International Journal of Information Security (2024) 23:3465–3479
https://doi.org/10.1007/s10207-024-00897-x

REGULAR CONTRIBUTION

Unmasking SDN flow table saturation: fingerprinting, attacks and defenses

Beytüllah Yiğit[1] · Gürkan Gür[2] · Bernhard Tellenbach[3] · Fatih Alagöz[1]

Published online: 4 August 2024
© The Author(s) 2024

Abstract
Software-Defined Networking stands as a pivotal technology in attaining the essential levels of flexibility and scalability demanded by pervasive and high-performance network infrastructure required for digital connected services. Nonetheless, its disaggregated and layered architecture makes it open to the time-based fingerprinting attacks. Besides, limited flow table capacity of the switches alleviates table saturation attacks. In this paper, an automated attacker tool called *TASOS* is proposed to infer flow table utilization rate, size and replacement algorithm. With this set of information, the attacker can conduct intelligent saturation attacks. Furthermore, a lightweight defense mechanism (*LIDISA*) for proactively deleting flow rules is described. A comprehensive simulation setup with different network conditions shows that the proposed techniques achieve superior success rate in diverse settings.

Keywords SDN security · Fingerprinting · Reconnaissance · Denial of service attack · Saturation attack

1 Introduction

Next-generation networks are envisioned to cultivate the future of a universally connected, data-intensive, and intelligent society. Serving this pivotal goal, Soft-ware-Defined Networking (SDN) integrates programmability as an inherent attribute of networks, ushering in automation and enhanced flexibility in network management by decoupling the control plane from the data plane. In contrast to traditional networks, SDN empowers a central controller to dynamically program the data plane and execute applications and network policies with a precise granularity of information, enabling the efficient steering and control of network traffic. Nonetheless, the improved flexibility and control achieved via SDN are accompanied by new security vulnerabilities, such as control plane saturation attacks [1]. Thus, the realization of SDN's full potential is hampered by security concerns, challenging its widespread practical implementation. Consequently, the security of SDN constitutes a dynamic and multifaceted domain of research [4, 7, 9, 13, 18, 33].

One of the primary threat vectors within the data plane is the flow table of SDN switches. Due to constraints such as high power consumption and cost, commercial SDN/OpenFlow switches possess restricted flow table capacities, typically ranging from hundreds to thousands [25]. This capacity is often inadequate for managing the millions of flows commonly encountered in enterprise and data center networks [40]. It is evident that future switches are expected to support larger flow tables. However, the efficiency of flow table management remains a significant concern. This is particularly important because TCAMs (Ternary Content-Addressable Memory) consume much more power and are costly, e.g., reported as approximately 100 times more power-hungry and expensive compared to conventional RAM in [11].

In SDN, given that switches are relatively simplistic devices, they rely on communication with the controller to handle packets that do not align with any existing flow rule. This communication method can be exploited by attackers in various manners. A notable instance of such an attack involves targeting an SDN switch with the aim of overwhelming its flow table by generating traffic that leads to numerous new flow table entries. Additionally, if the attacker can ascertain the TCAM size of the switch and flow table

✉ Beytüllah Yiğit
beytullah.yigit@std.bogazici.edu.tr

[1] Department of Computer Engineering, Bogazici University, 34342 Istanbul, Turkey

[2] Zurich University of Applied Sciences ZHAW, 8401 Winterthur, Switzerland

[3] armasuisse S+T, Thun, Switzerland

replacement algorithm, they can fine-tune the flow generation rate to overwhelm the switch with a minimal number of packets efficiently. This restricted packet generation benefits the attacker in terms of performance and reduces the risk of detection before the attack accomplishes its objective.

To assess whether a network is vulnerable to these attacks, an attacker needs to establish whether it operates as an SDN and gather relevant information. One method for identifying an SDN is to exploit the concept of First Packet Processing Time (FPPT), which is defined as the variation in Round-Trip Time (RTT) between the first and second packets of a novel unmatched flow. These techniques are commonly referred to as SDN fingerprinting and are often followed by methods aimed at determining pertinent SDN characteristics, such as flow rule timeout values, flow match fields, active flow rules and flow table capacity [2, 5, 8, 10, 14–16, 21, 25, 26, 35, 39, 40].

Following SDN fingerprinting, attackers can transition from the reconnaissance phase to the subsequent stages of the cyber kill chain and execute attacks that, in the most dire scenarios, have the potential to result in the complete shutdown of the entire network by utilizing the fingerprinting information. Nonetheless, it is essential to recognize that SDN fingerprinting research remains in its nascent stages with the integration of this technology into diverse network infrastructures and predominantly centers on the attacker's perspective, specifically on methods to extract increasingly reliable data from networks. Strategies for preventing attackers from gaining access to such information in the first place are seldom explored.

In our previous work [34], we inferred network type (SDN or traditional) and flow timeout values (hard and idle). Besides, a lightweight mechanism that adjusts timeout values to increase complexity and costs for attackers was proposed for defense. Likewise, our new research covers both offensive and defensive techniques regarding SDN fingerprinting, and their effectiveness is assessed through simulation-based experiments. The principal contributions of our work can be summarized as follows:

- A comprehensive and automated module for conducting flow table saturation attacks, named *Table Analyzer and Saturation Orchestrator for SDN (TASOS)*. This module effectively fingerprints flow table-related details, including size, utilization rate and replacement policy, enabling the execution of efficient saturation attacks using this information.
- Extensive simulations of *TASOS* demonstrate the high accuracy and adaptability of these proposed methods across various network conditions.
- Development of an agile and dynamic defense mechanism that enhances the complexity and expenses for attackers by adjusting flow table attributes in response to changes in network status.

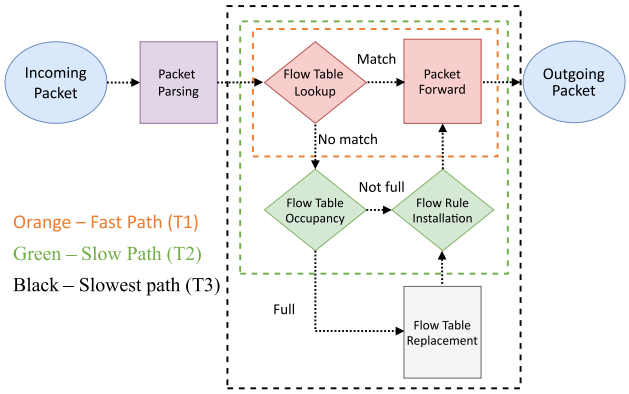


Fig. 1 SDN packet processing flowchart

The structure of this paper is as follows: We begin by presenting fundamental aspects of SDN relevant to this work in Sect. 2. Subsequently, we delve into a discussion of related work in Sect. 3. This is followed by an introduction to *TASOS* in Sect. 4 and our defense mechanism in Sect. 5. In Sect. 6, we detail the methodology of our extensive simulations and present the results, with a particular emphasis on the performance of *TASOS* and our defense mechanism. Section 7 addresses limitations and open questions, providing a critical perspective on our outcomes. Finally, in Sect. 8, we explore potential avenues for future research and conclude the paper.

2 SDN fundamentals

Figure 1 illustrates the packet processing logic within SDN. It delineates three distinct paths for a packet, demarcated by dashed rectangular regions in the diagram. Upon packet arrival in the data plane, the header information undergoes checks against the match fields of pre-existing flow rule entries. When a match is found, the action field of the corresponding entry is executed (*Fast Path*). In cases where no match is identified, the data plane triggers a `packet-in` message, which is subsequently processed by the controller. The controller then establishes a new flow rule to manage this novel flow (*Slow Path*). However, the switch must ascertain whether adequate space remains within the flow table for this new entry. In situations where the flow table is at full capacity, the controller or switch is compelled to undertake flow table replacement operations. This involves the removal of certain flow rules according to the table replacement algorithm, potentially leading to further performance degradation across the network (*Slowest Path*).

We will use the designations T_1, T_2 and T_3 to represent RTT required for the *Fast Path*, *Slow Path* and *Slowest Path*,

respectively. An attacker can dispatch an initial packet to generate a new flow, and then the corresponding RTT, T_2, can be calculated. Subsequently, a second packet is transmitted and its RTT, designated as T_1, is calculated. Following this, packets are dispatched with the intention of inducing the installation of a new flow for each of them, until the RTT for such a packet significantly surpasses T_2. This occurs when the flow table reaches full capacity, making the slowest path relevant. This specific RTT is indicated as T_3. For successful SDN fingerprinting, T_2 and T_3 should be significantly bigger than T_1 and T_2, respectively.

An essential aspect of the processing logic involves comprehending how and when flow rules are purged when the flow table is not at full capacity. OpenFlow [20], having established itself as the primary southbound protocol, incorporates two fundamental mechanisms for flow rule deletion in the majority of SDN environments: deletion orchestrated by the controller and deletion predicated on timeouts. OpenFlow also defines two distinct timeout values: hard and idle [20]. In the context of idle timeouts, flow rules are removed following a designated period of inactivity. Conversely, hard timeouts dictate the removal of flow rules after a specified period, irrespective of their activity status. These timeout values hold critical significance for attackers, as they can directly impact their fingerprinting efforts and the performance of their attacks.

When the flow table becomes full, newly arriving flows face two possibilities: they are either dropped, or an existing flow is evicted from the table to make space for the new request. However, removing an existing entry from the table introduces latency and places an additional workload on the controller. This initiates a sequence of message exchanges between the controller and the switch, ultimately driving the flow table operations. The flow replacement policy becomes relevant when the controller attempts to install a new flow on a switch with a full table. The switch responds with a `table_full` message and the controller can either choose to drop the new flow or install it by removing an earlier flow table entry.

OpenFlow currently offers three mechanisms for flow table management: flow timeouts, flow entry deletion and flow entry eviction, as specified in the OpenFlow specifications [20]. Switches are responsible for implementing timeouts and eviction, while the controller can delete flows by sending flow delete message. In OpenFlow 1.4, an "importance" field for the flow rules was introduced as an optional feature. This field is used in the eviction process and flow entries are evicted strictly in order of importance. Flow entries with lower importance will always be evicted before those with higher importance.

Two significant consequences can be deducted from the packet processing and flow rule logic described above. Firstly, an attacker can deduce valuable insights into the switch's state by comparing the RTT values of packets. This is due to the fact that each path taken by a packet involves distinct interactions and iterations. By analyzing the path information of a packet, which is contingent on flow rule timeouts, the attacker can gather data concerning the state and size of the flow table, the timeout configurations in use and even details about the flow rules themselves, including the match fields. Secondly, armed with this acquired information, an attacker could conduct more efficient and tailored attacks, exploiting these insights to optimize their malicious activities.

3 Related work

Fingerprinting encompasses the process of identifying and gathering information regarding a specific computer system or network, with the aim of discerning its unique characteristics and potential vulnerabilities. This involves the collection of various data points, including details on open ports, the operating system in use, software applications, network topology and configuration settings [29]. The information acquired through fingerprinting can serve multiple purposes, ranging from launching targeted attacks to enhancing security measures designed to safeguard against potential threats [32].

In this section, we offer a comprehensive overview of the literature concerning general SDN fingerprinting and strategies aimed at devising advanced fingerprinting techniques. We begin by providing an overview of general SDN fingerprinting works, followed by a more detailed examination of those specifically related to flow tables and saturation attacks.

Klöti et al. [15] stands as the inaugural theoretical work delving into the realm of SDN fingerprinting. The authors introduced the concept of employing flow rule timeout randomization as a potential defense mechanism. On the practical front, [26] is the first paper to demonstrate SDN fingerprinting by analyzing RTT discrepancies between packets. Additionally, in [8], the authors conducted an exploration into the influence of hop count and bandwidth on the accuracy of RTT-based fingerprinting. As a countermeasure against RTT-based fingerprinting, they suggest introducing a delay to a selection of packets from newly established inactive flows.

In [14], the authors address various challenges encountered when conducting SDN fingerprinting in real-world environments. These challenges include factors such as the presence of unknown background traffic, issues related to rule granularity and rule overlapping. Moreover, the authors introduced two defensive strategies: flow timeout randomization and traffic-aware mitigation. The latter recommendation entails the implementation of proactive rules for the ICMP

protocol and the application of rate limiting for the UDP protocol, with the aim of diminishing the attack surface.

The controller and SDN switches are also susceptible to saturation attacks. Early studies, such as [26] and [23], explore packet-in flooding attacks in which attackers aggressively generate a massive volume of `packet-in` messages to flood the SDN controller. High packet-in rates are a key characteristic of this attack type. AVANT-GUARD [27] introduces a connection migration module and actuating triggers module to extend the data plane functions, but it is applicable to TCP protocol only. Subsequently, FloodGuard [31] presents a protocol-in-dependent defense framework that pre-installs proactive flow rules to reduce table-miss packets and forwards those packets to additional data plane caches for mitigation. FloodDefender [23] takes a different approach by offloading table-miss packets to neighbor switches and employing a two-phase filtering strategy to filter out attack traffic. Lineswitch [3] extends OpenFlow with the goal of mitigating the same type of attack.

In [2], the authors endeavor to identify a broad spectrum of SDN policy parameters, encompassing attributes such as hard and idle timeouts, the match fields of flow rules and the controller's response to a `table_full` event (whether it rejects or deletes flow rules). Additionally, they leverage the common utilization of the fat-tree topology in SDN-deployed data centers to deduce certain information about the network's structure, including in-rack, in-pod and inter-pod groupings. As a defensive strategy, the paper proposes dynamic timeout assignment and the adjustment of the number of match fields in accordance with the load on the flow table. However, there is no performance evaluation available for the `table_full` event fingerprinting.

In [10], the state of the flow table, including its size and utilization rate, is deduced using peak and level-shift detection algorithms in conjunction with the T_1, T_2 and T_3 logic. For the replacement algorithm, only FIFO is considered. Notably, their algorithms are tailored to operate in scenarios where no background traffic is present and flow rules are permanent, which severely limits their practical applicability.

In [16, 40], the authors employ the T_1, T_2 and T_3 metrics to ascertain the size of the flow table and the number of active flow rules within it. Additionally, they leverage measurements to deduce whether the flow table replacement algorithm used is LRU or FIFO, achieving an 80% accuracy rate. Furthermore, the authors put forth two defensive strategies, namely routing aggregation and multilevel flow table architecture, although these solutions remain theoretical and lack practical implementation. It's worth noting that their timeout fingerprinting algorithms have been identified as having certain limitations, as pointed out in our previous work [34].

In [39], the authors identify the match fields of flow rules by changing packet headers. This information is then utilized to perform a packet-in flood. Additionally, they infer the policy for statistics collection enabling statistics-related attacks, which are out of scope of this paper. This allows the attacker to trigger statistics collection at their discretion. However, the paper does not provide performance tests to evaluate the accuracy of these methods. For defense, the authors propose the use of a multi-queue based structure in the switches, along with scheduling different downlink messages with specific types and priorities. Simultaneously, the controller monitors downlink message events and assigns varying priorities to different messages as a protective measure. This defense mechanism needs to modify the switch design to be integrated into a practical system.

In [35], an intelligent control plane saturation attack is devised, which incorporates a reconnaissance stage to gather information about the flow table size, utilization and the replacement policy (LRU or FIFO). The success ratio of the attack is reported as 90%. It's important to note that the attack design assumes that all the rules are stored in a single table. The reconnaissance stage is ideally carried out during periods of light background traffic, as heavy traffic significantly diminishes the accuracy of the reconnaissance. The algorithm is unable to precisely determine the flow table size and replacement policy, necessitating multiple iterations for improved accuracy.

In [25], a flow table saturation attack targeting dynamic timeouts is introduced. The attack is divided into three phases: inference, triggering and maintenance. Firstly, in the inference phase, the attacker deduces the type of timeout strategy being used (fixed, random, or dynamic) and identifies the potential timeout values for inserted flow rules. In the triggering phase, the attacker initiates flow rule installation. This is accomplished by employing multiple rounds of probes to induce the reinstallation of flow rules, compelling the controller to assign lengthy timeouts to them. This phase generates attack flows and contaminates the flow table with malicious flow rules. Finally, in the maintenance phase, the attacker crafts and dispatches packets with inter-arrival times falling within the idle timeout period, ensuring that the corresponding flow entries remain active indefinitely. Apparently, the timeout fingerprinting algorithm outlined does not account for hard timeouts when searching for idle timeouts. This oversight could potentially yield misleading results.

In [12], the authors combine the packet-in rate and the control channel occupation rate (CCOR) as inputs for Linear Discriminant Analysis (LDA) to detect saturation attacks. Subsequently, the controller assembles a white-list and installs migration flow rules as part of their defense strategy.

In [5], match fields, including their bit-masks and flow rule timeouts are inferred during the probing phase. To flood the flow tables based on the inferred network configurations, the lower bound of the attack rate is calculated. Each attack

packet is designed to trigger or maintain the consumption of one flow rule. However, it's worth noting that their timeout fingerprinting algorithm demands numerous iterations, resulting in a substantial number of probing packets and potentially requiring several hours to complete.

In [21], flow timeouts and flow table size are determined using algorithms from the literature. Subsequently, a slow TCAM exhaustion attack is executed based on the inferred parameters by compelling the installation of new forwarding rules and perpetually maintaining them active. Botnets are employed for flow installation. As a defense mechanism, it suggests randomly selecting rules to be dropped whenever the system is overloaded.

The flow replacement algorithm determines how the flow table manages entries when it becomes full. Therefore, knowledge of the flow table replacement algorithm is crucial for an effective saturation attack. That algorithm governs the internal state transition of an SDN network, making it an essential part of the inference model. Replacement fingerprinting has been explored in [2, 16, 35]. However, [2] only determines if a replacement algorithm exists or not, while [16, 35] focus on LRU and FIFO. Our algorithm covers four different algorithms: REJECT (rejecting new flows), LRU, FIFO and RANDOM. REJECT is important since default controller applications typically do not respond to `table_full` error messages, relying on automatic removal of flow entries through the timeout mechanism [2]. RANDOM is also popular, as it performs better than FIFO with considerably less overhead [30].

References [10, 16, 35] try to find flow table size and utilization. However, [16, 35] use fixed thresholds and none of them employ statistical methods to calculate thresholds based on response times. Additionally, [5] does not provide an algorithm for table size fingerprinting but mainly presents their measurement procedure. Our algorithm, *TASOS*, stands out by incorporating statistical methods in threshold calculation for a more adaptive approach.

In the saturation attack, it is imperative not only to deplete the TCAM memory of the SDN switch by compelling the installation of new flow rules but also to sustain their perpetuity, thereby obstructing new rules from being installed to cater to legitimate endpoints. Consequently, the inference of timeouts becomes essential. Furthermore, a mechanism must exist to ensure the steady activation of the inserted flow rules. The attacker sends packets with a meticulously calculated inter-arrival time, ensuring that the flow rules never expire due to timeouts. Another consideration is that the rate and duration of packet-in messages serve as common indicators for detecting such attacks, as observed in studies like [12, 31]. Hence, maintaining a low packet-in rate during the attack is crucial to elude detection. Notably, [10, 35, 39] do not incorporate flow timeouts into their analyses. In [16], attempts are made to flood the flow table within a single timeout duration, potentially revealing the probing attack. In *TASOS*, the attacker retains the ability to adjust the packet-in rate, allowing for a gradual flooding of the flow table with malicious flow rules and ultimately achieving saturation of the entire storage space, thereby circumventing the mentioned defenses.

Evidently, the dimensions of the flow table, encompassing its size and replacement algorithm, are of central importance in orchestrating efficient and sustained saturation attacks. Without precise knowledge of the flow table size, the attacker remains incapable of determining when to cease the influx of new flow rules. The choice of replacement algorithm is also pivotal, as it dictates the fate of the attacker's flows when the table reaches its capacity. For example, if FIFO is implemented as the replacement algorithm, the attacker simply needs to dispatch update packets without letting the flows expire. Conversely, in the case of LRU, the attacker must extend the update packet intervals, considering that the least recently used flows are evicted. Notably, [5, 21, 25, 39] overlook the consideration of replacement algorithms, while [25, 39] neglect to account for the flow table size.

When executing a saturation attack, the number of packets required is directly related to the cost of the attack, which is why an attacker is typically interested in keeping it as low as possible. During the reconnaissance phase, *TASOS* systematically gathers flow table-related data. Subsequently, guided by this intelligence and the attacker's chosen packet-in rate, the saturation attack is orchestrated with minimal resource expenditure. Noteworthy is also the $O(n)$ computational complexity of *TASOS*, surpassing the at least $O(n^2)$ complexities evident in works such as [5, 16, 21, 25, 35].

In summary, *TASOS* stands out as the only work that comprehensively addresses all essential fingerprinting information, including flow table size, utilization rate, and replacement algorithm. *TASOS* fingerprints a broader range of replacement algorithms, including the new RANDOM algorithm, and employs statistical methods to determine thresholds. It maintains a low rate relative to flow timeouts and replacement algorithms in a significantly more efficient manner. Additionally, *TASOS* uniquely performs fingerprinting and saturation attacks simultaneously.

Table 1 succinctly encapsulates key technical contributions, delineating their core methodologies, inferred SDN information through fingerprinting, and any applied defense mechanisms where present.

In a nutshell, a comprehensive survey of existing literature underscores the plethora of SDN fingerprinting approaches, yet their robustness remains subpar, often thriving solely in controlled environments. Concurrently, current saturation attack designs exhibit drawbacks such as inefficiencies and a lack of practical/real-world implementations, signaling the need for enhancements. Furthermore, the scarcity of solutions addressing slow-rate saturation attacks in the

Table 1 Related work

Work	Mechanism and Fingerprint Info
Ahmed et al. [2]	Infer SDN policy parameters such as hard and idle timeouts, match-fields, controller reaction at `table_full` event and information about the topology
Gregorczyk et al. [10]	Infer flow table capacity and flow table usage
Leng et al. [16, 40]	Infer flow table capacity and flow table usage. Determine flow replacement algorithm (LRU or FIFO)
Zhang et al. [39]	Infer match-fields and statistics policy (period and type). Packet-in flood and statistic query attacks. Defense via multi-queue based structure to schedule messages
Yu et al. [35]	Infer flow table capacity and flow table usage. Determine flow replacement algorithm (LRU or FIFO). Intelligent packet-in flood attack
Shen et al. [25]	Infer flow timeouts. Advanced saturation attacks against dynamic timeouts
Huang et al. [12]	Detection of saturation attacks by using packet-in rate and control channel occupation rate
Cao et al. [5]	Infer match-fields including their bit-masks and flow timeouts. Low rate saturation attacks
Pascoal et al. [21]	Low rate saturation attacks by using existing fingerprinting techniques. As defense, randomly evict some flow rules when the flow table is full

domain of defense mechanisms underscores the need for more robust approaches on the attacker side. In particular, adopting a lightweight defense strategy centered on saturation attacks becomes pivotal to foster broader acceptance of anti-fingerprin-ting techniques.

4 Table analyzer and saturation orchestrator for SDN (*TASOS*)

In this section, we introduce *TASOS* as a tool to deduce flow table size, load, and replacement algorithm, facilitating efficient saturation attacks. *TASOS* demonstrates enhanced efficiency and accuracy in fingerprinting compared to previous methods, particularly in diverse and challenging network conditions. Employing statistical methods tailored for such conditions, *TASOS* operates as an automated saturation attack tool. Once the attacker selects the target network, the fingerprinting process operates autonomously. In our prior work [34], we established that an attacker can discern the network type (traditional or SDN) and flow rule timeouts (hard and idle). *TASOS* builds upon this information, enabling an attacker to initiate a saturation attack effectively.

The procedure initiates with the identification of the network type. If recognized as an SDN, the process starts timeout fingerprinting to deduce hard and idle timeouts. Subsequently, flow table fingerprinting is activated, discerning flow table utilization rate, replacement policy and flow table size, respectively.

TASOS relies on three key assumptions. First, the attacker possesses knowledge of at least one flow match-field. This information can be obtained through SDN fingerprinting, as demonstrated in [5], but it is beyond the scope of this paper. Second, the attacker utilizes a host capable of sending packets to a server in the target network. Lastly, there exists a single SDN network on the path to the target network. The attacker sends packets to measure RTTs and infers information based on RTT deviations. To mitigate network jitter, the attacker sends two packets almost consecutively in back-to-back mode. Increased intervals between packets could result in significant network jitter affecting the results.

In our threat model, we assume an adversary who has access to one or multiple hosts to send packets to a server within the targeted network without needing to compromise any other devices on the network, including switches, controllers, and applications. Consequently, the attacker cannot block, save, alter, drop, or replay any communications within the SDN. Additionally, the attacker is incapable of deploying fake network entities to initiate a Sybil attack.

OpenFlow provides two timeouts for flow rules: hard and idle. According to the OpenFlow specification, these timeouts can be assigned values ranging from 0 to 65535 seconds [20]. A value of 0 signifies that the timeout is unset, allowing flow rules to persist until prompted for deletion by the controller. The idle timeout mechanism is deemed more adaptable than hard timeouts. Hard timeouts may result in rule expiration during bursts of packet transmission, leading to issues such as packet loss, increased latency, or degraded network performance [17]. Furthermore, the literature indicates limited adoption of hard timeouts [36]. Consequently, hard timeouts are not factored into the execution of our saturation attack.

To fingerprint an SDN and infer flow table states, careful determination of T_1, T_2 and T_3 thresholds is crucial.

Therefore, in *TASOS*, the initial two packets of a new flow are dispatched to a server within the target network. The first RTT (RTT_1) and second RTT (RTT_2) values are subsequently measured. To render this statistically meaningful, these measurements need to be conducted multiple times. The measurements can be performed in parallel or sequentially; however, parallel execution may elevate the risk of detection for the attacker. Once a set of RTT_1 and RTT_2 measurements is amassed, the sample mean and variance of RTT values are leveraged to determine thresholds, which are then used to infer the slow and slowest paths. Consequently, the sample means (RTT_{1mean} and RTT_{2mean}) and sample standard deviations (RTT_{1std} and RTT_{2std}) of RTT values are computed initially.

The slow path or T_2 indicates the installation of a new flow, providing valuable information for flow deletion control in the context of saturation attack. Statistically, 99.7% of the data falls within three sample standard deviations of the mean. Consequently, to establish a statistically valid threshold, the formula $T_2 = RTT_{2\text{mean}} + 3 \times RTT_{2\text{std}}$ is employed. Thus, when the RTT of a packet surpasses T_2, it is inferred that the flow associated with the packets has been deleted. The slowest path with T_3 indicates an attempt to install a new flow when the table is full, offering insights into the flow table state. Similar to T_2, the statistically valid threshold is determined using the formula $T_3 = RTT_{1\text{mean}} + 3 \times RTT_{1\text{std}}$. If the RTT of a packet exceeds T_3, it is concluded that a flow rule is sent to the switch while the switch's flow table is full.

The parameters utilized in our algorithms are detailed in Table 2. Initially, *TASOS* operates in two distinct threads: one for the installation of new flows and another for sending periodic packets to ensure the longevity of generated flows. Both threads run periodically. Given that SDN saturation attacks do not demand extensive traffic and existing defenses are often triggered by traffic volume [21], the attacker can strategically choose a flow installation rate, denoted as *flowCountPerSecond*, to circumvent defense mechanisms. The period of the flow installation thread is determined as $1/flowCountPerSecond$. Additionally, the period of the flow updater thread can be adjusted by setting the *flowUpdatePeriod* parameter. It's essential for *flowUpdatePeriod* to be less than the idle timeout, denoted as T_{idle}.

The flow update algorithm is outlined in Algorithm 1. The flow update thread assesses active flows and their last update time (Line 1). If the time elapsed since the last update surpasses $T_{\text{idle}} - T_{\text{residual}}$, a packet is dispatched for that flow and its update time is refreshed (Line 3). Here, $T_{residual}$ is pivotal, necessitating a setting between $flowUpdatePeriod < T_{\text{residual}} < 2 \times flowUpdatePeriod$ to ensure that flows never expire.

The flow table fingerprinting algorithm is shown in Algorithm 2. Initially, the process commences with the initialization of parameters (Lines 1–5). Subsequently, within

Algorithm 1: Flow Update Algorithm

```
Data: T_idle, T_residual
1  for flow, timestamp in flows do
2      if (now - timestamp) >(T_idle - T_residual) then
3          update flow and its timestamp
```

Algorithm 2: Flow Table Fingerprinting Algorithm

```
Data: iterationCount (τ), T_3
Result: FT_size, FT_residual, RA
1  FT_size ← −1
2  FT_residual ← −1
3  RA ←"Searching"
4  sendNewFlow ← True
5  algorithmCounter ← 0
6  while sendNewFlow do
7      Send new flow and get RTT
8      if RTT > T_3 then
9          if (FT_residual == −1) then
10             FT_residual ← NoAF
11         if RA is "Searching" then
12             Replacement Algorithm Identifier(RTT)
13         else
14             remove deleted entries
15             FT_size ← NoAF + 1
```

Algorithm 3: Replacement Algorithm Identifier

```
Data: RTT, iterationCount (τ)
Result: RA
1  if RTT == maxValue then
2      REJECT ← REJECT + 1
3      if REJECT > τ then
4          RA ← "REJECT"
5          return
6  else if algorithmCounter == 0 then
7      sendNewFlow ← False
8  if not (sendNewFlow) then
9      if some of our entries are deleted and these entries have
       shortest remaining time for flow update then
10         LRU ← LRU + 1
11         if LRU > τ then
12             RA ← "LRU"
13             sendNewFlow ← True
14             T_residual ← T_idle
15     else
16         algorithmCounter ← algorithmCounter + 1
17         if algorithmCounter > τ then
18             sendNewFlow ← True
19 else
20     if some of our entries are deleted and these entries are in
       order then
21         FIFO ← FIFO + 1
22         if FIFO > τ then
23             RA ← "FIFO"
24             sendNewFlow ← False
25     else
26         RANDOM ← RANDOM + 1
27         if RANDOM > τ then
28             RA ← "RANDOM"
```

Table 2 Parameters for flow table fingerprinting algorithm

Parameters	Explanation
iterationCount (τ)	Number of times the replacement algorithm identifier algorithm is executed
flowCountPerSecond	Number of new flows per second
T_{idle}	Idle timeout
flowUpdatePeriod	Period of the flow update mechanism (in seconds)
$T_{residual}$	Residual idle timeout time used for flow update
FT_{size}	Flow table size
$FT_{residual}$	Residual flow size of the flow table
RA	Flow replacement algorithm
NoAF	Number of active flows

a loop, new flows are sent. If the RTT of a newly introduced flow exceeds the threshold denoted as T_3 (Line 8), it is inferred that the flow table has reached its capacity. Following this, a check is performed to ascertain the status of $FT_{residual}$ (Line 9). If it remains unset, $FT_{residual}$ is established as the count of active flows (Line 10). Thereafter, an assessment is made regarding the existence of a Replacement Algorithm (RA) (Line 11). In the event of its absence, the initiation of the flow table replacement fingerprinting procedure is started (Line 12).

The flow table replacement finder algorithm is delineated in Algorithm 3. Initially, an examination is conducted to determine if the RTT is equivalent to the designated *maxValue* (Line 1). Here, *maxValue* denotes the timeout of the new connection. The intention is to establish a new connection with the transmitted packet. However, when the connection fails, a timeout has occurred. Detecting a timeout implies that the controller rejects the new connection due to a full table. Consequently, the "REJECT" counter is incremented. Nevertheless, it is imperative to acknowledge that timeouts may also arise from network instabilities. Thus, prior to definitively attributing the replacement algorithm as "REJECT", multiple validations are executed to mitigate the impact of variations originating from network fluctuations (Line 3). If the "REJECT" counter surpasses the prescribed iterationCount (τ), the decision is that the replacement algorithm is unequivocally designated as "REJECT."

In the event that the RTT does not equate to *maxValue* and the *algorithmCounter* stands at zero (Line 6), the generation of new flows is terminated (Line 7). Subsequently, at Line 8, a validation ensues to determine whether the new flow has been generated. These checks are imperative due to the iterative nature of the replacement algorithm identifier, leading to diverse execution paths. The halting of new flow generation is essential, as, in the scenario of a full table with a replacement algorithm other than "REJECT", the deletion of a flow becomes imperative. In the case of "LRU" as RA, it is highly probable that one of our flows will be deleted since we update them conservatively. For "FIFO", the likelihood is elevated that a benign flow will be deleted, given their earlier installation. In the case of "RANDOM", the probability of the deleted flow belonging to either our flows or benign ones is contingent upon the ratio of $FT_{residual}$ and FT_{size}.

On Line 8, given its straightforward nature, the next step unfolds with "LRU". Upon the deletion of some of our active flows (this information is derived from the flow updater process), analysis is directed towards ascertaining whether these deleted flows also possess the shortest remaining update time (Line 9). If this condition is met, the LRU counter is incremented; otherwise, the *algorithmCounter* is incremented. RA is not definitively assigned as "LRU", considering the possibility of it being "RANDOM" where deleted flows are coincided by chance. Consequently, a check is instituted to confirm that the LRU counter exceeds the stipulated iterationCount (τ) (Line 11). If this criterion is fulfilled, the replacement algorithm (RA) is set as "LRU" (Line 12). Concurrently, the new flow generation is started (Line 13), aiming at the removal of pre-existing benign rules. To facilitate this, $T_{residual}$ is established as T_{idle} (Line 14), ensuring the periodic update of our flows at each *flowUpdatePeriod* which makes our flows less likely to be selected for removal.

If *algorithmCounter* exceeds τ (Line 17), the generation of new flows recommences (Line 18). In this scenario, RA is either "FIFO" or "RANDOM". Therefore, additional flow generation is indispensable to forcefully displace benign flows. The process includes an examination to ascertain if some of our active flows are deleted and if the deletions adhere to a specific order (Line 20). If so, the FIFO counter is incremented (Line 21); otherwise, the RANDOM counter is increased (Line 26). Once again, "FIFO" is not explicitly assigned to RA, as it may also be "RANDOM", where the deletion occurrences are happenstance. Consequently, a verification is conducted to confirm that the FIFO counter surpasses τ (Line 22). If so, RA is set as "FIFO" (Line 23). Besides, the generation of new flows is ceased (Line 24) since the table is full and FIFO is started to choose our flow

entries for deletion. Similarly, an examination is conducted to determine if the RANDOM counter exceeds τ (Line 27). If affirmative, RA is configured as "RANDOM" (Line 28). Notably, the generation of new flows is not halted, as benign flows may still be present.

Upon determining the replacement algorithm (RA), the initiation of flow table size fingerprinting unfolds. Deleted flows are discarded and excluded from updates (Line 14). This approach enables the preservation of solely the active flows, mitigating the need for further replacements. Subsequently, FT_{size} is designated as the count of active flows (Line 15).

Our algorithm is engineered to both fingerprint the network and execute saturation attacks simultaneously in a single run. The attacker can perpetuate this process, sustaining saturation attacks for an extended duration as desired.

5 Lightweight defense against flow table fingerprinting and saturation attacks (*LIDISA*)

Flow table fingerprinting is crucial for orchestrating efficient DDoS attacks against an SDN controller. Through the discernment of flow table size and replacement algorithm, the attacker can sustain flow rules within the table with optimal resource allocation. Acquiring insights into the flow table's utilization provides a metric for evaluating the effectiveness of the attack in terms of obstructing legitimate flows.

The efficient management of flow tables stands as an active area of research, as exemplified by works such as [30, 37]. However, these endeavors primarily concentrate on enhancing the efficiency of flow tables rather than fortifying their security. Consequently, an attacker can still exploit SDN fingerprinting to orchestrate saturation attacks. Some works, such as [2, 40], offer theoretical guidance on actions to take when the flow table reaches full capacity or how to prevent flow table overflow but lack practical implementations or actionable insights.

In [38], the authors prioritize flows based on evaluation scores to counter saturation attacks. The "importance" field of malicious flows is assigned lower values, with evaluation scores determined by the volume of packet-in instances originating from a specific source address within a given time frame. Conversely, [30] proposes the random eviction of flow rules when flow table utilization surpasses a predefined threshold.

In [24], akin to the approach in [30], flow rules are evicted when TCAM utilization exceeds a predetermined threshold. The selection of flow rules for removal is based on historical flow-removed messages data, where flows that have been removed and subsequently reinstalled after an extended period are candidates for removal. These defensive mechanisms, however, do not effectively thwart flow table fingerprinting or saturation attacks, as they become operational only when the table is full or nearing full capacity. Additionally, a learning attacker employing methodologies like those illustrated in [25] could potentially bypass these mechanisms.

For a defense solution, three distinct success criteria are specified: the prevention of flow table fingerprinting, the mitigation of flow table saturation and compatibility to the SDN with a lightweight mechanism. To counteract fingerprinting, we develop a lightweight randomization mechanism, introducing complexity and uncertainty into the network. This approach necessitates increased investments from potential attackers, rendering the network more challenging to explore. Network parameter randomization, a well-explored domain within Moving Target Defense [6, 28, 29], serves as the foundation for this proposal. Through randomization, data associated with the flow table become a moving target, impeding attackers from predicting network operations and compelling them to expend additional packets and time. The absence of knowledge regarding flow table operations also extends the time and packet requirements for saturation attacks, thereby elevating the likelihood of detection by the defender. This method strategically leverages network statistics while applying randomization to flow table operations, introducing higher levels of unpredictability for attackers while maintaining minimal impact on network functionality.

Our defense proposal in this work embodies an SDN-friendly mechanism, as it can be implemented without necessitating alterations to SDN components or protocols. Given that it randomizes flow rule operations based on basic statistics, its impact on performance is negligible. Moreover, the mechanism is executed by a background thread to prevent any increase in flow setup latencies. The core concept of *LIDISA* revolves around proactive eviction by the controller. Proactive evictions serve as a valuable tool for evading flow table saturation. However, the eviction of an active flow carries the risk of undesired consequences and degradation of network performance. To mitigate this, evictions should be as infrequent as possible, a goal easily attained by evicting only when utilization of the flow table surpasses an acceptable threshold. Additionally, the selection of the flow to be evicted must be executed judiciously, ideally targeting malicious ones. Evicting such flows serves to save TCAM space without adversely affecting legitimate users.

5.1 Design and operation

The defense parameters of *LIDISA* are presented in Table 3. In our solution, the defense module periodically checks the occupancy of the flow table. We measure the number of flow rules by counting `packet-in` and `flow-removed`

messages instead of relying on statistic message usage. This approach ensures that the resources of the controller, switches and even the control channel are not wasted. Upon surpassing a designated threshold, proactive rule deletion mechanism is initiated. However, relying on a fixed threshold presents inflexibility and susceptibility to inference by a learning attacker. To address this concern, we introduce randomness to the threshold in each period using the randomization coefficient (β). This randomization ensures dynamic variations in flow table operations, rendering it challenging for even a learning attacker to adapt to these changes. Our mechanism randomly selects the $\text{FTU}_{threshold}$ value in every period according to the following rule:

$$FTUB_{\text{threshold}} \times (1-\beta) < FTU_{\text{threshold}} < FTUB_{\text{threshold}} \times (1+\beta) \quad (1)$$

For proactive rule eviction, the selection and deletion of certain rule entries are imperative. The objective is to target malicious entries, guided by two key differentiators between normal traffic and attack traffic: the frequency of new flows and the packet count per flow, as noted in [38]. In the pursuit of minimizing attack costs, attackers often opt for a minimal Packet-Per-Second (PPS) ratio, aiming to maximize the impact with fewer packets. To identify malicious flows, we utilize Openflow aggregate flow statistics mechanism to gather information on the number of packets and the duration of flows. Flows are then ranked based on PPS metric, and we select flows with the lowest PPS. The number of selected flows, Nsf, is determined by the following formula:

$$Nsf = FTU_{\text{threshold}} \times FTR_{\text{Deletion}} \quad (2)$$

In this manner, our objective is to exclude normal flows by focusing solely on mice flows. This approach also mitigates the inadvertent impact of unintentionally selecting legitimate flows. To exemplify our mechanism, let's assume that $\text{FTUB}_{threshold}$, β and $\text{FTR}_{Deletion}$ are set at 90%, 0.05 and 0.1, respectively. Additionally, FT_{size} is set as 1000. Therefore, $\text{FTU}_{threshold}$ can fall within the range of 85.5% and 94.5%. Basically, the $\text{FTUB}_{threshold}$ is multiplied by the randomization coefficient, and the result is added/subtracted to find the range of $\text{FTU}_{threshold}$. Suppose in a given period, the flow table has 900 flows, and then $\text{FTU}_{threshold}$ is assigned as 880. We gather the statistics and subsequently choose $880 * 0.1 = 88$ flows with the least PPS and proceed to delete them.

6 Simulations and performance evaluation

The topology employed in the experiments is depicted in Fig. 2. Our configuration encompasses two hosts, serving as a traffic generator and a server, a controller and a single switch. To emulate hosts and switches, Mininet [19] is chosen for its flexibility in configuring link parameters such as bandwidth and delay. RYU [22] serves as the SDN controller. The proposed algorithms are implemented in Python. For simulations and RTT measurements, *TASOS* utilizes UDP packets. However, any other network tool, such as `ping`, can also be employed. Simulations are executed on a computer equipped with an Intel i7-6500U 2.5 GHz (2 cores) processor and 16GB RAM.

Table 3 Parameters for defense mechanism

Parameters	Explanation
$FTU_{threshold}$	Flow table utilization threshold
$FTUB_{threshold}$	Flow table utilization base threshold
Randomization Coefficient (β)	Randomization coefficient
$FTR_{Deletion}$	Ratio of flow rules for deletion
Nsf	Number of the selected flows

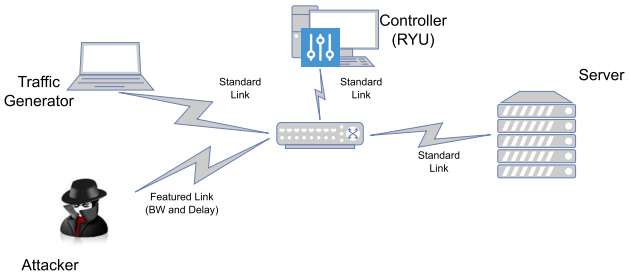


Fig. 2 Simulation topology

6.1 Fingerprinting and saturation experiments

In this section, we conduct experiments to deduce the flow table information. We assume that the network type has already been fingerprinted and we are aware that it is an SDN with known flow rule timeouts [34]. To comprehensively assess the impact of network parameters and explore the robustness of our solution, we generate the same topology with different characteristics. Thus, we investigate the fingerprinting success rate concerning link delay, bandwidth and background traffic (BT). Network and algorithm parameters for the fingerprinting simulation are listed in Table 4 and Table 5, respectively. There are 27 distinct network conditions and each condition is simulated 100 times to analyze accuracy. The network parameters (link delay and bandwidth) are applied to the link of the attacker (Featured Link in the Fig. 2). For background traffic, a traffic generator generates traffic for the server. The algorithm parameters remain constant as we aim to observe the effect of the network parameters. T_{idle} is set to 10 seconds, as idle timeout is typically configured within the range of 9 to 11 s [21]. *flow-*

Table 4 Network parameters

Network parameters	
Link delay (ms)	0, 50, 100
Bandwidth (Mbps)	10, 100, 1000
BT (Mbps)	10, 100, 1000

Table 5 Algorithm parameters

Parameter	Value
iterationCount (τ)	3
flowCountPerSecond	1
T_{idle}	10
flowUpdatePeriod	1
FT_{size}	1000
$T_{residual}$	1.5
$FT_{residual}$	500
RA	REJECT

Table 6 FT_{size} accuracy for different network parameters

(BW, BT)	LD = 0	LD = 50	LD = 100
(10,10)	1	1	1
(10,100)	1	1	1
(10,1000)	1	1	1
(100,10)	1	1	**0.99**
(100,100)	1	1	1
(100,1000)	1	1	1
(1000,10)	1	1	**0.99**
(1000,100)	1	1	1
(1000,1000)	1	1	**0.99**

CountPerSecond is set to 1 to demonstrate that a low-rate packet-in generation is still sufficient for saturation attacks. The benign flows are initiated at the onset of the attack. They are continually updated until the flow table reaches full capacity. Subsequently, they are gradually displaced by malicious flows or evicted due to timeouts.

The accuracy of *TASOS* for flow table size fingerprinting under different network parameters is presented in Table 6. *TASOS* consistently identifies the flow table utilization and replacement algorithm correctly. Highlighted cells indicate results that differ from 1. The overall success ratios of the simulations range between 0.99 and 1.0. Notably, as the link delay increases, the success probability appears to be negatively impacted. This observation is significant, as a larger link delay limits the effect of the flow setup delay. Although RTT also increases in such scenarios, the flow setup time for the network remains constant. Bandwidth and background traffic appear to have a negligible impact on the success ratio of *TASOS*.

Table 7 Algorithm parameters

Algorithm parameters	
FT_{size}	100, 500, 1000
$FT_{residual}$	50%, 75%
RA	LRU, FIFO, REJECT, RANDOM

TASOS exhibits superior accuracy under more challenging network conditions compared to related works [2, 10, 16, 35]. For scalability analysis, we conduct 27 different simulations to examine the effects of background traffic, link delay and bandwidth. Notably, [2, 10, 16, 35] do not consider link delay or bandwidth in their experiments and only [35] examines the effect of background traffic, which severely degrades its performance.

Subsequently, to understand the impact of certain algorithm parameters, we conduct additional experiments using maximum values for the network parameters for a marginal case. Thus, link delay, bandwidth and BT are set to 100 ms, 1000 Mbps and 1000 Mbps, respectively. Algorithm parameters for this simulation are detailed in Table 7. $FT_{residual}$ is established as a ratio of FT_{size}. There are 24 different conditions and each condition is simulated 100 times to assess the accuracy performance.

Similar to the network parameter experiments, the success ratio for flow table utilization and replacement algorithm is 1.0. The accuracy of *TASOS* for fingerprinting with different algorithm parameters is presented in Table 8. The success ratios of REJECT, LRU and FIFO are consistently high, ranging between 0.98 and 1.0. RANDOM exhibits lower success ratios, as, after the flow table is filled with our flows, it is probable that one of our flows is evicted. Consequently, the flow table size may be inaccurate if the simulation duration concludes before determining the exact table size. However, we generally determine the table size with a margin of error of minus 5%. Another noteworthy finding is that the flow table size or utilization rate does not significantly affect the success ratio of our algorithms.

We do not exclusively perform simulations for saturation attacks, as accurately fingerprinting flow table parameters necessitates executing saturation attacks. These parameters cannot be determined without saturating the flow table. Consequently, the efficiency of the saturation attacks mirrors the results related to flow table size. Once fingerprinting is completed, the attack can proceed with greater efficiency since the attacker knows the required number of flows to saturate the flow table and how to minimize the probability of eviction of their flows.

In saturation attack, the switch's flow table becomes saturated, hindering its ability to accommodate new benign flows.

Table 8 Flow table size fingerprinting accuracy results

$(FT_{size}, FT_{residual})$	REJECT	FIFO	LRU	RANDOM
(100,50%)	1	1	**0.99**	**0.98**
(100,75%)	**0.99**	**0.98**	1	**0.98**
(500,50%)	1	1	1	**0.97**
(500,75%)	1	**0.98**	**0.99**	**0.97**
(1000,50%)	**0.98**	1	1	**0.94**
(1000,75%)	**0.99**	**0.99**	**0.99**	**0.93**

Consequently, the switch struggles to serve these new flows effectively. The accuracy of fingerprinting is crucial for the success of such attacks. For instance, if the flow table is incorrectly assessed as full and the residual flow table size is inaccurately identified, RA fingerprinting may also fail. Consider a scenario where the RA is LRU, but it is incorrectly identified as FIFO; as a result, the update period is not increased, and resulting in continuous deletion of malicious flows. Consequently, both the saturation attack and flow table fingerprinting would fail under these circumstances.

In simulations, *TASOS* operates at a low rate, with *flowCountPerSecond* set to 1. Given that common detection systems rely on packet-in message rates to identify saturation attacks, *TASOS* has the potential to evade these detection mechanisms.

In [2], only the controller's reaction to the table full event (REJECT or not) is inferred and there is no accuracy calculation. In [10, 16], flow table size and utilization are estimated with error margins of 10% and 1%, respectively, without determining the exact values. Moreover, both [10, 16] assume that replacement algorithm is given. Additionally, [16] only considers LRU and FIFO, while [10] considers just FIFO. Once again, *TASOS* demonstrates superior accuracy under more challenging network conditions compared to related works [2, 10, 16, 35]. However, these comparisons are naturally contingent on the best available knowledge regarding the design and performance of those algorithms by the respective authors and for different (simulated or real) network environments. It is essential to note that, for instance, *TASOS* might be operating in a "more favorable" network environment.

Another crucial consideration is the cost associated with fingerprinting and saturation attacks. Our algorithm is designed to execute flow table fingerprinting and saturation attacks simultaneously. Moreover, *TASOS* acquires all necessary information, including flow table size, utilization rate, and replacement algorithm. Notably, there is no single related work that comprehensively covers all these data. Furthermore, *TASOS* introduces additional replacement algorithms for fingerprinting.

6.2 Fingerprinting countermeasure simulations

We evaluate *LIDISA*, which employs proactive eviction to counteract fingerprinting and saturation attacks. In simulations, $\text{FTUB}_{threshold}$, β and $\text{FTR}_{Deletion}$ are set at 90%, 0.05 and 0.1, respectively. Additionally, FT_{size} and $\text{FT}_{residual}$ are set as 1000 and 500, respectively. The remaining parameters are the same as those in Table 7. In the simulations, we execute both our attack and defense modules. Our attack algorithm consistently provides incorrect estimates for flow table size and utilization. The replacement algorithm is occasionally correctly identified by chance; however, it is predominantly deduced as RANDOM.

To assess the impact of our solution on performance, *TASOS* initiates attacks on the network with the defense mechanism being both active and inactive. We monitor average response time to packet-in, CPU usage and memory usage metrics to gauge the effect of our defense solution. The performance simulation is conducted 100 times, with 50 of them featuring an active defense solution. Our defense module solely compares the active flow count with $\text{FTU}_{threshold}$ during normal operations, resulting in minimal impact. In an attack scenario, it gathers statistics and deletes flows based on these statistics. According to the simulations, the average response time for `packet-in` increases by 3.5%, rising from 5.45 to 5.64 milliseconds during attack times. Similarly, CPU and memory usage experience increase of 6.54% and 1.86%, respectively. These results indicate that the proposed method has a slight impact on performance even during attack times while effectively preventing fingerprinting and saturation attacks.

7 Discussion

SDN has become an integral part of today's ICT system, finding applications in various segments of networking and telecommunications infrastructure. However, since its widespread deployment in controlled and restricted environments such as data centers and core networks, security concerns within the realm of SDN are often relegated to academic discussions. Moreover, the perceived performance implications of security countermeasures are often cited as justifications for less risk-aware treatments of potential threats in this domain.

Traditional flooding DDoS attacks and SDN saturation attacks share similarities but differ in several aspects. Notably, if the connection between the controller and switches is out-of-band (as is often the case), traditional flooding DDoS attacks cannot be executed by simply flooding the link. Additionally, flooding attacks are generally more brute-force in nature, whereas saturation attacks employ more sophisticated methods. For example, saturation attacks

typically require far less bandwidth compared to traditional DDoS attacks (measured in Kbps versus Gbps).

SDN flow table fingerprinting can be conducted both with and without the use of spoofing. The need for IP spoofing usually arises when the match fields of the flow rules require it, especially in scenarios involving the parallel execution of fingerprinting algorithms. For instance, if the match fields include layer 4 port information, generating flow rules might be triggered by simply altering the port numbers. While detecting spoofing can be relatively easy when the attacker is inside the targeted network, it becomes more complicated when the attacker is external. Consequently, the detection or prevention of SDN fingerprinting is not a straightforward task using traditional security approaches. In our simulations, the fingerprinting process is executed without spoofing.

The duration of the SDN fingerprinting process varies based on network characteristics, fingerprinting algorithm parameters and the execution method of the algorithms. The parameter most dominant in affecting duration is *flowCountPerSecond*. Its careful assignment is crucial, as large values can risk detection of the attack, while small values can extend the attack's duration. This extension also increases the required number of packets for fingerprinting, given that the flow update process can generate numerous refreshing packets. Therefore, there exists an implicit trade-off between *flowCountPerSecond*, duration and the cost of the attack. An interesting research problem could involve finding the optimal value for *flowCountPerSecond* according to different network characteristics.

Another critical consideration is the dependency of fingerprinting accuracy on several factors, including the attack's duration, the chosen replacement algorithm, and the characteristics of benign flows (such as volume, duration, and rate). For instance, if the replacement algorithm is set to REJECT and certain legitimate flows never expire due to timeouts, flow table size fingerprinting may yield incorrect results. A similar effect occurs when there are permanent flows (where both flow timeouts (hard and idle) are not set). Moreover, the rate at which new benign flows are generated can significantly impact the algorithm's accuracy. When the replacement algorithm is RANDOM or FIFO, the algorithm may struggle to accurately determine the true flow table size, especially if the rate of benign flows exceeds the flowCountPerSecond parameter. In the case of RANDOM, as the flow table fills with malicious flows, the probability increases that one of these malicious flows will be selected for eviction. Consequently, evacuating all legitimate flows may take a considerable amount of time or even fail altogether, leading to inaccurate estimations of flow table size. However, even in such scenarios, the saturation attack can still be successful as the network fails to handle legitimate flow installation requests continuously. Additionally, the eviction of benign flows will degrade network performance. Exploring the fingerprinting of the rate of new benign flows and adjusting the flowCountPerSecond parameter accordingly could be an intriguing research avenue to enhance the accuracy and efficiency of fingerprinting and saturation attacks. Thus, there is an implicit trade-off between accuracy and overhead for the attacker.

The algorithm may encounter prolonged waiting periods in scenarios where the table reaches full capacity (Line 8 in Algorithm 2), the RTT is not maxValue (Line 1 in Algorithm 3), and benign flows are evicted due to timeouts. In such cases, where the algorithm ceases to send new flows and there is still available space in the flow table, it must await the gradual filling of the table with benign flows, a process that may take noticeable time. In that case, attacker can reactivate new flow creation manually, or the algorithm can make it automatically.

The simulation topology utilized in our research is relatively simple since the inclusion of additional hosts or switches has minimal effect on the fingerprinting process. The end-to-end latency of the probing packet is a key factor for fingerprinting since the entire concept relies on the RTT difference of different packets. Therefore, introducing more hosts and switches can potentially increase the load on the controller, subsequently enhancing the success ratio of fingerprinting, as the delay in installing flow rules is contingent on the controller's workload. Moreover, the difference in RTT can significantly increase if there is more than one switch installed between the communicating end hosts, with each switch repeating the process of sending the unknown flow to the controller. Nevertheless, evaluating *TASOS* and *LIDISA* in real-world scenarios can provide valuable data and insights to validate their effectiveness and uncover potential areas for enhancement. However, SDN-related publicly available data is somewhat lacking, and we have not been able to find any.

If there are simultaneous attackers attempting to perform fingerprinting and/or saturation attacks, they likely diminish each other's accuracy and effectiveness since each attacker tries to evict the other's flows to saturate the flow table. If the attackers collaborate, it can be considered as a single attacker utilizing multiple machines. We also anticipate that our defense mechanism remains unaffected by the number of hosts or attackers, as it primarily monitors flow table occupancy and evicts the least used flows when necessary. However, our mechanism may evict more benign flows in the presence of multiple simultaneous attackers. This presents an intriguing avenue for future research.

While on the road to the target network, our implicit assumption is that the probing packets exclusively traverse a single SDN network. The viability of our algorithm in real-world scenarios remains uncertain, primarily due to the lack of data regarding the deployment ratio of SDN in the telecommunications sector. Exploring the transit of packets through

a multi-domain SDN in a fingerprinting scenario could be an interesting research direction for future work. This becomes particularly intriguing when switches on the path have different flow tables sizes.

With our proposed defense mechanism, *LIDISA*, our goal isn't to completely eliminate saturation attacks but rather to increase the difficulty for the attacker. Presently, attackers can accurately deduce FT_{size} and $FT_{residual}$ with relative ease and reliability. Even if the attacker is aware of our defense algorithm, *LIDISA* introduces an element of uncertainty, as these parameters are approximated based on random thresholds that the attacker cannot predict. Furthermore, attackers can evict benign flows without expending significant traffic. However, with *LIDISA*, attackers must exceed an unknown flow rate to evict these flows. If attackers are willing to invest additional resources by sending more packets, they can evict more active flows. Therefore, our objective of preventing fingerprinting and mitigating saturation attacks is achieved by increasing their costliness.

The performance of our algorithm, *LIDISA*, can be influenced by the characteristics of benign flows (e.g., volume, duration, and rate). If numerous benign flows exhibit significantly low rates, *LIDISA* may select these benign flows for eviction.

8 Conclusions

While there is a significant body of work utilizing SDN as a security enabler, the security of SDN itself remains a critical concern. In this work, we introduce the concept of SDN flow table fingerprinting and the type of information that can be extracted from it. Additionally, we address saturation attacks against SDN flow tables. We propose *TASOS*, an efficient and automated fingerprinting agent capable of inferring the flow table size, utilization rate and replacement algorithm. Subsequently, using the fingerprint data, *TASOS* proceeds to execute efficient saturation attacks. *TASOS* achieves high accuracy even in challenging network conditions. Extensive simulations have been conducted across a diverse range of network parameters, demonstrating that *TASOS* performs exceptionally well in those scenarios. We also present a lightweight proactive eviction mechanism (*LIDISA*) that increases the cost for attackers, making it more challenging for them to explore and exploit the network.

As a potential avenue for future research, machine learning algorithms, particularly deep learning, could be employed to extract more diverse information from an SDN network through fingerprinting techniques. Exploring defense solutions aiming to blur the difference between T_1 and T_2 could also be an intriguing research idea.

Funding Open access funding provided by the Scientific and Technological Research Council of Türkiye (TÜBİTAK).

Research data policy and data availability statements The data used in this study are available upon request.

Declarations

Conflict of interest Authors declare no Conflict of interest except with the universities mentioned in the affiliations.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Ahmad, S., Mir, A.H.: Protection of centralized SDN control plane from high-rate Packet-In messages. Int. J. Inf. Secur. (2023). https://doi.org/10.1007/s10207-023-00685-z
2. Ahmed, B., Ahmed, N., Malik, A.W., Jafri, M., Hafeez, T.: Fingerprinting SDN policy parameters: an empirical study. IEEE Access **8**, 142379–142392 (2020). https://doi.org/10.1109/ACCESS.2020.3012176
3. Ambrosin, M., Conti, M., De Gaspari, F., Poovendran, R.: LineSwitch: tackling control plane saturation attacks in software-defined networking. IEEE/ACM Trans. Netw. **25**(2), 1206–1219 (2016)
4. Ayodele, B., Buttigieg, V.: SDN as a defence mechanism: a comprehensive survey. Int. J. Inf. Secur. (2023). https://doi.org/10.1007/s10207-023-00764-1
5. Cao, J., Xu, M., Li, Q., Sun, K., Yang, Y., Zheng, J.: Disrupting SDN via the data plane: a low-rate flow table overflow attack. In: Security and Privacy in Communication Networks: 13th International Conference, SecureComm 2017, Niagara Falls, Proceedings 13, pp. 356–376. Springer (2018)
6. Cho, J.H., Sharma, D.P., Alavizadeh, H., Yoon, S., Ben-Asher, N., Moore, T.J., Kim, D.S., Lim, H., Nelson, F.F.: Toward proactive, adaptive defense: a survey on moving target defense. IEEE Commun. Surv. Tutor. **22**(1), 709–745 (2020). https://doi.org/10.1109/COMST.2019.2963791
7. Correa Chica, J.C., Imbachi, J.C., Botero Vega, J.F.: Security in SDN: a comprehensive survey. J. Netw. Comput. Appl. **159**, 102595 (2020). https://doi.org/10.1016/j.jnca.2020.102595
8. Cui, H., Karame, G.O., Klaedtke, F., Bifulco, R.: On the fingerprinting of software-defined networks. IEEE Trans. Inf. Forensics Secur. **11**(10), 2160–2173 (2016). https://doi.org/10.1109/TIFS.2016.2573756

9. Deb, R., Roy, S.: A comprehensive survey of vulnerability and information security in SDN. Comput. Netw. **206**, 108802 (2022). https://doi.org/10.1016/j.comnet.2022.108802
10. Gregorczyk, M., Mazurczyk, W.: Inferring flow table state through active fingerprinting in SDN environments: a practical approach. In: Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT 2021, pp. 576–586 (2021). https://doi.org/10.5220/0010573900002998
11. Guo, Z., Liu, R., Xu, Y., Gushchin, A., Walid, A., Chao, H.J.: Star: preventing flow-table overflow in software-defined networks. Comput. Netw. **125**, 15–25 (2017)
12. Huang, X.B., Xue, K.P., Xing, Y.T., Hu, D.W., Li, R., Sun, Q.B.: An efficient scheme to defend data-to-control-plane saturation attacks in software-defined networking. J. Comput. Sci. Technol. **37**(4), 839–851 (2022)
13. Kaur, S., Sandhu, A.K., Bhandari, A.: Investigation of application layer DDoS attacks in legacy and software-defined networks: a comprehensive review. Int. J. Inf. Secur. **22**(6), 1949–1988 (2023)
14. Khorsandroo, S., Tosun, A.S.: Time inference attacks on software defined networks: challenges and countermeasures. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), pp. 342–349 (2018). https://doi.org/10.1109/CLOUD.2018.00050
15. Klöti, R., Kotronis, V., Smith, P.: OpenFlow: a security analysis. In: 2013 21st IEEE International Conference on Network Protocols (ICNP), pp. 1–6 (2013). https://doi.org/10.1109/ICNP.2013.6733671
16. Leng, J., Zhou, Y., Zhang, J., Hu, C.: An inference attack model for flow table capacity and usage: Exploiting the vulnerability of flow table overflow in software-defined network. (2015). Preprint at arXiv:1504.03095
17. Li, Q., Huang, N., Wang, D., Li, X., Jiang, Y., Song, Z.: HQTimer: a hybrid Q-learning-based timeout mechanism in software-defined networks. IEEE Trans. Netw. Serv. Manage. **16**(1), 153–166 (2019)
18. Maleh, Y., Qasmaoui, Y., El Gholami, K., Sadqi, Y., Mounir, S.: A comprehensive survey on SDN security: threats, mitigations, and future directions. J. Reliab. Intell. Environ. 1–39 (2022)
19. Mininet. http://mininet.org/. Accessed 03 Jan 2024
20. OpenFlow Switch Specification 1.5.1. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. Accessed 03 Jan 2024
21. Pascoal, T.A., Fonseca, I.E., Nigam, V.: Slow denial-of-service attacks on software defined networks. Comput. Netw. **173**, 107223 (2020)
22. RYU SDN Framework. https://ryu-sdn.org/. Accessed 03 Jan 2024
23. Shang, G., Zhe, P., Bin, X., Aiqun, H., Kui, R.: FloodDefender: protecting data and control plane resources under SDN-aimed DoS attacks. In: IEEE INFOCOM 2017-IEEE Conference on Computer Communications, pp. 1–9. IEEE (2017)
24. Shen, Y., Wu, C., Cheng, Q., Kong, D.: AFTM: An adaptive flow table management scheme for OpenFlow switches. In: 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 917–922. IEEE (2020)
25. Shen, Y., Wu, C., Kong, D., Cheng, Q.: Flow table saturation attack against dynamic timeout mechanisms in SDN. Appl. Sci. **13**(12), 7210 (2023)
26. Shin, S., Gu, G.: Attacking software-defined networks: a first feasibility study. In: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, p. 165–166. Association for Computing Machinery, New York (2013). https://doi.org/10.1145/2491185.2491220
27. Shin, S., Yegneswaran, V., Porras, P., Gu, G.: Avant-guard: scalable and vigilant switch flow management in software-defined networks. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pp. 413–424 (2013)
28. Soussi, W., Christopoulou, M., Gür, G., Stiller, B.: MERLINS - moving target defense enhanced with deep-RL for NFV in-depth security. In: 2023 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 65–71 (2023). https://doi.org/10.1109/NFV-SDN59219.2023.10329594
29. Soussi, W., Christopoulou, M., Xilouris, G., Gür, G.: Moving target defense as a proactive defense element for beyond 5G. IEEE Commun. Stand. Mag. **5**(3), 72–79 (2021). https://doi.org/10.1109/MCOMSTD.211.2000087
30. Vishnoi, A., Poddar, R., Mann, V., Bhattacharya, S.: Effective switch memory management in OpenFlow networks. In: Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, pp. 177–188 (2014)
31. Wang, H., Xu, L., Gu, G.: Floodguard: A DOS attack prevention extension in software-defined networks. In: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 239–250. IEEE (2015)
32. Yaacoub, J.P.A., Noura, H.N., Salman, O., Chehab, A.: Robotics cyber security: Vulnerabilities, attacks, countermeasures, and recommendations. Int. J. Inf. Secur. 1–44 (2022)
33. Yigit, B., Gur, G., Tellenbach, B., Alagoz, F.: Secured communication channels in software-defined networks. IEEE Commun. Mag. **57**(10), 63–69 (2019). https://doi.org/10.1109/MCOM.001.1900060
34. Yiğit, B., Gür, G., Alagöz, F., Tellenbach, B.: Network fingerprinting via timing attacks and defense in software defined networks. Comput. Netw. **232**, 109850 (2023). https://doi.org/10.1016/j.comnet.2023.109850
35. Yu, M., Xie, T., He, T., McDaniel, P., Burke, Q.K.: Flow table security in SDN: adversarial reconnaissance and intelligent attacks. IEEE/ACM Trans. Netw. **29**(6), 2793–2806 (2021)
36. Zarek, A., Ganjali, Y., Lie, D.: Openflow Timeouts Demystified. University of Toronto, Toronto (2012)
37. Zhang, L., Wang, S., Xu, S., Lin, R., Yu, H.: TimeoutX: an adaptive flow table management method in software defined networks. In: 2015 IEEE Global Communications Conference (GLOBECOM), pp. 1–6. IEEE (2015)
38. Zhang, M., Bi, J., Bai, J., Dong, Z., Li, Y., Li, Z.: FTGuard: a priority-aware strategy against the flow table overflow attack in SDN. In: Proceedings of the SIGCOMM Posters and Demos, pp. 141–143 (2017)
39. Zhang, M., Li, G., Xu, L., Bai, J., Xu, M., Gu, G., Wu, J.: Control plane reflection attacks and defenses in software-defined networks. IEEE/ACM Trans. Netw. **29**(2), 623–636 (2020)
40. Zhou, Y., Chen, K., Zhang, J., Leng, J., Tang, Y.: Exploiting the vulnerability of flow table overflow in software-defined network: attack model, evaluation, and defense. Secur. Commun. Netw. **2018**, 1–15 (2018)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.