generated by DUDFTO being replaced is similar to benign flow entries. For LRU, it replaces the least recently used flow entry. In this case, for benign small flows (only a small number of packets), the likelihood of malicious flow entries generated by DUDFTO and benign flows being replaced is similar. However, for benign large flows, LRU may preferentially replace flow entries generated by DUDFTO. But since DUDFTO continuously send attack packets that can match the malicious flow entries, it can mitigate this impact to some extent.

D. Inferring Flow Table State

In real network, due to the existence of benign traffic, the flow table state is constantly changing. Therefore, we can only infer whether flow table capacity is full, so as to determine whether DUDFTO should stop sending new attack packets.

When a packet does not match any flow entry, and the flow table is not fully occupied, its RTT remains at about ρ [16]. But when the flow table capacity is full, if a new packet cannot be matched by the existing flow entries, the measured RTT is larger than ρ. Sometimes the RTT cannot be measured even after waiting for a long time. The reason is that when the flow table is full, if the switch receives new packets, it must clear the old flow entry in the flow table, and this clearing process will lead to a larger RTT.

DUDFTO considers ten packets as a group, and measures the RTT of the attack packets while sending them. DUDFTO continuously sends new attack packets, when there is a more significant increase in RTT, it can indicate that the flow table is full. Through several probing, DUDFTO can roughly infer the number of attack packets needed to fill the flow table from the number of probe packets sent, thereby stopping sending new attack packets. Meanwhile, when the flow table storage resource is full, in order to overflow the flow table more efficiently, DUDFTO will continue to send at least N packets, as previous work [17].

IV. Evaluation

In the evaluation, we first introduce the experimental setup, which includes the relevant environment and the parameter settings used in the experiments. To evaluate DUDFTO, we compared DUDFTO with LOFT and RAFA to demonstrate its effectiveness in terms of match fields probing ability, timeout

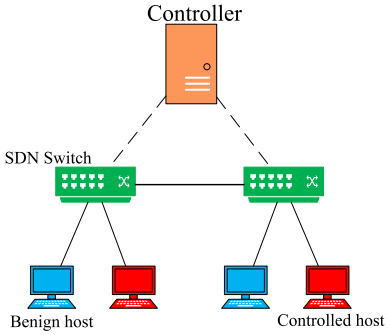


Fig. 7. Experiment Topology.

probing relative error, number of timeout probing packets, number of packet_in and flow_mod messages generated by the attack, rate distribution of packet_in and flow_mod messages generated during the attack, and number of detected attack packets. We note that LOFT comprehensively studied probing match fields, probing timeout settings and launching flow table overflow attack based on the probed information. It is similar to our research, so comparing LOFT with DUDFTO can prove whether DUDFTO is valid. For other attacks, for example, [18], [19] and [21], their research are either different from DUDFTO or we cannot adequately compare DUDFTO with them. Therefore, we decided to mainly compare DUDFTO with LOFT. Furthermore, to analyze the effectiveness of DUDFTO's timeout probing algorithm, we compared it with LOFT and RAFA.

A. Experiment Setup

In the evaluation, RYU v4.9.1 was used as the controller. Meanwhile, the SDN network was simulated using Mininet v2.3. All experiments were run on a Vostro 3681 computer with 8gb RAM, Intel i5-10400 CPU, and Windows 10 operating system. According to the earlier work [17], Mininet was used to build the SDN network topology. It contains two switches, each of which can install 1000 TCAM-based flow rules. The two controlled hosts are connected to two switches respectively. As previous works [17], the network topology used in the evaluation is shown in Fig. 7. Meanwhile, η is set to 0.01, 0.31 0.61. χ is set to 1.1.

DUDFTO is proposed to probe match fields and timeout settings, and launch flow table overflow attack to overwhelm the SDN switches. Therefore, we compare DUDFTO with LOFT in terms of match fields probing ability, timeout probing relative error, number of packet_in and flow_mod messages generated by the attack, rate distribution of packet_in and flow_mod messages generated during the attack. However, as RAFA mainly focuses on probing timeouts, so DUDFTO is compared with RAFA in terms of timeout probing relative error and number of timeout probing packets.

Moreover, to better verify the stealthily attack ability, we run DUDFTO and LOFT attacks separately in the same intrusion detection method. Some intrusion detection methods have been proposed against flow table overflow attack [28], [29], [30], [31], [32], [36]. Among them, [28] presents a method for detecting and mitigating flow table overflow attack, named FTMaster. Firstly, FTMaster extracts the features of flow table to determine whether the flow table is under attack. Subsequently, it uses various features of flow entries to find the attack flow entries of the attacked flow table. Finally, it blocks the attack network flows based on the detected flow entries. FTMaster can effectively defend against flow table overflow attack, so we chose [28] as the intrusion detection method in the evaluation. We collect the flow entries every second, including malicious flow entries and the flow entries deleted by the intrusion detection method (which is recorded as detected flow entries). By analyzing the number of malicious flow entries and detected flow entries of DUDFTO and LOFT, we can evaluate the stealthiness of DUDFTO and LOFT.

TABLE III
TIMEOUT VALUE SETTINGS

Setting	idle-timeout	hard-timeout
I_1	5	10
I_2	10	30
I_3	20	60
I_4	30	80
I_5	30	120
I_6	40	240
I_7	50	400
I_8	60	600

B. Evaluation Metrics

In the following evaluation, to effectively compare the timeout probing performance of DUDFTO with LOFT and RAFA in challenging link delays, we replay the real network traffic from [26] as the background traffic, and set the link delays to 0ms, 100ms, and 200ms respectively. Here, we set different link delays mainly to change the RTT of the probing packet measured by DUDFTO to simulate different network environments. Different from [33], we do not consider the influence of link delay on the installation of flow entries, and we mainly use the difference of RTT or one-side transmission delay between packets to probe the match fields and timeout values. Then, we configure eight sets of different timeout settings as the probing scenarios. The hard-timeout values are set as 10 to 600 seconds and the idle-timeout values are set as 5 to 60 seconds, as shown in Table III. After that, we use the last four sets of timeout settings to compare DUDFTO and LOFT in terms of the number of packet_in and flow_mod messages generated by the attack and the rate distribution of packet_in and flow_mod messages generated during the attack. When comparing the stealthiness of DUDFTO and LOFT, the idle-timeout value is set to 10 seconds, and the attack is launched at the 50th seconds. Other settings are same as [28].

To validate the effectiveness of DUDFTO's link failure verification method, we simulate different link failure environments and compare the timeout probing performance of DUDFTO, LOFT, and RAFA under various failure scenarios. Specifically, we consider two link failure scenarios: (1) a failure occurring every 2 seconds, representing a high-failure-frequency SDN environment, and (2) a failure occurring every 60 seconds, representing a typical SDN failure environment.These scenarios are employed to evaluate whether our method can handle both frequent and occasional failures.

Additionally, we select three timeout settings—I_1, I_4, and I_8—to represent small, medium, and large timeout scenarios. In this experiment, we denote DUDFTO without link failure verification as DUDFTO-NoLFV and DUDFTO with link failure verification as DUDFTO-LFV.

DUDFTO is compared in terms of the ability of probing match fields, the relative error of probing timeout settings, the number of packet_in and flow_mod messages generated by the attack, the rate distribution of packet_in and flow_mod messages generated during the attack, the number of malicious and detected flow entries of DUDFTO and LOFT. The ability of probing match fields indicates whether the attacker can precisely infer match fields. The timeout probing relative error indicates how close the probed timeout value is to the real value. The number of packet_in and flow_mod messages generated by the attack can visually represent the amount of information generated by the attack, so it can be used to measure the stealthiness of the attack. The rate distribution of packet_in and flow_mod messages generated by the attack can indicate the fluctuating variation of the attack, and can be used to further measure the performance of the attack. The number of malicious and detected flow entries of DUDFTO and LOFT indicates their likelihood of being detected by the intrusion detection method.

C. Analysis of the Ability of Probing Match Fields

In this section, the ability of probing match fields of DUDFTO and LOFT is compared. We found that LOFT can not probe $ipv4\_src$ and $ipv6\_src$ precisely. In the following we will analyze the reason for this result from the probing method, probing process, and probing results.

When probing match fields, LOFT infers the match fields by measuring the RTT of the probing packets. In contrast, DUDFTO infers the match fields by measuring the one-sided transmission delay of the probing packets.

To probe whether the $ipv4\_src$ field is used in the match fields, LOFT first constructs and sends a probing packet containing the $ipv4\_src$ field to make the flow table install a flow entry. After that, LOFT constructs a new probing packet by modifying the $ipv4\_src$ field of the probing packet and sends it. Meanwhile, LOFT measures the RTT of probing packets to infer whether a new flow entry is installed, so as to infer whether $ipv4\_src$ is used in the matching fields. The same process is used for $ipv6\_src$.

However, due to the $ipv4\_src$ field of the probing packet sent by LOFT is forged, the receiver cannot find the sender when there is no corresponding IP-MAC mapping in the ARP table of the receiver. Thus, the receiver cannot send the response packet back to the sender. Therefore, LOFT cannot measure the RTT of probing packets, so it cannot infer whether $ipv4\_src$ is included in the match fields. The same phenomenon occurs in probing $ipv6\_src$.

For DUDFTO, it infers whether the $ipv4\_src$ field is included in the match fields by measuring the one-sided transmission delay of the probing packet. After DUDFTO constructing a probe packet containing the $ipv4\_src$ field and sending it, DUDFTO measures its one-sided transmission delay at the receiver, instead of measuring the RTT. Since the receiver's address is real, although the sender cannot be found because $ipv4\_src$ has been modified, DUDFTO can also measure the one-sided transmission delay. In conclusion, DUDFTO has ability to probe whether the $ipv4\_src$ and $ipv6\_src$ fields are included in the match fields, while LOFT cannot probe these fields.

D. Comparison of Probing Timeout

In this subsection, the timeout probing performances of DUDFTO, RAFA, and LOFT are compared, in terms of the hard-timeout probing relative error $Error_h$, idle-timeout probing relative error $Error_i$, and number of timeout probing packets. Here, $Error_h$ and $Error_i$ are calculated using Eqs. (17) and (18). Meanwhile, in this section, we set different link delays as 0ms, 100ms, and 200ms to evaluate the timeout probing relative error of DUDFTO, RAFA, and LOFT under different link delays.

$$Error_h = \frac{|H_t - H_p|}{H_t} \tag{17}$$

$$Error_i = \frac{|I_t - I_p|}{I_t} \tag{18}$$

Here, H_t and I_t denote the true hard-timeout value and true idle-timeout value, respectively. H_p and I_p denote the probed hard-timeout value and probed idle-timeout value, respectively.

Table IV shows H_p of DUDFTO, RAFA and LOFT. It can be observed that when the link delay are 0ms, 100ms, and 200ms, most timeout relative errors of DUDFTO, RAFA, LOFT are quite small. According to Table IV, when the link delay is 0ms, we can also see that DUDFTO's hard-timeout probing relative error and idle-timeout probing relative error are smaller than RAFA and LOFT under most timeout settings. For example, when the true hard-timeout value is 10s, the hard-timeout value probed by DUDFTO is 9.91s (0.9%). The hard-timeout values probed by LOFT and RAFA are 10.15s (1.5%), 10.54s (5.4%), respectively. At this point, the hard-timeout values probed by LOFT is 11s by rounding the probed timeout values. Hence, LOFT will make incorrect inferences of hard-timeout value.

In Table IV, when the link delays are 100ms and 200ms, we can observe that the probed hard-timeout of DUDFTO, RAFA and LOFT are affected to different degrees. Although the relative errors of DUDFTO and RAFA increased, they always remained below 0.5s. However, for LOFT, its relative errors were greater than 0.5s under most timeout settings, which indicates that high RTT may affect the timeout probing accuracy of LOFT.

Table V shows I_p of DUDFTO, RAFA, and LOFT, when the link delays are 0ms, 100ms and 200ms. We can also observe that the timeout probing relative error of DUDFTO, RAFA and LOFT are affected by link delays. But unlike the results in Table IV, the idle-timeout probing relative error of RAFA is obviously increased, especially for large link delays. For instance, when link delays are 100ms and 200ms, in $I_2, I_3, I_4, I_5, I_6, I_7, I_8$, the probed idle-timeout values are quite smaller than the real idle-timeout values. The result may be

TABLE IV
THE COMPARISON OF THE HARD-TIMEOUT PROBING RELATIVE ERROR UNDER I_1, I_2, I_3, I_4, I_5, I_6, I_7, AND I_8

ALG \ T_{hard} / Delay		10	30	60	80	120	240	400	600
0ms	DUDFTO	**9.91(0.9%)**	**29.91(0.3%)**	**59.94(0.1%)**	**79.91(0.1%)**	**119.95(0.04%)**	**239.94(0.03%)**	399.94(0.02%)	599.93(0.01%)
	RAFA	10.15(1.5%)	30.41(1.4%)	60.1(0.17%)	79.66(0.43%)	120.13(0.11%)	240.11(0.05%)	**400.03(0.01%)**	599.96(0.01%)
	LOFT	10.54(5.4%)	30.61(2.0%)	60.33(0.55%)	80.27(0.34%)	120.13(0.11%)	240.21(0.09%)	399.92(0.02%)	600.07(0.01%)
100ms	DUDFTO	**9.84(1.6%)**	**29.85(0.5%)**	59.80(0.33%)	79.82(0.23%)	**119.84(0.13%)**	239.67(0.14%)	399.64(0.09%)	599.56(0.07%)
	RAFA	10.21(2.1%)	30.43(1.4%)	**59.92(0.13%)**	**79.96(0.05%)**	120.25(0.21%)	240.28(0.12%)	**399.94(0.02%)**	**599.91(0.02%)**
	LOFT	10.92(9.2%)	30.21(0.7%)	60.38(0.63%)	80.41(0.51%)	120.63(0.53%)	**240.16(0.06%)**	400.58(0.14%)	600.64(0.11%)
200ms	DUDFTO	9.81(1.9%)	**29.80(0.7%)**	**59.81(0.32%)**	79.61(0.48%)	119.75(0.21%)	239.65(0.15%)	399.62(0.09%)	599.52(0.08%)
	RAFA	**10.10(1.0%)**	29.32(2.27%)	59.74(0.43%)	79.67(0.41%)	**119.91(0.08%)**	**240.04(0.02%)**	**400.15(0.04%)**	**599.94(0.01%)**
	LOFT	11.47(14.7%)	29.14(2.9%)	58.91(1.82%)	**79.97(0.03%)**	119.11(0.74%)	239.31(0.29%)	401.68(0.42%)	600.73(0.12%)

TABLE V
THE COMPARISON OF THE IDLE-TIMEOUT PROBING RELATIVE ERROR UNDER I_1, I_2, I_3, I_4, I_5, I_6, I_7, AND I_8

ALG \ T_{idle} / Delay		5	10	20	30 in I_3	30 in I_4	40	50	60
0ms	DUDFTO	**4.97(0.6%)**	**9.97(0.3%)**	**19.97(0.15%)**	**29.97(0.1%)**	**29.97(0.1%)**	**39.97(0.08%)**	**49.96(0.08%)**	**59.94(0.1%)**
	RAFA	5.15(3.0%)	10.04(0.4%)	20.1(0.5%)	30.1(0.33%)	30.20(0.67%)	40.02(0.5%)	50.1(0.2%)	60.1(0.17%)
	LOFT	4.70(6.0%)	10.25(2.5%)	19.79(1.05%)	29.49(1.7%)	30.20(0.67%)	39.70(0.75%)	50.14(0.28%)	60.14(0.23%)
100ms	DUDFTO	4.85(3.0%)	**9.84(1.6%)**	**19.76(1.2%)**	**29.81(0.63%)**	**29.80(0.67%)**	**39.81(0.47%)**	**49.79(0.42%)**	59.65(0.58%)
	RAFA	**5.13(2.6%)**	4.3(57.0%)	6.1(69.5%)	7.0(76.7%)	8.5(71.7%)	11.9(70.3%)	15.5(69.0%)	19.04(68.27%)
	LOFT	4.85(3.0%)	10.68(6.8%)	19.39(3.05%)	28.33(5.57%)	30.54(1.8%)	37.84(5.4%)	51.89(3.78%)	**60.33(0.55%)**
200ms	DUDFTO	4.62(7.6%)	**9.61(3.9%)**	**19.61(2.0%)**	**29.61(1.3%)**	**29.61(1.3%)**	**39.61(0.98%)**	**49.74(0.52%)**	**59.58(0.7%)**
	RAFA	**5.30(6.0%)**	4.3(57.0%)	6.2(69.0%)	7.0(76.7%)	8.5(71.7%)	11.8(70.5%)	15.4(69.2%)	18.64(68.93%)
	LOFT	3.37(32.6%)	7.79(22.1%)	19.03(4.85%)	29.18(2.73%)	28.39(5.37%)	39.59(1.02%)	48.67(2.66%)	58.86(1.9%)

caused by its idle-timeout probing method. RAFA probes idle-timeout by gradually increasing the sleep step. If the total sleep time does not exceed the hard-timeout and the RTT increases, RAFA will take the last sleep time plus the RTT as the idle-timeout. However, RAFA calculates the total sleep time by adding the sleep time without adding the RTT. When the RTT is large, the total sleep time plus multiple real RTT may exceed the hard-timeout. At this moment, RAFA will recognize that the deletion of the flow entry is caused by idle-timeout rather than hard-timeout. Hence, it will record the sleep time plus RTT as the probed idle-timeout. In this case, the probed idle-timeout will be much smaller than the true idle-timeout value.

From Tables IV and V, we can also see that the timeout probing relative errors of DUDFTO gradually decrease with the increase of timeout values. This is because we calculate the timeout probing relative error based on Eqs. (17) and (18). In Eqs. (17) and (18), the true timeout value is the denominator. The absolute value of the probed timeout value subtracting the true timeout value is the numerator. Since the relative error between the probed timeout value by DUDFTO and the timeout value steadily remains between 0.03-0.42s, the timeout probing relative errors will decrease as the timeout value increases. Moreover, it should be noted that in this work, to more accurately evaluate the timeout probing relative error, the probed timeout values are listed by using the values with a fractional part instead of integer values. However, when launching the flow table overflow attack, the probed timeout values are integer values obtained by rounding off operation.

In addition, when the link delay is 100ms and 200ms, we can see that DUDFTO's idle-timeout probing relative error is smaller than LOFT under most timeout settings. For example, when the true idle-timeout is 20s, the idle-timeout values probed by DUDFTO are 19.76s (0.15%), 19.61 (2.0%), respectively, and the idle-timeout values probed by LOFT are 19.39s (3.05%), 19.03s (4.85%), respectively. This is because LOFT does not consider the impact of RTT when probing idle-timeout, so the error of idle-timeout probed by LOFT is large when the RTT is high.

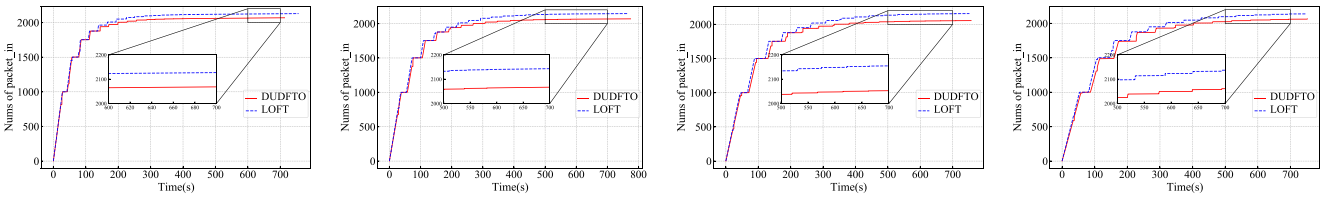
Table VI shows the number of timeout probing packets generated by DUDFTO, LOFT and RAFA under I_1, I_2, I_3, I_4, I_5, I_6, I_7, and I_8. We can see that the number of timeout probing packets generated by DUDFTO is less than RAFA and LOFT under I_1, I_2, I_3, I_4, I_5, I_6, I_7, and I_8. For example, when DUDFTO probes the timeout settings I_3 and the link delay is 0ms, the number of timeout probing packets generated by DUDFTO is 51, while RAFA and LOFT are 2125, 1140, respectively. This is because the number of timeout probing packets generated by DUDFTO is only affected by the execution times of Feedback-based binary search step and Remainder binary search step and the timeout settings. For example, assuming that the number of timeout probing packets generated by Fast increment step is *C*, and the execution times of Feedback-based binary search step and Remainder binary search step are *N* and *M*, the number of timeout probing packets are about $2N + M \times (T_{hard}/T_{idle} + 1) + C$ for DUDFTO. For LOFT, its step in probing hard-timeout is 0.5 and *n* packets will be concurrently sent, so the number of timeout probing packets is bigger than DUDFTO. For RAFA, the steps used in probing hard-timeout and idle-timeout are 0.5 and 0.3, which will generate more timeout probing packets than DUDFTO. In addition, it needs to repeat at least 5 times to probe both hard and idle-timeouts. Therefore, the number of timeout probing packets generated by RAFA is bigger than LOFT and DUDFTO.

E. Comparison of Number of Packet_in and Flow_mod Messages

In this subsection, we compare the number of packet_in and flow_mod messages generated by DUDFTO and LOFT

TABLE VI
THE COMPARISON OF THE NUMBER OF TIMEOUT PROBING PACKETS UNDER I_1, I_2, I_3, I_4, I_5, I_6, I_7, AND I_8

ALG \ Settings / Delay		I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
0ms	DUDFTO	**31**	**45**	**51**	**53**	**62**	**73**	**90**	**145**
	RAFA	220	490	480	1470	2125	2984	3625	6758
	LOFT	125	320	530	780	1140	2350	4148	5855
100ms	DUDFTO	**31**	**45**	**51**	**53**	**62**	**73**	**90**	**145**
	RAFA	155	220	405	526	762	1447	1479	2828
	LOFT	90	195	330	430	610	1130	1850	2695
200ms	DUDFTO	**31**	**45**	**51**	**53**	**62**	**73**	**90**	**145**
	RAFA	105	165	305	395	570	1037	1715	2510
	LOFT	75	145	255	335	460	865	1390	2045



(a) I_5 (b) I_6 (c) I_7 (d) I_8

Fig. 8. The number of packet_in comparison under I_5, I_6, I_7, and I_8 (only idle-timeout is set).

during the attack. These evaluations are carried out using two kinds of timeout settings: 1) I_5, I_6, I_7, I_8 with only idle-timeout is set, and 2) I_5, I_6, I_7, I_8 with both idle-timeout and hard-timeout are set. In addition, to evaluate the attack method under varied timeout configurations, we also compare the number of packet_in and flow_mod messages under I_4, where the hard-timeout is not an integer multiple of the idle-timeout.

As shown in Fig. 8,[1] DUDFTO is compared with LOFT for the number of packet_in messages generated during the attack. We can find that DUDFTO is always superior to LOFT under all timeout settings. For example, among the four timeout settings, the numbers of packet_in messages generated by DUDFTO are 2069, 2069, 2055, 2065, respectively. And the numbers of packet_in messages generated by LOFT are 2129, 2145, 2157, 2142, respectively. More precisely, as shown in Fig. 8(c), the number of packet_in messages generated by DUDFTO is 102 (4.7%), which is less than that of LOFT.

In Fig. 10, when both the hard-timeout and idle-timeout are set, DUDFTO and LOFT are compared using the number of packet_in messages generated during the attack. We can still find that DUDFTO is always superior to LOFT under the different timeout settings. Due to the presence of hard-timeout, both DUDFTO and LOFT generate more packet_in messages than the timeout settings I_5, I_6, I_7, and I_8 with only idle-timeout is set. However, from Fig. 10, it can be seen that the number of packet_in messages generated by DUDFTO is always less than LOFT. Among the four timeout policies, the numbers of packet_in messages generated by DUDFTO are 9682, 5734, 3952, 3242, respectively, while the numbers of packet_in messages generated by LOFT are 10085, 5977, 4035, 3473, respectively. Hence, the numbers of packet_in messages generated by DUDFTO are 403 (4.0%), 243 (4.0%), 85 (2.1%), and 231 (6.7%) less than LOFT.

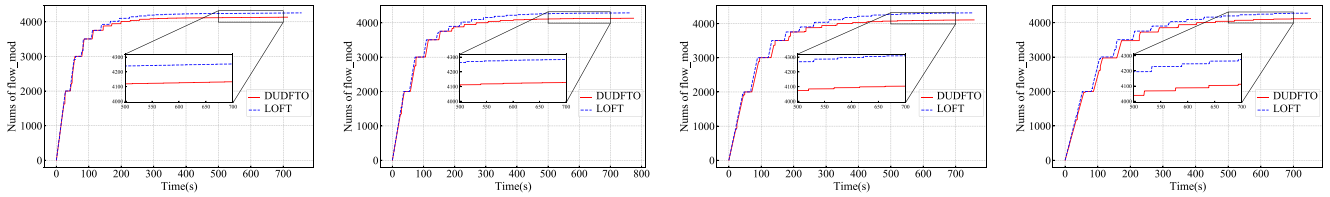
Figs. 9 and 11 show the number of flow_mod messages generated by DUDFTO and LOFT under the timeout settings I_5, I_6, I_7, and I_8 with only the idle-timeout is set or both hard-timeout and idle-timeout are set. In Fig. 9, we can see that the number of flow_mod messages generated by DUDFTO is less than LOFT. In the four timeout settings, the numbers of flow_mod messages generated by DUDFTO are 4134, 4132, 4108, 4120, respectively, while those of LOFT are 4258, 4288, 4314, 4284, respectively. The numbers of flow_mod messages generated by DUDFTO are 124 (2.9%), 156 (3.6%), 206 (4.8%), and 164 (3.8%), which are less than LOFT, respectively. In Fig. 11, we can obtain the same conclusion. For example, the numbers of flow_mod messages generated by DUDFTO are 19364, 11468, 7904, 6480, respectively. For LOFT, the numbers are 20170, 11948, 8070, 6944, respectively. Hence, the number of flow_mod packets generated by DUDFTO is 806 (4.0%), 480 (4.0%), 166 (2.1%), and 464 (6.7%). These numbers are all less than LOFT.

Fig. 12 shows the number of packet_in and flow_m-od messages generated by DUDFTO and LOFT under the timeout settings I_4. As we can see, when the hard-timeout is not an integer multiple of the idle-timeout, DUDFTO still generates less packet_in and flow_mod messages than LOFT. At the 800th second, the numbers of packet_in and flow_mod messages generated by DUDFTO are 12230 and 24460 respectively, while the numbers of packet_in and flow_mod messages generated by LOFT are 14007 and 28014. The numbers of packet_in and flow_mod generated by DUDFTO are 1777(14.5%) and 3554(14.5%).

F. Comparison of Attack Rate Distribution

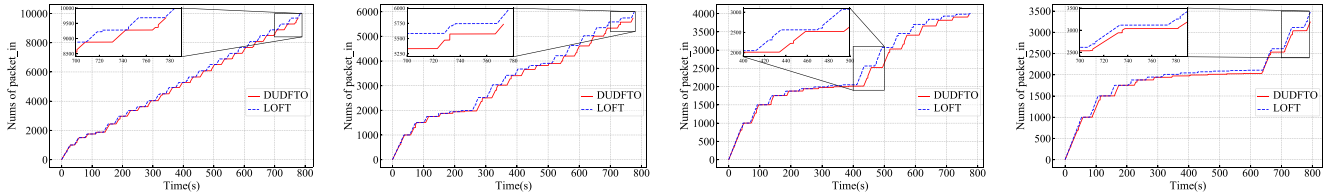
In this subsection, the rate distribution of packet_in and flow_mod messages of DUDFTO and LOFT are further compared for different timeout settings: 1) I_5, I_6, I_7, I_8

[1]In order to reduce the influence of these figures on the paper layout, we have saved the enlarged version of the figures in the attachment. Please refer to the attachment [37] for the enlarged figures of Figs. 8 to 17.



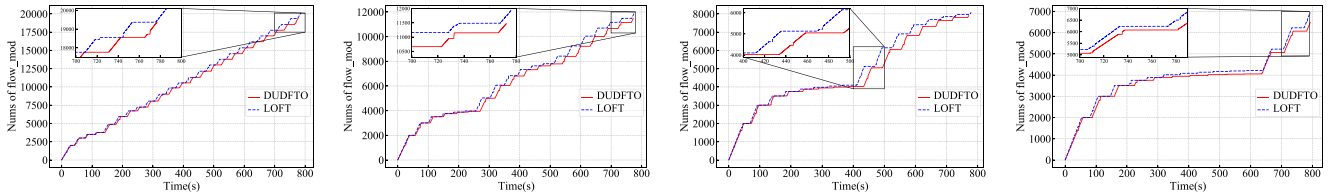
(a) I_5 (b) I_6 (c) I_7 (d) I_8

Fig. 9. The number of flow_mod comparison under I_5, I_6, I_7, and I_8 (only idle-timeout is set).



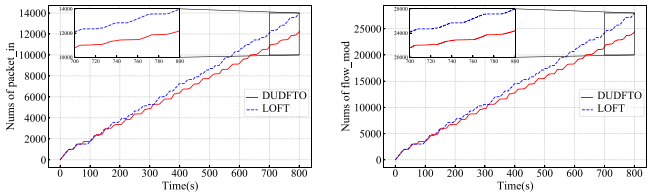
(a) I_5 (b) I_6 (c) I_7 (d) I_8

Fig. 10. The number of packet_in comparison under I_5, I_6, I_7, and I_8 (idle-timeout and hard-timeout are set).



(a) I_5 (b) I_6 (c) I_7 (d) I_8

Fig. 11. The number of flow_mod comparison under I_5, I_6, I_7, and I_8 (idle-timeout and hard-timeout are set).



(a) The number of packet_in (b) The number of flow_mod

Fig. 12. The number of packet_in and flow_mod comparison under I_4 (idle-timeout and hard-timeout are set).

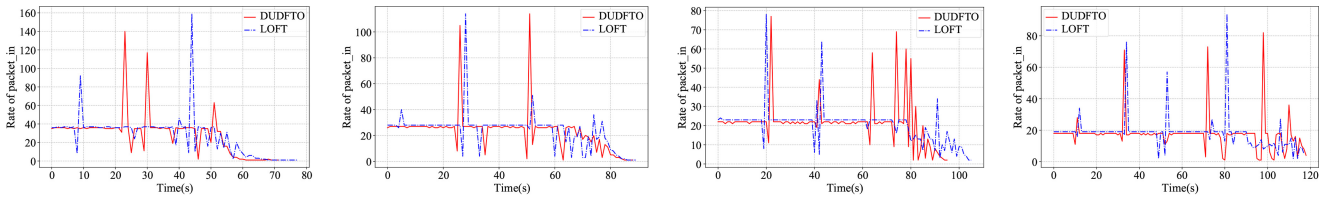
with only idle-timeout is set, and 2) I_5, I_6, I_7, I_8 with both idle-timeout and hard-timeout are set. Meanwhile, to evaluate the attack method under varied timeout configurations, we also compare the rate distribution of packet_in and flow_mod message under I_4.

As can be seen in Fig. 13, DUDFTO presents more fluctuations than LOFT for the same timeout settings. Meanwhile, the maximum packet_in messages rate of DUDFTO is smaller than that of LOFT. For example, in Fig. 13(c), the packet_in messages rate of LOFT is 23pps during 22 to 40s, while the rate of DUDFTO is 21pps to 22pps. And we can observe that the maximum packet_in messages rate of DUDFTO is 140pps and LOFT is 159pps in Fig. 13(a).

In Fig. 14, we observe that the packet_in message rate distribution of DUDFTO and LOFT are a little difference during the attack. The reason is that the existing of hard-timeout causes the flow entries to expire, so the switch continuously re-sends new packet_in messages, which reinforces the fluctuation of rate. However, it can still be observed in Fig. 14 that the maximum packet_in message rate of DUDFTO is smaller than LOFT. For example, the maximum packet_in message rate of DUDFTO are 156pps, 126pps, 91pps, and 75pps, while the maximum packet_in message rates of LOFT are 176pps, 118pps, 99pps, and 83pps.

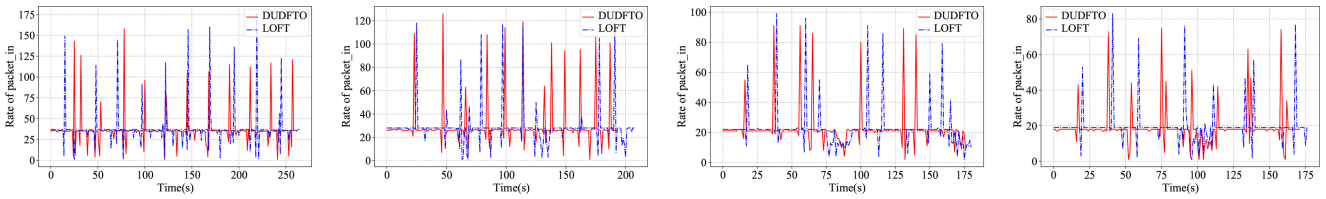
Furthermore, the rate distribution of flow_mod messages of DUDFTO and LOFT is compared, as depicted in Figs. 15 and 16. In Fig. 15, the flow_mod message rate fluctuation and maximum rate of DUDFTO are better than LOFT with only idle-timeout is set. In Fig. 16, the maximum flow_mod message rate of DUDFTO is less than LOFT.

Fig. 17 show the comparison of the rate distribution of packet_in and flow_mod messages of DUDFTO and LOFT under the timeout settings I_4. We can observe that DUDFTO shows more violent fluctuations, while LOFT is relatively flat. This is because LOFT maintains a fixed rate of packet sending, while DUDFTO dynamically adjusts the packet sending rate. At the same time, we can also find that maximum packet_in and flow_mod message rate of DUDFTO is larger than LOFT. It is most likely caused by DUDFTO increasing the packet sending rate of the remainder part. But as Fig. 12 and show, DUDFTO generates fewer packet_in and flow_mod messages than LOFT, even though its maximum packet_in and flow_mod message rate has been increased. Therefore, a



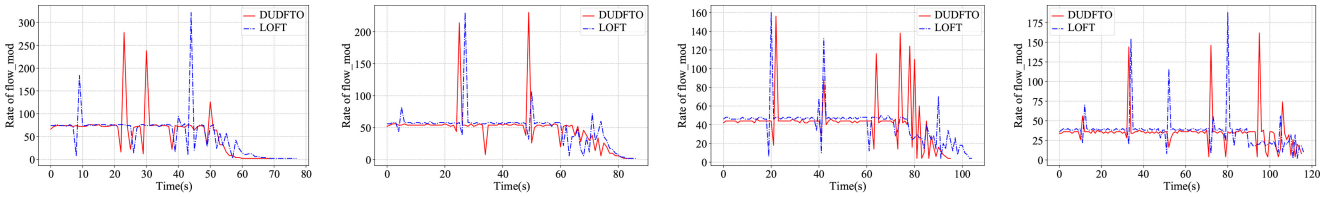
(a) I_5 (b) I_6 (c) I_7 (d) I_8

Fig. 13. The rate of packet_in messages comparison under I_5, I_6, I_7, and I_8 (only idle-timeout is set).



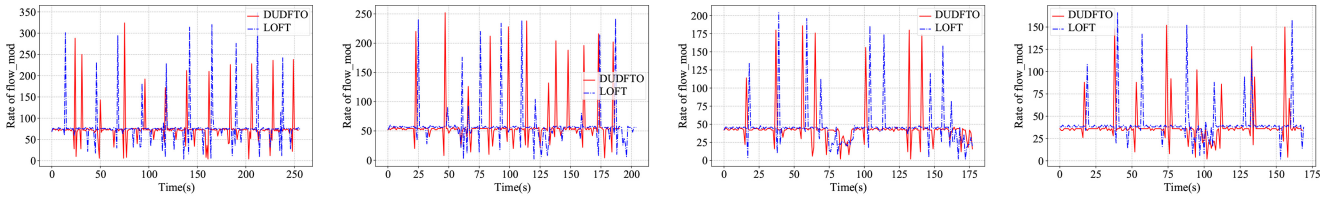
(a) I_5 (b) I_6 (c) I_7 (d) I_8

Fig. 14. The rate of packet_in messages comparison under I_5, I_6, I_7, and I_8 (idle-timeout and hard-timeout are set).



(a) I_5 (b) I_6 (c) I_7 (d) I_8

Fig. 15. The rate of flow_mod messages comparison under I_5, I_6, I_7, and I_8 (only idle-timeout is set).



(a) I_5 (b) I_6 (c) I_7 (d) I_8

Fig. 16. The rate of flow_mod messages comparison under I_5, I_6, I_7, and I_8 (idle-timeout and hard-timeout are set).

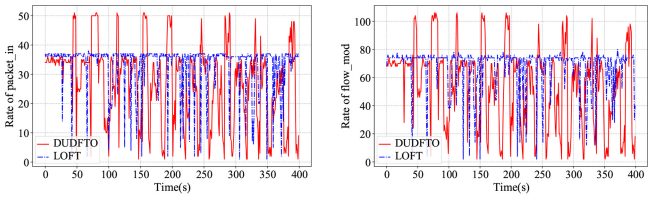
partial increase in the packet_in and flow_mod message rate is worthwhile, the stronger fluctuation also proves that DUDFTO is more dynamic.

G. Comparison of the Number of Malicious Flow Entries and Detected Flow Entries

In this subsection, we compare the number of malicious flow entries and detected flow entries of DUDFTO and LOFT when FTMaster is running. $LOFT\_flow$ indicates the malicious flow entries generated by LOFT. $DUDFTO\_flow$ indicates the malicious flow entries generated by DUDFTO. $drop\_l$ and $drop\_d$ indicate the flow entries installed by LOFT and DUDFTO are detected and deleted by the intrusion detection method.

Fig. 18 shows the number of malicious flow entries and detected flow entries of DUDFTO and LOFT, respectively. We can see that the number of malicious flow entries generated by DUDFTO and LOFT gradually increases from the 50th second. At 70th seconds, we can observe that the malicious flow entries $LOFT\_flow$ reaches its maximum, while the attack flow entries $DUDFTO\_flow$ continues to increase. However, with the operation of FTMaster, we can see that the number of malicious flow entries $LOFT\_flow$ gradually decreases, while the number of malicious flow entries $DUDFTO\_flow$ is not affected. Meanwhile, we can also observe that the detected flow entries $drop\_l$ of LOFT starts to increase, while the detected flow entries $drop\_d$ of DUDFTO only increases a little. Eventually, the flow table overflow attack detection method has detected 200 $drop\_l$ flow entries for LOFT and only 10 $drop\_d$ flow entries for DUDFTO.

From the above results, we can conclude that DUDFTO is more stealthy than LOFT. This result may be caused by the following phenomena. In FTMaster, it determines whether the flow entries are attack flow entries by extracting six features,



(a) The rate of packet_in (b) The rate of flow_mod

Fig. 17. The rate of packet_in and flow_mod messages comparison under I_4 (idle-timeout and hard-timeout are set).

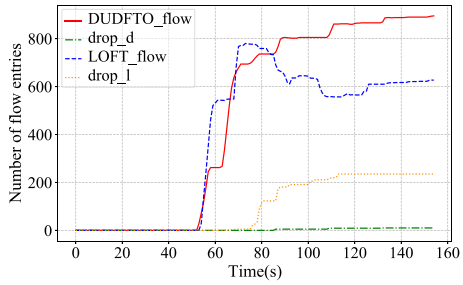


Fig. 18. The number of malicious flow entries and detected flow entries comparison.

including the duration time (DT), the number of bytes (NB), the number of packets (NP), the mean packet size (MPS), the mean packet interarrival time (MPIT), and the mean transmission speed (MTS). Since the attack rate of LOFT is fixed, when FTMaster calculates MPIT and MTS for the flow entries generated by LOFT, the MPIT and MTS corresponding to each flow table entry will remain consistent. For FTMaster, this is an obvious feature. Therefore, FTMaster can easily detect LOFT. In contrast, the attack rate of DUDFTO varies based on a uniform distribution. When FTMaster calculates MPIT and MTS for the flow entries generated by DUDFTO, the MPIT and MTS corresponding to each flow table entry will change with the variation in attack rate. Hence, it may be difficult for FTMaster to detect the malicious flow entries generated by DUDFTO.

H. The Effectiveness of the Link Failure Verification Method

In this subsection, we verify the effectiveness of DUDFTO's link failure verification method. We simulate different link failure scenarios: (1) a failure occurring every 2 seconds, (2) a failure occurring every 60 seconds. Additionally, we select three timeout settings—I_1, I_4, and I_8—to represent small, medium, and large timeout scenarios. Finally, we compare the timeout probing performance of DUDFTO, LOFT, and RAFA. In this experiment, we denote DUDFTO without link failure verification as DUDFTO-NoLFV and DUDFTO with link failure verification as DUDFTO-LFV.

As shown in Table VII, DUDFTO-LFV achieves the lowest relative error in timeout probing in Scenario (2), while it terminates probing in Scenario (1). For instance, under Scenario (2) and timeout setting I_1, DUDFTO-LFV exhibits a hard timeout probing relative error of 1.3%, whereas DUDFTO-NoLFV, LOFT, and RAFA report errors of 3.8%, 6.8%, and 8.0%, respectively. In Scenario (1) and timeout setting I_4,

TABLE VII
THE COMPARISON OF THE PROBING RELATIVE ERROR UNDER I_1, I_4, I_8 AND DIFFERENT LINK FAILURE SCENARIOS

Link Failure	Method	I_1		I_4		I_8	
		T_{idle}	T_{hard}	T_{idle}	T_{hard}	T_{idle}	T_{hard}
Scenario (1)	RAFA	48.0% (2.60)	89.6% (1.04)	91.3% (2.60)	94.7% (4.28)	96.2% (2.30)	99.8% (1.04)
	LOFT	100% (0)	78.3% (2.17)	100% (0)	97.3% (2.17)	94.1% (3.55)	99.1% (5.11)
	DUDFTO-NoLFV	100% (0)	54.7% (4.53)	100% (0)	63.0% (29.63)	100% (0)	90.0% (59.94)
	DUDFTO-LFV	**N/A**	**N/A**	**N/A**	**N/A**	**N/A**	**N/A**
Scenario (2)	RAFA	6.0% (5.3)	8.0% (10.77)	1.7% (30.5)	74.5% (20.4)	34.7% (39.16)	93.9% (36.6)
	LOFT	5.0% (4.75)	6.8% (10.68)	13.3% (26.02)	33.1% (53.52)	41.3% (35.24)	93.6% (38.29)
	DUDFTO-NoLFV	3.2% (4.84)	3.8% (9.62)	0.7% (29.79)	61.0% (31.17)	0.6% (59.63)	100% (0)
	DUDFTO-LFV	**0.6%** **(4.97)**	**1.3%** **(9.87)**	**0.43%** **(29.87)**	**0.125%** **(79.90)**	**0.042%** **(59.75)**	**0.06%** **(599.64)**

DUDFTO-LFV terminates probing, while the hard timeout relative errors for DUDFTO-NoLFV, LOFT, and RAFA are 63.0%, 97.3%, and 94.7%, respectively. This discrepancy arises because DUDFTO-NoLFV, LOFT, and RAFA lack link failure verification capabilities. When the link failure causes the RTT of a probe packet to increase or become unmeasurable, these algorithms mistakenly attribute the delay to an idle or hard timeout, leading to inaccurate timeout estimates.

Table VII further reveals that DUDFTO-NoLFV possesses limited link failure checking capabilities. For example, in Scenario (1) and timeout setting I_1, DUDFTO-NoLFV probes a hard timeout of 4.53s and an idle timeout of 0s. This proves that DUDFTO-NoLFV misclassifies idle timeout as hard timeout in the environment with frequent link failure, highlighting that DUDFTO's feedback-based probing algorithm provides limited resilience against link failure.

By introducing link failure verification method, DUDFTO-LFV mitigates the negative impact of link failures on timeout probing results, thereby improving the effectiveness of timeout probing. DUDFTO-LFV does not return results in Scenario (1) as expected, as performing link failure checking indefinitely would significantly extend timeout probing duration. In real-world network environments [35], link failure usually occurs at relatively long intervals, so two link failure verifications are sufficient. The results from Scenario (2) further confirm that this mechanism effectively prevents probing errors caused by link failures.

I. Results Analysis and Discussion

From the above evaluation results, it can be seen that DUDFTO outperforms LOFT and RAFA. For example, when probing match fields, DUDFTO can probe all fields, while LOFT cannot probe the $ipv4\_src$, and $ipv6\_src$. When probing timeout values, DUDFTO is more accurate than LOFT and RAFA. More importantly, DUDFTO can successfully probe all timeout settings under challenging link delay and link failure scenarios, while LOFT and RAFA may be interfered by high RTT and link failures. Meanwhile, the number of timeout probing packets generated by DUFTO is less than RAFA and LOFT. During the attack phase, DUDFTO generates less packet_in and flow_mod messages than LOFT, and the detected probability of DUDFTO is less than LOFT, making it more stealthy.

Based on the comparison results of the above experiments, we can conclude that DUDFTO is superior to LOFT and RAFA. This phenomenon may be caused by the fact that DUDFTO designs a down-to-up timeout probing algorithm. DUDFTO probes idle-timeout and hard-timeout at the same time, exploits the difference in the RTT, and employs the link failure verification method to judge the current probed timeout value, thus avoiding the interference of link failures and the interference problem of hard-timeout and idle-timeout. In addition, it improves the timeout probing accuracy in high-RTT by adding the RTT to the probed timeout value. In other words, DUDFTO has ability to probe the timeout settings more precisely. Moreover, due to the fact that the attack rate of DUDFTO is calculated based on a uniform distribution, DUDFTO can exhibit a more diverse set of features when countering the flow table overflow attack detection methods. Therefore, DUDFTO can conduct attacks in a more covert manner.

V. Conclusion

In this paper, we propose DUDFTO, a down-to-up timeout probing and dynamic flow table overflow attack. Firstly, DUDFTO infers the match fields by modifying the fields of the probing packet and measuring the one-sided transmission delay of the probing packet. Then, the down-to-up feedback-based timeout probing algorithm is used to probe the timeout settings. After that, DUDFTO uses the probed network configurations to design attack strategies against different timeout settings. Finally, DUDFTO probes the flow table state to stop sending new attack packets.

However, DUDFTO still has some limitations. When probing timeout settings, DUDFTO will spend more time than RAFA and LOFT. In future work, we aim to further research how to reduce the probing time of DUDFTO. We hope that our research will contribute to the further development of SDN security research.

References

[1] Y. Cui et al., "Towards DDoS detection mechanisms in software-defined networking," *J. Netw. Comput. Appl.*, vol. 190, Sep. 2021, Art. no. 103156.

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

[3] A. Vishnoi, R. Poddar, V. Mann, and S. Bhattacharya, "Effective switch memory management in OpenFlow networks," *Proc. 8th ACM Int. Conf. Distrib. Event-Based Syst. (DEBS)*, 2014, pp. 177–188.

[4] "Open Networking Foundation: 'OpenFlow switch specification1.3.0'," Jun. 2012. [Online]. Available: http://www.cs.yale.edu/homes/yu-minlan/teach/csci599-fall12/papers/openflow-spec-v1.3.0.pdf

[5] M. Xiao, Y. Cui, Q. Qian, and G. Shen, "KIND: A novel image-mutual-information-based decision fusion method for saturation attack detection in SD-IoT," *IEEE Internet Things J.*, pp. vol. 9, no. 23, pp. 23750–23771, Dec. 2022.

[6] L. Ran, Y. Cui, C. Guo, Q. Qian, G. Shen, and H. Xing, "Defending saturation attacks on SDN controller: A confusable instance analysis-based algorithm," *Comput. Netw.*, vol. 213, Aug. 2022, Art. no. 109098.

[7] T. Xie, T. He, P. McDaniel, and N. Nambiar, "Attack resilience of cache replacement policies," *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2021, pp. 1–10.

[8] D. Tang, L. Tang, W. Shi, S. Zhan, and Q. Yang, "MF-CNN: A new approach for LDoS attack detection based on multi-feature fusion and CNN," *Mobile Netw. Appl.*, vol. 26, no. 4, pp. 1705–1722, 2021.

[9] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (SIGCOMM)*, 2013, pp. 165–166.

[10] J. Sonchack, A. Dubey, A. J. Aviv, J. M. Smith, and E. Keller, "Timing-based reconnaissance and defense in software-defined networks," in *Proc. 32nd Annu. Conf. Comput. Security Appl. (ACSAC)*, 2016, pp. 89–100.

[11] J. Sonchack, A. J. Aviv, and E. Keller, "Timing SDN control planes to infer network configurations," in *Proc. ACM Int. Workshop Security Softw. Defined Netw. Netw. Funct. Virtualization*, 2016, pp. 19–22.

[12] P. Lin, P. Li, and V. L. Nguyen, "Inferring OpenFlow rules by active probing in software-defined networks," in *Proc. 19th Int. Conf. Adv. Commun. Technol. (ICACT)*, 2017, pp. 415–420.

[13] A. Azzouni, O. Braham, T. M. T. Nguyen, G. Pujolle, and R. Boutaba, "Fingerprinting OpenFlow controllers: The first step to attack an SDN control plane," *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2016, pp. 1–6.

[14] B. Ahmed, N. Ahmed, A. W. Malik, M. Jafri, and T. Hafeez, "Fingerprinting SDN policy parameters: An empirical study," *IEEE Access*, vol. 8, pp. 142379–142392, 2020.

[15] B. Yiğit, G. Gür, F. Alagöz, and B. Tellenbach, "Network fingerprinting via timing attacks and defense in software defined networks," *Comput. Netw.*, vol. 232, Aug. 2023, Art. no. 109850.

[16] 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," *Security Commun. Netw.*, vol. 2018, Jan. 2018, Art. no. 4760632.

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

[18] Y. Shen, C. Wu, D. Kong, and Q. Cheng, "Flow table saturation attack against dynamic timeout mechanisms in SDN," *Appl. Sci.*, vol. 13, no. 12, p. 7210, 2023.

[19] 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.

[20] M. Zhang, G. Li, L. Xu, J. Bi, G. Gu, and J. Bai, "Control plane reflection attacks in SDNs: New attacks and countermeasures," *Proc. 21st Int. Symp. Res. Attacks, Intrusions, Defenses*, 2018, pp. 161–183.

[21] T. A. Pascoal, T. E. Fonseca, and V. Nigam, "Slow denial-of-service attacks on software defined networks," *Comput. Netw.*, vol. 173, May 2020, Art. no. 107223.

[22] B. Isyaku, M. B. Kamat, K. B. Abu Bakar, M. S. M. Zahid, and F. A. Ghaleb, "IHTA: Dynamic idle-hard timeout allocation algorithm based OpenFlow switch," in *Proc. IEEE 10th Symp. Comput. Appl. Ind. Electron. (ISCAIE)*, 2020, pp. 170–175.

[23] 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. Int. Conf. Vis. Towards Emerg. Trends Commun. Netw. (ViTECoN)*, 2019, pp. 1–6.

[24] X. Li, and Y. Huang, "A flow table with two-stage timeout mechanism for SDN switches," *Proc. IEEE 21st Int. Conf. High Perform. Comput. Commun. IEEE 17th Int. Conf. Smart City IEEE 5th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, 2019, pp. 1804–1809.

[25] B. Sooden and M. R. Abbasi, "A dynamic hybrid timeout method to secure flow tables against DDoS attacks in SDN," in *Proc. 1st Int. Conf. Secure Cyber Comput. Commun. (ICSCCC)*, 2018, pp. 29–34.

[26] "BigFlows," 2024. [Online]. Available: https://tcpreplay.appneta.com/wiki/captures.html

[27] N. N. Dao, J. Park, M. Park, and S. Cho, "A feasible method to combat against DDoS attack in SDN network," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, 2015, pp. 309–311.

[28] 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.

[29] N. M. Yungaicela-Naula, C. Vargas-Rosales, J. A. Pérez-Díaz, and D. F. Carrera, "A flexible SDN-based framework for slow-rate DDoS attack mitigation by using deep reinforcement learning," *J. Netw. Comput. Appl.*, vol. 205, Sep. 2022, Art. no. 103444.

[30] 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, Apr. 2023, Art. no. 103597.

[31] D. Tang, S. Wang, B. Liu, W. Jin, and J. Zhang, "GASF-IPP: Detection and mitigation of LDoS attack in SDN," *IEEE Trans. Services Comput.*, vol. 16, no. 5, pp. 3373–3384, Sep./Oct. 2023.

[32] W. Dongbin et al., "Preventing flow table overflow against denial of service attack in software defined network," *J. Commun.*, vol. 44, no. 1, p. 2, 2023.
[33] I. I. Awan, N. Shah, M. Imran, and N. Saeed, "An improved mechanism for flow rule installation in-band SDN," *J. Syst. Archit.*, vol. 96, pp. 1–19, Jun. 2019.
[34] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford, "A NICE way to test OpenFlow applications," in *Proc. 9th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2012, pp. 127–140.
[35] M. Ibrar, L. Wang, G. M. Muntean, A. Akbar, N. Shah, and K. R. Malik, "PrePass-Flow: A machine learning based technique to minimize ACL policy violation due to links failure in hybrid SDN," *Comput. Netw.*, vol. 184, Jan. 2021, Art. no. 107706.
[36] D. Tang, Z. Zheng, K. Li, C. Yin, W. Liang, and J. Zhang, "FTOP: An efficient flow table overflow preventing system for switches in SDN," *IEEE Trans. Netw. Sci. Eng.*, vol. 11, no. 3, pp. 2524–2536, May/Jun. 2024.
[37] "Attachment of DUDFTO." 2024. [Online]. Available: https://github.com/ljs12324/DUDFTO

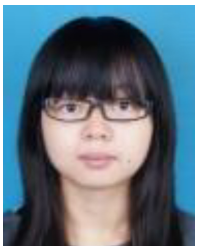
Jiasong Li received the bachelor's degree in information security major from Guizhou University, where he is currently pursuing the Graduate degree. His research interests include the fields of data security and network security, especially in network attack in SDN.

Yunhe Cui received the Ph.D. degree from Southwest Jiaotong University, Chengdu, Sichuan, China. He is currently an Associate Professor with the College of Computer Science and Technology, Guizhou University, China. He has authored and co-authored over 20 peer-reviewed journal and conference papers. His research interests include network intrusion detection and prevention, software-defined networking, network telemetry, traffic engineering, edge computing, and data centers.

Yi Chen (Member, IEEE) received the Ph.D. degree from Southwest Jiaotong University, Chengdu, China, in 2022. He is currently a Lecturer with Guizhou University, Guiyang, Guizhou, China. His research interests include multimedia information security and artificial intelligence security.

Guowei Shen received the Ph.D. degree from Harbin Engineering University. He is currently a Professor with Guizhou University. His main research interests include big data, computer networks, and cyber security.

Chun Guo received the Ph.D. degree in information security from the Beijing University of Posts and Telecommunications in July 2014. He is currently a Professor with the College of Computer Science and Technology, Guizhou University, China. His research interests include data mining, intrusion detection, and malware detection.



Qing Qian (Member, IEEE) received the Ph.D. degree from Southwest Jiaotong University, Chengdu, China, in 2018. She is an Associate Professor with the School of Information, Guizhou University of Finance and Economics. Her research interests include the security of multimedia information, cryptography, and network security.