Flow Table Security in SDN: Adversarial Reconnaissance and Intelligent Attacks

Mingli Yu, *Student Member, IEEE*, Tian Xie, *Student Member, IEEE*, Ting He, *Senior Member, IEEE*, Patrick McDaniel, *Fellow, IEEE* and Quinn K. Burke, *Student Member, IEEE*

***Abstract*—The performance-driven design of SDN architectures leaves many security vulnerabilities, a notable one being the communication bottleneck between the controller and the switches. Functioning as a cache between the controller and the switches, the flow table mitigates this bottleneck by caching flow rules received from the controller at each switch, but is very limited in size due to the high cost and power consumption of the underlying storage medium. It thus presents an easy target for attacks. Observing that many existing defenses are based on simplistic attack models, we develop a model of intelligent attacks that exploit specific cache-like behaviors of the flow table to infer its internal configuration and state, and then design attack parameters accordingly. Our evaluations show that such attacks can accurately expose the internal parameters of the target flow table and cause measurable damage with the minimum effort.**

***Index Terms*—Software Defined Networking, cache inference, Denial of Service attack.**

I. Introduction

As a new networking paradigm, *Software Defined Networking (SDN)* has fundamentally changed the way networks are built and maintained. By separating the data plane and the control plane, SDN moves the control functions to a logically centralized controller, thus enabling flexible routing, service composition, and network management. These advantages have lead to massive adoption of SDN in enterprises and datacenters worldwide.

Meanwhile, the widespread adoption raises the issue of security. While SDN eases the defense against traditional IP-network attacks such as port scanning and firewall probing [2] by using agile structures and policies [3], it also introduces new vulnerabilities that allow attackers to learn about the target network and use the learned information to attack it.

We hereby focus on the vulnerabilities of the *flow table*, which is a data structure at each SDN-enabled switch that stores the flow rules received from the controller. These flow rules, each containing match, action, and several other fields (e.g., priority, counters), encode how the controller wants each flow to be processed by the switch. Packets not matching existing rules in the flow table will typically be forwarded to the controller for further processing, which invokes slow elements such as the switch CPU [4] and significantly degrades the performance. However, due to the high cost and power consumption of the underlying storage medium, flow tables are usually small, holding up to a few thousand rules [5].

The authors are with the Pennsylvania State University, University Park, PA, USA. Email: {mxy309, tbx5027, tzh58, pdm12, qkb5007}@psu.edu.

A preliminary version of this work was presented at INFOCOM'20 [1].

This research was sponsored by the National Science Foundation under award CNS-1946022.

This vulnerability has been explored to launch various flow table overflow attacks [6], [7], [8], [9] and control plane saturation attacks [10]. However, existing studies have only modeled *unintelligent attacks* that apply simple techniques to bluntly harm the target. We argue that an *intelligent attacker*, that employs a reconnaissance stage to learn the internal configuration (e.g., size, policy) and state (e.g., load) of the target flow table, can attack more precisely and efficiently.

To demonstrate the above claim, we develop algorithms to explicitly infer the size, policy, and state of the target flow table from probes sent by a compromised host, based on which intelligent Denial of Service (DoS) attacks can be mounted to cause measurable damage with the minimum effort.

A. Related Work

SDN vulnerabilities: The design of existing SDN architectures and protocols is performance-driven, leaving many security vulnerabilities. The weakest aspect of SDN is the interdependency between the controller and the switches. The *controller$\rightarrow$switch dependency* arises due to the needs for the controller to maintain an updated network state for making proper control decisions. This creates a vulnerability if the attacker can gain control of one or more switches [11] to inject fake links [12], eavesdrop on control messages and client traffic [13], impersonate the controller [14], or otherwise subvert the control applications by sending false reports [15]. The *switch$\rightarrow$controller dependency* arises due to the needs for the data plane to obtain instructions on packet processing from the control plane, which can create a communication bottleneck [4]. This bottleneck can be exploited in active attacks [6], [7], [8], [9], [10], adversarial reconnaissance [16], [17], [18], [19], and joint reconnaissance and attack [20]. We will exploit this bottleneck for joint reconnaissance and attack on the flow table. Unlike previous works [17], [18], [19] that focus on inferring specific rules, we aim at inferring global parameters of the flow table (e.g., size, policy) useful for planning later attacks.

SDN defenses: Traditional network defenses are often based on detecting the signatures of specific attacks or using machine learning or other models to detect anomalies against normal behaviors. In SDNs, although the centralized control eases the defense against traditional network attacks through agile structures and policies [3], it remains open how to design robust defenses against a large threat surface and unknown attacks. Existing defenses can be classified into *proactive defenses* that extend OpenFlow to mitigate vulnerabilities such as [10], and *reactive defenses* that focus on detecting specific ongoing attacks such as [12], [21]. However, a systematic understanding of the attack vectors and fundamental limits is still missing. In this sense, our work contributes to the future development of defenses in SDN by developing reconnaissance techniques that utilize the switch→controller dependency to extract information about the internal states and policies of SDN switches and demonstrating how this information can be used to launch more intelligent attacks.

Flow table management: To put our work into context, we briefly review techniques for flow table management. Existing techniques can be classified into: (i) rule replacement, (ii) rule compression, and (iii) rule distribution [22]. Among the three, approaches (ii-iii) still have many open issues [22]. In contrast, approach (i) is a mature technology that has been widely implemented in OpenFlow switches, e.g., Open vSwitch supports the First In First Out (FIFO) policy and the Least Recently Used (LRU) policy [23], and hardware switches usually employ FIFO [7]. We will thus focus on flow tables managed by rule replacement policies.

Rule replacement policies & their security: Existing rule replacement policies are designed exclusively for a *benign environment*. In addition to FIFO and LRU, researchers have proposed other rule replacement policies, such as those based on the Least Frequently Used (LFU) policy [24], [25], [26], and the approximation of Bélády's optimal replacement policy [27]. In contrast, very few studies have considered the security of rule replacement policies in an *adversarial environment*. In [7], [8], [9], the feasibility of filling the flow table with the attacker's rules was demonstrated, but the attack was unintelligent. In [6], an attack was designed to cause flow table overflow with the minimum rate of attacking traffic. However, it assumed that an attacker's rule will remain in the flow table until its timeout, which is not valid with reactive eviction. The work closest to ours is [16], which attempted to model an intelligent adversary that infers the flow table size and occupancy under a given policy (FIFO or LRU). However, their algorithms require knowledge of the replacement policy, and ignore the interference from background traffic.

Security of general caches: Viewing the flow table as a cache of flow rules, one might borrow results on inference/attack of general caches, but even then existing results are very limited. In [28], a procedure was proposed to infer the cache replacement policy and input parameters from observations of all the misses. This solution is not applicable to adversarial reconnaissance as the attacker can only observe the results of his own packets. In [29], two attacks were proposed to replace popular contents in the cache by unpopular contents, but their parameters were not intelligently designed. In [30], algorithms were proposed to infer the size and other parameters of an LRU cache, but it did not address the problem of unknown cache replacement policy.

B. Summary of Contributions

We demonstrate the feasibility of *intelligent reconnaissance-based attacks* in SDN by exploiting the data-control plane bottleneck and specific cache-like behaviors of the flow table.

1) We formulate the *adversarial cache inference problem* to jointly infer both static parameters (e.g., size, policy) and dynamic parameters (e.g., current load) of the flow table from a single compromised host.
2) By analyzing the behaviors of candidate policies, we develop algorithms to explicitly infer the size, the policy, and the load parameters of the target flow table, using only two primitives that have been shown to be feasible.
3) We demonstrate the value of the inferred information by designing intelligent Denial of Service (DoS) attacks that minimize the attack rate while sufficiently degrading the performance of legitimate users.
4) We verify through synthetic/trace-driven simulations and trace-driven Mininet experiments that the proposed solution can achieve accurate reconnaissance (with more than 90% accuracy) and effective attack design despite interference from background traffic.

Compared to the preliminary version [1], this work makes the following additional contributions: (i) we analyze the impact of controller-imposed timeouts on our algorithms; (ii) we add a discussion of possible defenses against the proposed attacks, identifying a tradeoff between defense efficacy and performance penalty for future research; (iii) we demonstrate an alternative method for policy inference based on TTL approximation that achieves a high accuracy under heavy background traffic and a low probing rate, which complements the previously proposed algorithm that requires light background traffic and a relatively high probing rate; (iv) we use traces of significantly higher rates and complement the simulations with experiments based on a virtual SDN testbed, thus validating our algorithms in a more realistic environment.

Roadmap. Section II introduces our models and problem formulation. Sections III–IV present our solutions for reconnaissance and attack design. Sections V–VI validate the proposed solutions. Section VII concludes the paper. Further considerations (timeouts, defenses) are provided as supplementary materials in Section VIII.

II. Problem Formulation

A. Flow Table Model

We model the flow table at the target switch as a *cache of flow rules*. This model is valid as long as all the rules are stored in a single table, which is often the case in practice. According to the OpenFlow Switch Specification 1.5.1 [31], each rule contains a number of fields including match, priority, counter, action, and timeout, that specify which packets will be processed by this rule and how. OpenFlow allows a variety of header fields to be used as match fields, e.g., source/destination MAC addresses, IP addresses, and port numbers. It is up to the controller which fields to use. However, prior work [6] has shown that by sending probing packets while changing one header field at a time, the attacker can learn which header fields are used in matching packets.

As a cache of flow rules, the flow table is characterized by two basic parameters: (1) the *size* that specifies the maximum number of stored flow rules and (2) the *replacement policy* that specifies which rule will be evicted if a new rule needs to be installed when the table is full. The first parameter is akin to the cache size, and the second parameter is akin to the cache replacement policy. Although OpenFlow also allows rules to be proactively removed due to timeouts, the use of timeouts is optional, and their influence will be dominated by reactive rule replacements when the flow table is full.

We denote the flow table size by C (unit: rules), and the replacement policy by π. In commodity switches, C is usually small, up to a few thousand [5]. The replacement policy π also comes from a small set of candidate policies. For example, Open vSwitch [23] implements an approximation of Least Recently Used (LRU) when rules have idle timeouts and no hard timeout, or First In First Out (FIFO) when rules have hard timeouts and no idle timeout[1]. A study [7] found that certain hardware switches use FIFO. While there are more sophisticated policies proposed by researchers, e.g., [24], [25], [26], [27], they are yet to be adopted in production. We will therefore focus on the common case of $\pi \in \{\text{FIFO, LRU}\}$ and discuss extensions to other cases when appropriate.

B. Adversary Model

We consider an external attacker that performs reconnaissance and attacks against the target switch from a compromised host. This also models coordinated reconnaissance and attacks from multiple compromised hosts against the same switch. We assume the following primitives for the attacker:

- **Primitive 1:** The attacker can detect whether a given probe results in a hit or miss at the target flow table. This capability has been demonstrated in previous studies [16], [33]. For example, the attacker can measure the round-trip time (RTT) of the probe and compare it with a threshold learned from RTTs of sure misses (e.g., packets with randomly generated source IP addresses) and sure hits (e.g., the second packet in a pair of back-to-back packets with identical headers).
- **Primitive 2:** The attacker can craft a probe that requires a new rule. It has been shown in [6] that the attacker can learn the matching fields and craft the probes by modifying one or multiple matching fields, such that each crafted probe requires a distinct rule. Moreover, as each matching field has a large solution space (e.g., all MAC/IP addresses or port numbers), the randomly crafted matching fields will almost never coincide with those of legitimate packets.

Let d_I denote the (average) delay between a miss and the time that the requested rule is installed, i.e., the rule installation time. We assume that the attacker can estimate d_I by measuring the differences between RTTs of hits and misses.

Remark: Our attacks are based on the above primitives that have been validated on current SDN implementations. In particular, Primitive 1 has only been validated on edge switches (Primitive 2 is about learning the matching fields used by the controller, hence independent of the location of the target switch). Intuitively, edge switch is the most vulnerable to host-based attacks as it is directly exposed to the attacker. It remains open how feasible it is to perform similar reconnaissance and attacks on switches deeper in the network.

C. The Adversarial Cache Inference Problem

The goal of the attacker is to optimally use the above primitives to learn about and attack the flow table. This includes inferring the size C, the policy π, and other parameters, as well as using this information to design more efficient attacks. We refer to this problem as the *adversarial cache inference problem*, as solutions to this problem are also applicable to other types of caches as long as the same primitives are supported. In the sequel, we will interchangeably use ‘flow table’ and ‘cache’, and ‘flow rule’ and ‘content’.

Challenges: While simplified versions of the above problem have been tackled in prior works [16], [30], the solutions therein do not solve our problem. In [16], two different algorithms were proposed to infer the cache size under FIFO and LRU, respectively, but the policy must be known to apply the right algorithm. In [30], an algorithm was proposed to infer the size of an LRU cache, but it did not consider the problem that the replacement policy can be different and unknown. To our knowledge, this is the first work simultaneously addressing unknown cache size, unknown replacement policy, and interference from background traffic.

III. Joint Cache Size and Policy Inference

As size and policy are static parameters[2], the attacker can infer these parameters during off-peak hours when there is little background traffic, in preparation for larger attacks. In this section, we demonstrate the feasibility of such attacks by developing explicit size and policy inference algorithms.

A. Cache Size Inference

We will show that under mild conditions on the replacement policy, the cache size can be inferred without knowing the exact policy. Modeling the internal state of the cache by an ordered list of cached contents $(f_1, \ldots, f_C)$, where the content f_1 at the *head* of the list is the last to evict and the content f_C at the *tail* is the first to evict, we assume:

1) the newly entered content is always at the head;
2) if all the cached contents are only requested once, then the content at the tail is the first content entering the cache.

These conditions hold under both FIFO and LRU, two of the most commonly-used replacement policies. Moreover, they hold for a more general family of *permutation policies* [34]. For the ease of presentation, in the sequel we will use f_i to denote both a probe and the content requested by a probe.

Basic idea: Our key observation is that under the above conditions, the cache size can be revealed by a “forward-backward probing experiment” as follows. We illustrate the basic idea in Fig. 1 in the simplest case when there is no background traffic. Given an estimated cache size c, we generate c distinct probes, send them back-to-back in the

[1]Open vSwitch implements a rule replacement policy that under the basic setting (empty ‘groups’ column and same ‘importance’ for all flows), chooses the flow that expires soonest for eviction [32]. This implies a hybrid of LRU and FIFO in the presence of both hard and idle timeouts.

[2]Although in theory, SDN architecture allows the admin to remotely change configurations inside a switch, we find configurations such as flow table size and replacement policy to be fixed once the switch starts operating.

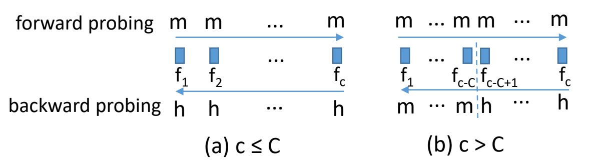


Fig. 1. Forward-backward probing ('h': hit; 'm': miss).

Algorithm 1: Robust Cache Size Estimation (RCSE)

```
input : Initial guess of cache size $c_0$, number of repetitions
        per experiment $n$, rule installation time $d_I$
output: Estimated cache size $\widehat{C}$
1  $\widehat{C} \leftarrow 0$;
2  $c \leftarrow c_0$;
3  while true do
4      foreach $i = 1, \ldots, n$ do
5          $\delta \leftarrow$ forward-backward-probing$(c, d_I)$;
6          $\widehat{C} \leftarrow \max(\widehat{C}, \delta)$;
7          if $\delta = c$ then
8              break;
9      if $\widehat{C} < c$ then
10         break;
11     else
12         $c \leftarrow 2 \times c$;
13 return $\widehat{C}$;
   forward-backward-probing$(c, d_I)$:
14 $\delta \leftarrow 0$;
15 foreach $i = 1, \ldots, c$ do
16     send probe $f_i$;
17 wait for $d_I$;
18 foreach $i = c, c-1, \ldots, 1$ do
19     send probe $f_i$;
20     if $f_i$ results in a hit then
21         $\delta \leftarrow \delta + 1$;
22     else
23         break;
24 return $\delta$;
```

order of $f_1, \ldots, f_c$ ("forward probing"), and wait for time d_I to ensure that the requested contents are installed. We then send these probes in the reverse order $f_c, \ldots, f_1$ ("backward probing"). In absence of background traffic, the cache state should be $(f_c, \ldots, f_{c-\min(c,C)+1})$ after the forward probing, with $f_{c-\min(c,C)+1}$ being the next to evict. Thus, the backward probing should yield hits for exactly $\min(c, C)$ probes.

This probing mechanism has been used to estimate the size of an LRU cache [30]. However, it was not realized then that the method applies to a broader set of policies, and there was no consideration of background traffic.

Algorithm: We now formalize this idea and augment it to guard against background traffic. The algorithm, *Robust Cache Size Estimation (RCSE)* (Algorithm 1), is based on a subroutine called `forward-backward-probing`(c, d_I) that performs the above probing experiment and returns the number of hits δ. Note that once we encounter a miss during backward probing, all the subsequent probes will lead to misses, and hence no further probe is needed.

The main algorithm repeats the experiment for each value of c for n times (lines 4–8), where n controls the tradeoff between the robustness against background traffic and the probing cost. Note that since only the largest number of hits is recorded (line 6), once we reach the upper bound c (line 7), there is no need to further repeat the experiment.

Overall, RCSE starts with an initial guess of the cache size $c = c_0$ (line 2), and then doubles it every n experiments (line 12) until the largest number of hits out of n experiments is still less than c (line 10), at which point this largest number of hits is returned as the estimated cache size.

Accuracy: We now analyze the accuracy of RCSE (Algorithm 1) in the presence of background traffic. The key observation is that background traffic can only cause underestimation of the cache size. Thus, if we repeat the experiment many times, then the largest number of hits across the experiments will converge to the true cache size.

To formalize this intuition, suppose that the background traffic is modeled as a Poisson process of rate λ.

Theorem III.1. Let T_c denote *the time to send c probes*. The error probability of RCSE (Algorithm 1) decays exponentially in n, and specifically,

$$\Pr\{\widehat{C} \neq C\} \leq (1 - e^{-\lambda T_{4(C-1)}})^n. \tag{1}$$

Proof. As background traffic can only cause underestimation of the cache size, $\Pr\{\widehat{C} \neq C\} = \Pr\{\widehat{C} < C\}$.

Since δ on line 5 of Algorithm 1 equals $\min(c, C)$ if there is no background traffic during the experiment, we have

$$\Pr\{\delta < \min(c, C)\} \leq 1 - e^{-\lambda T_{2c}}. \tag{2}$$

Let c^* be the final value of c. To have $\widehat{C} < C$, we must have (i) $c^* \leq C$ and $\widehat{C} < c^*$, or (ii) $c^* > C$ and $\widehat{C} < C$. In case (i),

$$\Pr\{\widehat{C} < C\} \leq \Pr\{\max_{i=1,\ldots,n} \delta_i < c^*\} \leq (1 - e^{-\lambda T_{2c^*}})^n, \tag{3}$$

where δ_i is the result of the i-th probing experiment for input c^*. In case (ii),

$$\Pr\{\widehat{C} < C\} \leq \Pr\{\max_{i=1,\ldots,n} \delta_i < C\} \leq (1 - e^{-\lambda T_{2c^*}})^n. \tag{4}$$

Although c^* is random, we must have $\widehat{C} = c^*/2$ in the second-to-last round in order to execute line 12. As $\widehat{C}$ is monotone increasing, having $\widehat{C} < C$ when the algorithm stops implies that $c^*/2 < C$ and hence $c^* \leq 2(C-1)$. This implies that

$$\Pr\{\widehat{C} < C\} \leq (1 - e^{-\lambda T_{2c^*}})^n \leq (1 - e^{-\lambda T_{4(C-1)}})^n, \tag{5}$$

which proves the theorem. □

We note that although the above analysis is done for Poisson traffic, *RCSE applies to other types of traffic too*. Intuitively, for background traffic following general renewal processes, we expect a similar exponential decay due to the repetition of the same experiments, except that the term $(1 - e^{-\lambda T_{4(C-1)}})$ will be replaced by the probability[3] of having at least one arrival during time $T_{4(C-1)}$. We have further verified the performance of RCSE on real traces (see Fig. 4–6).

Probing cost: Measured by the number of probes, the probing cost of RCSE is bounded as follows.

Theorem III.2. The number of probes required by RCSE is upper-bounded by

$$\begin{cases} n(7C - 2c_0 + 1) & \text{if } c_0 \leq C, \\ n(c_0 + C + 1) & \text{if } c_0 > C. \end{cases} \tag{6}$$

[3]Rigorously, it is the conditional probability of having the next arrival in time $T_{4(C-1)}$, conditioned on the elapsed time since the previous arrival.

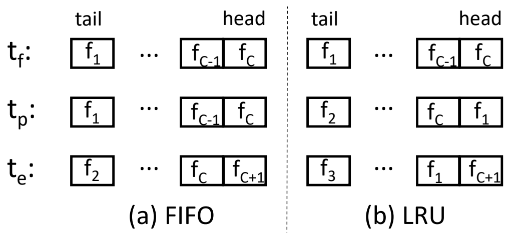


Fig. 2. Cache state during "flush-promote-evict-test"; t_i: d_I time after step i ($i = f$: flush, $i = p$: promote, $i = e$: evict).

Proof. The worst case is when the experiment for each value of c is repeated for n times, and c grows from the initial guess c_0 to the first value greater than C. Depending on the initial guess c_0, there are two cases:

Case 1. $c_0 \leq C$: The value of c grows as $c_0, 2c_0, \dots, 2^{m+1}c_0$, where $m = \lfloor \log(C/c_0) \rfloor$. For $c = c_0, \dots, 2^m c_0$, each experiment takes at most $2c$ probes. For $c = 2^{m+1}c_0$, each experiment takes at most $c + C + 1$ probes. Therefore, the total number of probes is at most

$$\sum_{i=0}^{m} nc_0 2^{i+1} + n(2^{m+1}c_0 + C + 1)$$
$$= n(6c_0 \cdot 2^m - 2c_0 + C + 1) \leq n(7C - 2c_0 + 1). \quad (7)$$

Case 2. $c_0 > C$: There is only one round with $c = c_0$, in which each experiment takes at most $c_0 + C + 1$ probes. Therefore, the total number of probes is at most $n(c_0 + C + 1)$. □

Theorem III.2 clearly characterizes the dependency of the probing cost on the initial guess of the cache size. It is easy to see that the minimum worst-case probing cost is $2n(C+1)$, achieved at $c_0 = C + 1$.

B. Replacement Policy Inference

Given the cache size C (estimated by RCSE), we further infer the replacement policy. We will first consider the special but realistic case where the policy is known to be either FIFO or LRU, and then discuss more general cases.

Basic idea: Our idea is to employ a four-step probing experiment that we call "flush-promote-evict-test". Fig. 2 illustrates this idea in the basic case when there is no background traffic. The first step ("flush") is to fill the cache with distinct contents $f_1, \dots, f_C$, which creates the same cache state under both candidate policies. The second step ("promote") is to introduce a difference in the cache state by requesting f_1 again. Under FIFO, f_1 will remain at the tail of the cache, but under LRU, f_1 will be promoted to the head of the cache. The third step ("evict") is to introduce a difference in the set of cached contents by requesting a new content f_{C+1}, which will evict f_1 under FIFO, but f_2 under LRU. The last step ("test") is to test this difference by requesting f_2 again after the eviction occurs. As the last probe will result in a hit under FIFO and a miss under LRU, we can detect which policy is used.

Algorithm: We now augment this procedure to improve its robustness. The algorithm, called *Robust Cache Policy Detection (RCPD)* (Algorithm 2), is based on a subroutine called `flush-promote-evict-test`(C, d_I) that performs the above experiment and returns the result of the last probe. The main algorithm repeats this experiment for N times, where N controls the tradeoff between the robustness and the probing

Algorithm 2: Robust Cache Policy Detection (RCPD)

```
input : Cache size $C$, number of experiments $N$
output: Detected cache replacement policy
1  foreach $i = 1, \dots, N$ do
2      if flush-promote-evict-test$(C, d_I)$ returns 'hit' then
3          return 'FIFO';
4  return 'LRU';
   flush-promote-evict-test$(C, d_I)$:
5  send $C$ distinct probes $f_1, \dots, f_C$ back to back;
6  send $f_1$ again;
7  send a new probe $f_{C+1}$;
8  wait for $d_I$;
9  send $f_2$ again;
10 return the result ('hit'/'miss') of the last probe;
```

cost. As shown next, only FIFO can result in 'hit', and hence once an experiment returns 'hit', we can detect the policy as FIFO and skip the remaining experiments. We will only detect the policy as LRU if all the experiments return 'miss'.

Accuracy: The presence of background traffic can only cause error in one way: under FIFO, contents installed due to background traffic may cause f_2 to be evicted before the test step, resulting in 'miss'; however, under LRU, background traffic will not change the experiment result, which is always 'miss'. Thus, by repeating the experiments many times and only detecting the policy as LRU if all the experiments report 'miss', our detected policy will converge to the ground truth.

To formalize this intuition, we again model the background traffic as a Poisson process of rate λ.

Theorem III.3. RCPD (Algorithm 2) will always detect LRU correctly, but may detect FIFO as LRU with a probability of $(1 - e^{-\lambda T_{C+3}})^N$, where T_c is defined as in Theorem III.1.

Proof. Under LRU, content f_2 must have been evicted before the test step (either by a subsequent probe or by the background traffic), and thus the subroutine will always return 'miss', which makes RCPD return 'LRU' with certainty. Under FIFO, Fig. 2 (a) has explained that if there is no background traffic during an experiment of "flush-promote-evict-test" (which lasts for T_{C+3}), then the subroutine will return 'hit'. Meanwhile, if there is any background traffic during the experiment which inserts at least one new content, then f_2 will be evicted before the test, and hence the subroutine will return 'miss'. Hence, the probability for the subroutine to return 'miss' under FIFO is the probability to have at least one arrival in the background traffic during an experiment, which equals $1 - e^{-\lambda T_{C+3}}$. The overall probability of mistakenly detecting FIFO as LRU via N independent experiments is thus $(1 - e^{-\lambda T_{C+3}})^N$. □

We note that similar to RCSE, *RCPD itself does not rely on the Poisson assumption*. In practice, we expect a similar exponentially decaying error probability for general types of traffic due to the repetition of experiments, which has been verified on real traces (see Fig. 8).

Probing cost: It is easy to see that the maximum number of probes required by RCPD is $N(C+3)$. Meanwhile, we show that in the special case of no background traffic, RCPD with $N = 1$ achieves the optimal probing cost.

Theorem III.4. In the case of no background traffic, the number of probes required by any algorithm to distinguish FIFO and LRU is at least $C+3$.

Proof. As the cache may be initially empty, at least C distinct probes are required to fill the cache, before which there will be no eviction and hence no invocation of the replacement policy. Moreover, to generate different responses (hits/misses), the cached contents must be different under the two policies, which requires at least one probe for a content already in the cache to put different contents at the tail of the cache, and at least one probe for a new content to evict the content at the tail. Finally, to detect the difference in cached contents, at least one more probe is needed, such that the requested content is cached under one policy but not cached under the other policy. Hence, the required number of probes is at least $C+3$. □

Discussion: The above idea can be extended to distinguish multiple candidate policies. Using binary detection algorithms like RCPD to differentiate two sets of policies via carefully designed probing sequences, we can gradually narrow down the candidate policies. We will also discuss another approach to handle multiple candidate policies in Section IV-A4.

IV. Intelligent Attacks

Having learned the cache (i.e., flow table) size and policy, we now demonstrate how this information can be used to launch attacks against legitimate users of the cache.

A. Intelligent Side Channel Attack

We show that under common assumptions, the attacker can use the information (size and policy) learned during reconnaissance to infer parameters of the background traffic. Unlike previous side channel attacks [17], [18], [19] that focus on inferring specific rules, we aim at inferring parameters useful for planning DoS attacks, such as the number of active flows and the individual flow rates.

1) Model of Background Traffic: We assume that the background traffic consists of F flows, each modeled as an independent Poisson process of rate λ_i, requesting content (i.e., rule) f_i ($i \in \{1,\ldots,F\}$). Let $\lambda := \sum_{i=1}^F \lambda_i$ denote the total rate. The goal of this attack is to jointly infer F and $(\lambda_i)_{i=1}^F$. In our evaluations, we further assume that the flow sizes follow the Zipf distribution with skewness α, i.e., $\frac{\lambda_i}{\lambda} = \frac{i^{-\alpha}}{\sum_{j=1}^F j^{-\alpha}}$, which reduces the unknown parameters to λ, F, and α. However, the Zipf assumption is not mandatory for our solution. The Poisson traffic model, a.k.a. the *Independent Reference Model (IRM)*, has been widely used in the literature. It is also known that the amount of traffic in different flows follows the Zipf distribution [35].

2) Background on TTL Approximation: We will leverage a recent advance in the caching literature, which approximately predicts the performance of caches based on their *Time-To-Live (TTL) approximations* [36], [37]. A TTL cache handles different contents independently by associating each cached content with a timer, and evicting the content when the timer expires, independently of the other contents. Although cache replacement policies may not follow TTL-based eviction, many commonly-used policies (e.g., LRU, FIFO, RANDOM, q-LRU, k-LRU) can be closely approximated by TTL-based policies in terms of hit probability [37].

In particular, the following has been shown for IRM [38]:

• *TTL Approximation for FIFO:* A FIFO cache can be approximated by a *non-reset TTL cache* with a constant timeout τ, i.e., each content entering the cache will be evicted after time τ, regardless of the request pattern. The hit probability of content f_i with request rate λ_i is given by

$$h_i^{\text{FIFO}} = \frac{\lambda_i \tau}{1+\lambda_i(d_I+\tau)}, \tag{8}$$

where τ, referred to as the *characteristic time*, is the solution to the following *characteristic equation*:

$$\sum_{i=1}^F \frac{\lambda_i \tau}{1+\lambda_i(d_I+\tau)} = C. \tag{9}$$

• *TTL Approximation for LRU:* An LRU cache can be approximated by a *reset TTL cache* with a constant timeout τ, i.e., each content in the cache will be evicted after an idle time of τ (i.e., not being requested for time τ). The hit probability of content f_i with request rate λ_i is given by

$$h_i^{\text{LRU}} = \frac{e^{\lambda_i \tau}-1}{\lambda_i d_I + e^{\lambda_i \tau}}, \tag{10}$$

where the characteristic time τ is the solution to the following characteristic equation:

$$\sum_{i=1}^F \frac{e^{\lambda_i \tau}-1}{\lambda_i d_I + e^{\lambda_i \tau}} = C. \tag{11}$$

Remark: We note that TTL approximations have been proved accurate for more general traffic models, e.g., renewal processes [37] and stationary ergodic processes [36], for which our approach also applies.

3) Attack Strategy: The idea is that by requesting a new content (via a carefully crafted probe [6]) at a selected rate, the attacker can measure the hit probability and compute the characteristic time by the TTL approximation. Plugging the computed characteristic time into the characteristic equation will yield an equation of the unknown parameters λ, F, and α. The attacker can repeat this procedure under different probing rates to obtain a system of equations, from which the three unknown parameters can be solved.

Specifically, the attacker will perform three experiments with different probing rates $\lambda_0^{(1)}$, $\lambda_0^{(2)}$, and $\lambda_0^{(3)}$. In the j-th experiment ($j=1,\ldots,3$), he will send probes requesting a new content according to an independent Poisson process of rate $\lambda_0^{(j)}$, and measure the hit probability $h_0^{(j)}$. If the policy is FIFO, the attacker can compute the characteristic time by

$$\tau_{\text{FIFO}}^{(j)} = \frac{h_0^{(j)}(1+\lambda_0^{(j)} d_I)}{\lambda_0^{(j)}(1-h_0^{(j)})}, \tag{12}$$

and then plug it into (9) to obtain an equation

$$\sum_{i=1}^F \frac{\lambda_i \tau_{\text{FIFO}}^{(j)}}{1+\lambda_i(d_I+\tau_{\text{FIFO}}^{(j)})} = C - h_0^{(j)}. \tag{13}$$

If the policy is LRU, the attacker can compute the characteristic time by

$$\tau_{\text{LRU}}^{(j)} = \frac{1}{\lambda_0^{(j)}} \log\left(\frac{1+h_0^{(j)}\lambda_0^{(j)}d_I}{1-h_0^{(j)}}\right), \tag{14}$$

and then plug it into (11) to obtain an equation

$$\sum_{i=1}^{F} \frac{e^{\lambda_i \tau_{\text{LRU}}^{(j)}}-1}{\lambda_i d_I + e^{\lambda_i \tau_{\text{LRU}}^{(j)}}} = C - h_0^{(j)}. \tag{15}$$

Since the obtained equations only contain three unknown variables λ, F, α (note: λ_i is a function of λ, F, α), the attacker can solve the three equations for their values.

Remark: The above approach can be extended to infer more parameters. For example, in the case of arbitrarily-valued $(\lambda_i)_{i=1}^F$ (assuming F is known), we can jointly infer all the λ_i's by solving F characteristic equations obtained from F probing experiments (using different probing rates), and the idea can be further extended to jointly infer F and $(\lambda_i)_{i=1}^F$ based on $\overline{F}+1$ probing experiments, assuming an upper bound $\overline{F}$ on F is known. However, we note that solving a large number of nonlinear equations will be numerically challenging. In practice, the IRM-Zipf model should be viewed as an approximation of the background traffic, and the above attack strategy as a way of inferring key parameters of the traffic (i.e., the number of active flows, the total flow rate, and the skewness of the flow size distribution). While real traffic may not satisfy the IRM-Zipf assumption, we have verified on real traces that such an approximation can enable accurate design of DoS attacks (see Fig. 16 (b)).

Moreover, to overcome the error in estimating the hit probabilities from measurements, the probing rates $\lambda_0^{(1)}$, $\lambda_0^{(2)}$, and $\lambda_0^{(3)}$ need to be widely different, and multiple probing flows can be sent concurrently, both for generating more diverse equations. We also observe that when $\lambda_i\tau$ is large, calculating the TTL approximation for LRU according to (10) may trigger an overflow error when evaluating $e^{\lambda_i\tau}$, in which case we can simply set $h_i^{\text{LRU}} \approx 1$. We have implemented these ideas and verified their efficacy (see Section V-B3).

4) Discussion on Size/Policy Inference: The TTL approximation also inspires an alternative approach to cache size and policy inference as explained below.

For policy inference, we (the attacker) can leverage a policy-agnostic characteristic time estimation algorithm in [30] to estimate the characteristic time. With this information, we can use (8, 10) to predict the hit probability for a probing flow of a given rate under each candidate policy. We then send the probing flow and measure its hit probability. The candidate policy for which the prediction is closer to the measured value is thus the inferred policy. This solution can be easily extended to the case of multiple candidate policies by comparing the measured hit probability to the predicted hit probability given by the TTL approximation for each of these policies.

For size inference, we note that the idea in Section IV-A3 can be easily extended to jointly infer C and the parameters (λ, F, α) of the background traffic, by conducting one more probing experiment to obtain one more characteristic equation. Note that this approach still requires knowledge of the policy, which can be obtained by the above method.

B. Intelligent Denial of Service (DoS) Attack

We now consider a type of DoS attack that aims at occupying the cache with contents not useful for legitimate users to lower their hit probabilities. Blunt versions of this attack have been studied in [6], [7], [8], [9]. However, we will show that knowledge of the cache size, policy, and load allows the attack to be designed more intelligently to achieve measurable damage with the minimum effort.

1) Attack Objective: We assume that before the attack, the attacker has learned the cache size C, the policy π, and the rates of background flows $(\lambda_i)_{i=1}^F$. We also assume that the attacker has crafted C_a distinct probes, each requiring a new content, by the method in [6]. The value of C_a should be large enough to occupy the cache (i.e., $C_a \geq C$) and small enough to avoid detection for brute-force attacks. We focus on the design of attack rates $(\lambda'_j)_{j=1}^{C_a}$, such that by sending each probe at rate λ'_j ($j=1,\ldots,C_a$), the attacker can lower the legitimate users' average hit probability to a target level $\bar{h}$ using the minimum total attack rate $\sum_{j=1}^{C_a}\lambda'_j$.

2) Optimal Attack Design: We will show how to optimally design the attack via the TTL approximation. For concrete analysis, we assume IRM for background traffic, but our approach can be extended to other traffic models using more general TTL approximations [36], [37].

First, we show that under the TTL approximation, the attack design problem can be reduced to a univariate optimization that assigns the same rate to all the attack flows.

Theorem IV.1. Under FIFO, an optimal design to make the TTL approximation of the legitimate users' average hit probability $\leq \bar{h}$ is to set $\lambda'_j \equiv \lambda_a$ ($j=1,\ldots,C_a$) for some constant λ_a. Under LRU, the same holds if d_I is sufficiently small.

Proof. Under FIFO, the TTL approximation of the legitimate users' average hit probability equals

$$\sum_{i=1}^{F} \frac{\lambda_i}{\lambda} \cdot \frac{\lambda_i\tau}{1+\lambda_i(d_I+\tau)}, \tag{16}$$

where the characteristic time τ is the solution to

$$\sum_{j=1}^{C_a} \frac{\lambda'_j\tau}{1+\lambda'_j(d_I+\tau)} + \sum_{i=1}^{F} \frac{\lambda_i\tau}{1+\lambda_i(d_I+\tau)} = C. \tag{17}$$

As $\frac{\lambda_i\tau}{1+\lambda_i(d_I+\tau)}$ is monotone increasing in τ, when bounding the total attack rate by B, the design that minimizes (16) should minimize τ, and hence minimize the second term on the left-hand side of (17). Thus, the optimal design should maximize the first term, i.e., be the optimal solution to

$$\max \sum_{j=1}^{C_a} \frac{\lambda'_j\tau}{1+\lambda'_j(d_I+\tau)} \tag{18a}$$

$$\text{s.t.} \sum_{j=1}^{C_a} \lambda'_j \leq B, \tag{18b}$$

$$\lambda'_j \geq 0, \qquad \forall j. \tag{18c}$$

As $\frac{\lambda'_j\tau}{1+\lambda'_j(d_I+\tau)}$ is a concave function in λ'_j, we see by Jensen's inequality that the optimal solution to (18) is $\lambda'_j \equiv B/C_a =: \lambda_a$. Thus, given any optimal design $\lambda'_j = \lambda^*_j$ ($j=1,\ldots,C_a$)

that makes (16) no more than $\bar{h}$ with the minimum total rate, the design $\lambda'_j \equiv \frac{1}{C_a}\sum_{i=1}^{C_a}\lambda_i^*$ $(j = 1, \ldots, C_a)$ also makes (16) no more than $\bar{h}$ with the same total rate, hence equally optimal.

Under LRU, the TTL approximation of the legitimate users' average hit probability equals

$$\sum_{i=1}^{F} \frac{\lambda_i}{\lambda} \cdot \frac{e^{\lambda_i \tau} - 1}{\lambda_i d_I + e^{\lambda_i \tau}}, \tag{19}$$

where the characteristic time τ is the solution to

$$\sum_{j=1}^{C_a} \frac{e^{\lambda'_j \tau} - 1}{\lambda'_j d_I + e^{\lambda'_j \tau}} + \sum_{i=1}^{F} \frac{e^{\lambda_i \tau} - 1}{\lambda_i d_I + e^{\lambda_i \tau}} = C. \tag{20}$$

Again, as $\frac{e^{\lambda_i \tau}-1}{\lambda_i d_I + e^{\lambda_i \tau}}$ is monotone increasing in τ, when bounding the total attack rate by B, the design that minimizes (19) should maximize the first term on the left-hand side of (20). Generally, $\frac{e^{\lambda'_j \tau}-1}{\lambda'_j d_I + e^{\lambda'_j \tau}}$ is neither concave nor convex in λ'_j. However, as $d_I \to 0$, it is reduced to $1 - e^{-\lambda'_j \tau}$, which is concave in λ'_j. Thus, by similar arguments as in the case of FIFO, having identical values for λ'_j's is optimal. $\square$

By Theorem IV.1, the attack design problem is reduced to finding the minimum value of λ_a, such that sending C_a attack flows, each at rate λ_a, can bring the legitimate users' average hit probability down to $\bar{h}$.

• *Attack rate design under FIFO:* By the TTL approximation (8), we know that to satisfy the upper bound on the average hit probability, the characteristic time τ needs to satisfy

$$\sum_{i=1}^{F} \frac{\lambda_i}{\lambda} \cdot \frac{\lambda_i \tau}{1 + \lambda_i (d_I + \tau)} \le \bar{h}. \tag{21}$$

Although this is effectively a high-order inequality of τ that is hard to solve in closed form, we observe that the left-hand side of (21) is monotone increasing in τ, and hence the solution must in the form of $\tau \le \tau^*$, where τ^* satisfies (21) with equality and can be found by a binary search. Then by (9), we have the following relationship between τ and λ_a:

$$\frac{C_a \lambda_a \tau}{1 + \lambda_a (d_I + \tau)} + \sum_{i=1}^{F} \frac{\lambda_i \tau}{1 + \lambda_i (d_I + \tau)} = C. \tag{22}$$

The left-hand side of (22) is monotone increasing in both τ and λ_a. Therefore, the minimum value of λ_a is achieved at the maximum value of τ, i.e., the optimal attack rate is

$$\lambda_a^{\text{FIFO}} = \frac{C - \sum_{i=1}^{F} \frac{\lambda_i \tau^*}{1 + \lambda_i (d_I + \tau^*)}}{C_a \tau^* - C(d_I + \tau^*) + \sum_{i=1}^{F} \frac{\lambda_i \tau^* (d_I + \tau^*)}{1 + \lambda_i (d_I + \tau^*)}}. \tag{23}$$

• *Attack rate design under LRU:* By the TTL approximation (10), we know that to satisfy the upper bound on the average hit probability, the characteristic time τ needs to satisfy

$$\sum_{i=1}^{F} \frac{\lambda_i}{\lambda} \cdot \frac{e^{\lambda_i \tau} - 1}{\lambda_i d_I + e^{\lambda_i \tau}} \le \bar{h}. \tag{24}$$

Again, solving (24) explicitly is difficult, but since its left-hand side is monotone increasing in τ, we know that the solution is in the form of $\tau \le \tau^*$, where τ^* satisfies (24) with equality and can be computed by a binary search. Then by (11), we have the following relationship between τ and λ_a:

$$\frac{C_a (e^{\lambda_a \tau} - 1)}{\lambda_a d_I + e^{\lambda_a \tau}} + \sum_{i=1}^{F} \frac{e^{\lambda_i \tau} - 1}{\lambda_i d_I + e^{\lambda_i \tau}} = C. \tag{25}$$

Since the left-hand side of (25) is monotone increasing in both τ and[4] λ_a, the minimum value of λ_a is achieved at the maximum value of τ. Plugging $\tau = \tau^*$ into (25) yields a transcendental equation of λ_a that can be solved numerically, and the solution is the optimal attack rate λ_a^{LRU}.

V. Simulations

We first evaluate the proposed reconnaissance and attack strategies via both synthetic and trace-driven simulations.

A. Simulation Setting

Synthetic simulation: We generate 100 instances of background traffic according to the model in Section IV-A1, where the number of flows $F = 5000$, the skewness $\alpha = 1.3$, and the total rate $\lambda = 0.01$ (packets/ms) during the size/policy inference, and $\lambda = 10$ (packets/ms) during the attacks. We set F according to the number of active flows at switches in data centers [39], and α according to the flow size distribution in a real trace [40]. We use a lower background traffic rate during size/policy inference as they are static parameters that can be inferred during off-peak hours[5]. We set the rate during attacks according to the average rate of the traces [40].

Trace-driven simulation: We generate background traffic according to the dataset UNI2 from a data center [40], which contains 9 packet traces. To perform more tests on these traces, we extract 5 subtraces from each trace by taking 10000 packets from a random point in time and repeating this for 5 times for each trace, which generates a total of 45 subtraces. The rates of these subtraces are between 9.84 and 11.31 packets/ms.

Common parameters: In both types of simulations, we set the cache size $C = 1000$ (rules) according to flow table sizes of commodity switches [5]. We generate probes according to an independent Poisson process of rate λ_a to be specified later, noting that only the *relative probing rate* λ_a/λ matters. We set the average new rule installation time to $d_I = 20$ (ms) according to measurements on commodity switches [24], [41]. In addition, we use the following default parameters for the proposed algorithms: $c_0 = 2$ and $n = 1$ for size inference, $N = 10$ for policy inference, and $C_a = C$ for DoS attack.

[4]The monotonicity in τ is easy to verify. For λ_a, taking the derivative of the left-hand side of (25) wrt λ_a yields $\frac{C_a}{(\lambda_a d_I + e^{\lambda_a \tau})^2}(d_I(\lambda_a \tau e^{\lambda_a \tau} - e^{\lambda_a \tau} + 1) + \tau e^{\lambda_a \tau})$. Since $xe^x - e^x + 1 \ge 0$ for all $x \ge 0$, $\lambda_a \tau e^{\lambda_a \tau} - e^{\lambda_a \tau} + 1 \ge 0$ and hence the derivative is non-negative, proving that the left-hand side of (25) is monotone increasing in λ_a.

[5]When the off-peak hours are unknown, the attacker can simply repeat RCSE and RCPD at evenly-spaced times of a day and aggregate the results to achieve accurate inference: RCSE can only underestimate the size, and therefore the maximum estimated size will be closer to the true size; RCPD will always detect LRU correctly, and therefore the true policy must be FIFO if at least one output of RCPD is 'FIFO'.

B. Results on Reconnaissance

1) Size Inference: We evaluate RCSE (Algorithm 1) in comparison with the size inference algorithms in [16] by the relative error of the estimated cache size: $|\widehat{C} - C|/C$. Note that RCSE is applicable without knowing whether the policy is FIFO or LRU ('policy-agnostic'), while [16] used two different algorithms for FIFO and LRU, and thus must know the policy ('policy-aware'). We also note that [16] originally set the stopping condition to be upon the eviction of an attacker's rule, but since the attacker will not know whether the evicted rule belongs to him without probing it, we replace this condition by the occurrence of an eviction, which can be inferred from the RTTs of probes as shown in [16]. As size inference occurs during off-peak hours, we set $\lambda = 0.01$ for background traffic and $\lambda_a = 1$ for probes (both in packet/ms) in synthetic simulations. In trace-driven simulations, we simulate the off-peak scenario by setting the relative probing rate to 100.

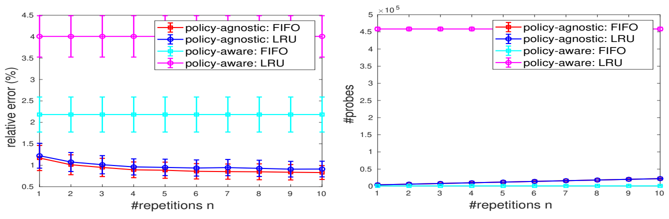
Fig. 3–4 show the results for varying the design parameter n in RCSE, and Fig. 5–6 show the results for varying the relative probing rate under $n = 1$. The algorithm proposed in [16] for FIFO ('policy-aware: FIFO') incurs about 500% of relative error for the traces and is hence omitted in Fig. 4–6 (a) for better visibility of the other curves. We see that: (i) although being policy-agnostic, RCSE closely matches or beats the accuracy of the existing size inference algorithms; (ii) increasing the number of repetitions n and increasing the probing rate can both improve the accuracy of RCSE; (iii) the performance of RCSE on the traces closely matches that on the Poisson traffic, which justifies the value of our performance analysis under the Poisson assumption, even though the traces exhibit non-Poisson properties (e.g., on-off patterns) [39]; (iv) compared to the existing algorithms in [16], RCSE has a comparable or significantly lower probing cost, which justifies its applicability in practice. For example, at $n = 1$ and a relative probing rate of 100, RCSE can achieve a relative error of 0.2% at a cost of about 4000 probes, i.e., probing at 1 packet/ms for 4 seconds.

We notice that the size inference algorithm proposed in [16] for FIFO ('policy-aware: FIFO') has a slightly increasing error in Fig. 5 (a) as the probing rate increases. This is because it suffers from not only underestimation error due to background traffic, but also overestimation error due to the delay d_I in installing new rules. Increasing the probing rate will reduce the underestimation error but increase the overestimation error. Under LRU ('policy-aware: LRU'), the underestimation error dominates and hence the overall error decreases with the probing rate; under FIFO ('policy-aware: FIFO'), the overestimation error dominates and hence the overall error increases with the probing rate.

2) Policy Inference: We then evaluate the accuracy of RCPD (Algorithm 2) in terms of error probabilities and probing costs under different ground-truth policies ('FIFO' and 'LRU'); see Fig. 7–10. Again, we assume the off-peak scenario as in the evaluation of RCSE.

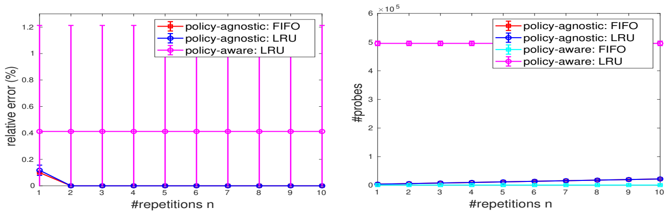
Compared to RCSE, RCPD requires a much higher probing rate. Intuitively, the probing rate should be C times higher than the background traffic rate, so that RCPD can finish one experiment of "flush-promote-evict-test" without being interfered. As $C = 1000$, the relative probing rate needs to be 1000. Under such a probing rate, Fig. 7–8 show that while a single experiment still has substantial error, repeating the experiment multiple times can reduce the error effectively. Fig. 9–10 show that increasing the probing rate is another effective way of reducing the error. Note that LRU is always inferred correctly as predicted in Theorem III.3. In contrast, inferring FIFO may cost fewer probes due to the early termination in RCPD (no early termination when inferring LRU, hence the deterministic probing cost under LRU). Despite requiring a high relative probing rate, RCPD is still feasible in practice as it only requires a short burst of probes, e.g., for $\lambda = 0.01$ packets/ms, RCPD only needs to probe at 10 packets/ms for one second (i.e., sending 10000 probes) to infer the policy with more than 95% accuracy.

Due to the requirement of a high relative probing rate, RCPD is not suitable when the background traffic rate is high. Nevertheless, we will show that in this case, the approach based on TTL approximation as described in Section IV-A4 can provide accurate policy inference at a low probing rate. We



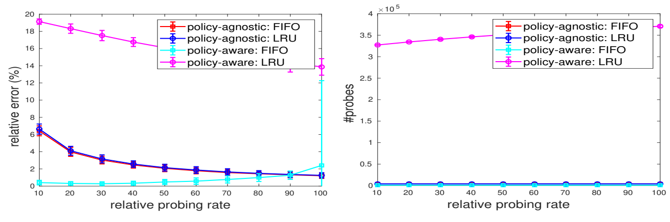
(a) relative error (b) probing cost

Fig. 3. Policy-agnostic size inference: vary #repetitions n (synthetic).



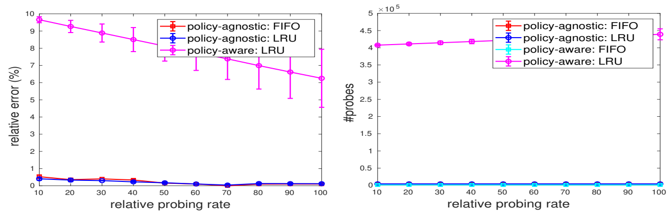
(a) relative error (b) probing cost

Fig. 4. Policy-agnostic size inference: vary #repetitions n (trace).



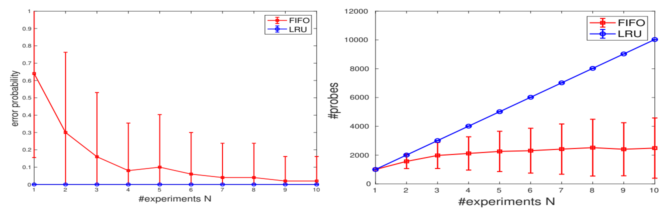
(a) relative error (b) probing cost

Fig. 5. Policy-agnostic size inference: vary probing rate (synthetic).



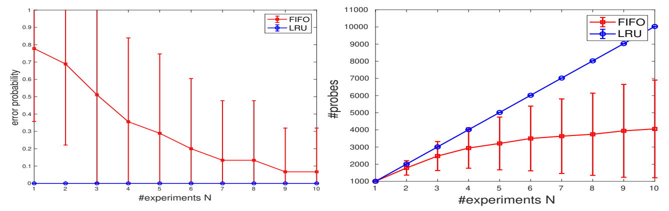
(a) relative error (b) probing cost

Fig. 6. Policy-agnostic size inference: vary probing rate (trace).



(a) error probability (b) probing cost

Fig. 7. Size-aware policy inference: vary #experiments N (synthetic).

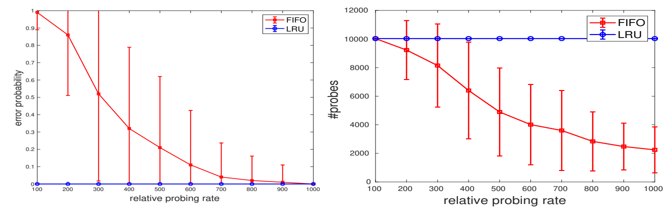


(a) error probability (b) probing cost

Fig. 8. Size-aware policy inference: vary #experiments N (trace).

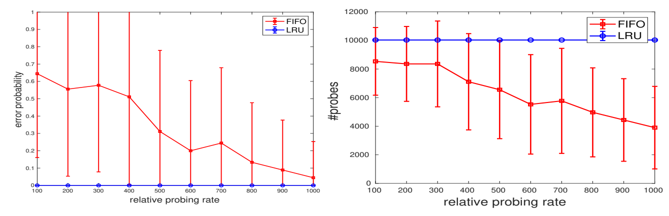
now evaluate this approach under a higher level of background traffic: $\lambda = 10$ packets/ms for synthetic traffic (recall that the trace rates are between 9.84 and 11.31 packets/ms).

To this end, we evaluate, in Table I, the hit ratio measured by the attacker ('measured'), together with the hit ratios predicted by the attacker based on the characteristic time estimated by the algorithm *Characteristic Time Estimation (CTE)* in [30] for each of the candidate policies ('predicted-FIFO/LRU'). The measured hit ratio is computed over 10 seconds for synthetic simulations and the duration of each trace (≈ 1000 seconds) for trace-driven simulations. In this experiment, we use the full traces (totally 9 of them) instead of the extracted subtraces, as CTE requires the cache to be refreshed during the experiment. We see that the measured hit ratio is much closer to the hit ratio predicted based on the true policy than to the hit ratio predicted based on the wrong policy, and hence can be used to identify the policy. In this experiment, we have set the probing rate to maximize the gap between the predicted hit ratios of the candidate policies, which is 0.0026–0.0028 packets/ms for synthetic traffic, and 0.0004–0.0005 packets/ms for the traces. Thus, to achieve accurate policy inference using this approach, the attacker only needs to probe at a very low rate.



(a) error probability (b) probing cost

Fig. 9. Size-aware policy inference: vary probing rate (synthetic).



(a) error probability (b) probing cost

Fig. 10. Size-aware policy inference: vary probing rate (trace).

TABLE I
MEASURED AND PREDICTED HIT RATIOS

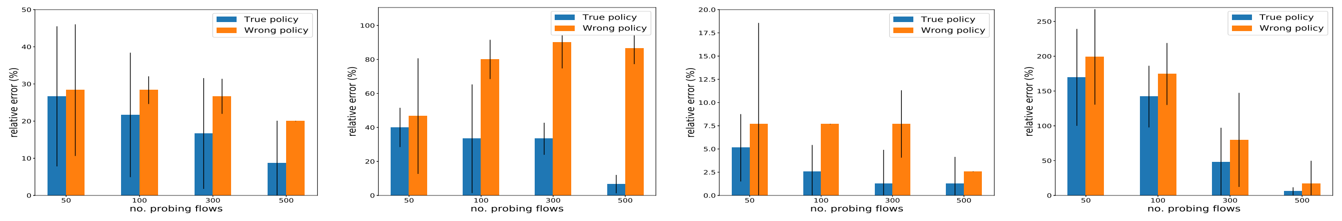
traffic	policy	measured	predicted-FIFO	predicted-LRU
synthetic	FIFO	0.7200	0.7156	0.9259
synthetic	LRU	0.8966	0.7158	0.9262
trace	FIFO	0.7628	0.7200	0.9246
trace	LRU	0.9271	0.8112	0.9868

3) Traffic Parameter Inference: We now evaluate the strategy in Section IV-A3 for inferring parameters of the background traffic during normal hours, where $\lambda = 10$ packets/ms for synthetic traffic and $\lambda \in (9.84, 11.31)$ packets/ms for the traces. For each realization of background traffic (synthetic or trace), we perform 4 probing experiments, where in the j-th experiment, lasting around 1000 seconds, we send multiple probing flows of rate $\lambda_0^{(j)}$ each. Note that the fourth experiment is needed for inferring the cache size C (see Section IV-A4). For synthetic traffic, we set the probing rates $(\lambda_0^{(1)}, \lambda_0^{(2)}, \lambda_0^{(3)}, \lambda_0^{(4)})$ to $(0.0004, 0.0009, 0.0052, 0.0100)$ under FIFO and $(0.00014, 0.0028, 0.0049, 0.0094)$ under LRU (unit: packets/ms); for the traces, we set the probing rates for each trace to achieve similar hit ratios as in the synthetic simulations, yielding rates of 0.00005–0.0014 packets/ms under FIFO and 0.00005–0.0012 packets/ms under LRU. We then repeat the experiments under different numbers of probing flows. In this experiment, we also use the full traces instead of the extracted subtraces, as using the approach in Section IV-A4 to infer the cache size C requires the total number of probing and background flows to be more than C (while each subtrace only has a few hundred flows).

We evaluate the accuracy in inferring each traffic parameter as the number of probing flows increases, as shown in Fig. 11–14 (a–c) (the error bar is missing for a bar when there is no variation), where 'true policy' means solving the characteristic time equations for the true underlying policy, and 'wrong policy' means solving the equations for the wrong policy. We see that: (i) the proposed strategy can infer parameters of the background traffic to reasonably good accuracy, i.e., within 10% of error for both synthetic traffic and the traces, (ii) the accuracy of the parameters inferred under the true policy significantly improves with the increase of the number of probing flows, and (iii) when sending sufficiently many probing flows, the error under the wrong policy is much higher than the error under the true policy, demonstrating the value of accurate policy inference. We note that although the aggregate probing rate to achieve good accuracy can be high in some experiments for synthetic traffic, the rate of a single probing flow is rather low, which allows the probing to be performed from distributed (compromised) hosts in a stealthy manner. We also observe that to achieve similar accuracy, the traces need a much lower probing rate than synthetic traffic[6].

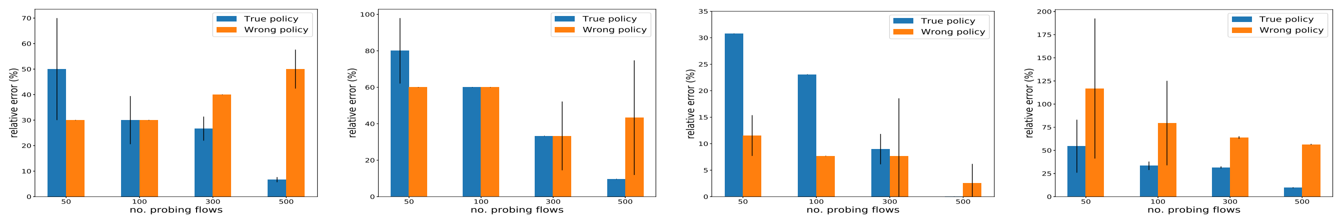
We further evaluate the accuracy of inferring the cache size C using the approach of joint parameter inference, as

[6]Technically, it is because we choose the probing rates to achieve similar hit ratios as in the synthetic simulations, and the on-off pattern in the traces [39] allows the probing flows to achieve a higher hit ratio during off periods. However, an explanation of why this leads to a satisfactory inference accuracy requires the analysis of TTL approximation for on-off traffic, which is beyond the scope of the current work.



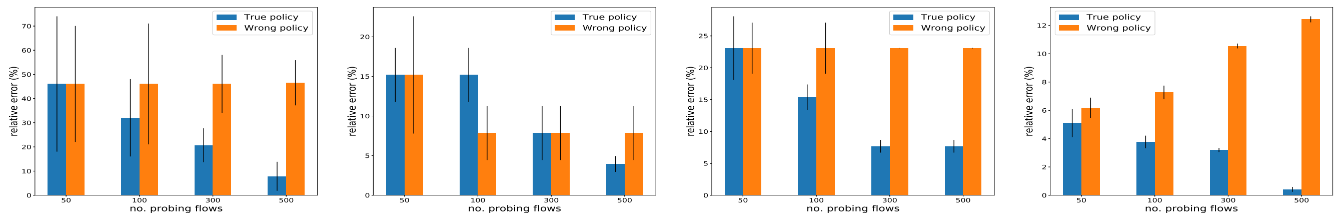
(a) estimate λ (b) estimate F (c) estimate α (d) estimate C

Fig. 11. Joint parameter inference under FIFO based on synthetic traffic



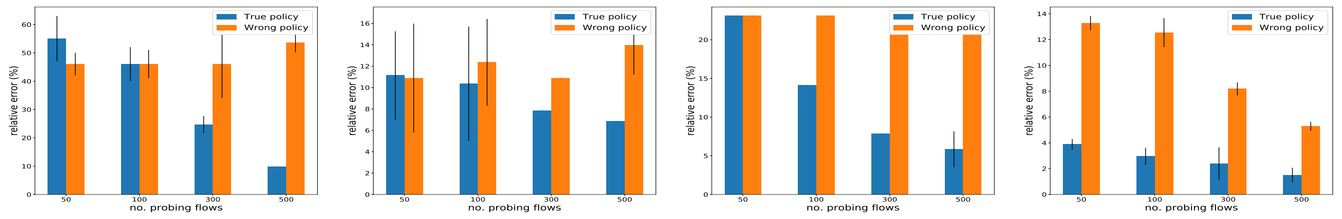
(a) estimate λ (b) estimate F (c) estimate α (d) estimate C

Fig. 12. Joint parameter inference under LRU based on synthetic traffic



(a) estimate λ (b) estimate F (c) estimate α (d) estimate C

Fig. 13. Joint parameter inference under FIFO based on trace



(a) estimate λ (b) estimate F (c) estimate α (d) estimate C

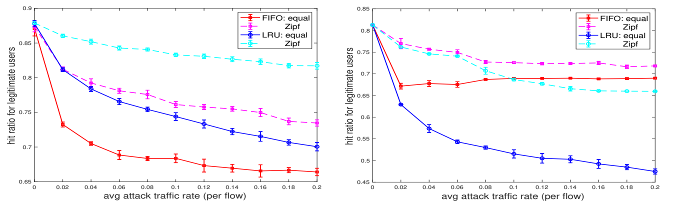
Fig. 14. Joint parameter inference under LRU based on trace

described in Section IV-A4. The results, shown in Fig. 11–14 (d), show that this approach can also infer C accurately, to within 10% of error for synthetic traffic and 2% of error for the traces. In comparison, RCSE has at least 50% error under the same total probing rate, due to the high background traffic rate. Again, knowing the true policy allows a much smaller error.

C. Results on DoS Attack

1) Effectiveness of Rate Allocation: We first verify the statement of Theorem IV.1 by comparing the equal-rate attack ('FIFO/LRU: equal'), where all the attack flows have the same rate, with an intuitive unequal-rate attack, where the attack flow rates follow the same Zipf distribution as the background flows ('FIFO/LRU: Zipf'). The result, shown in Fig. 15, confirms that the equal-rate attack is more effective, i.e., achieving a lower hit ratio for the legitimate users with the same total attack rate. We have obtained the same conclusion under other unequal-rate attacks. Note that there is a slight increase in the curve for 'FIFO: equal' based on the traces. We have verified that this is due to the new rule installation delay.

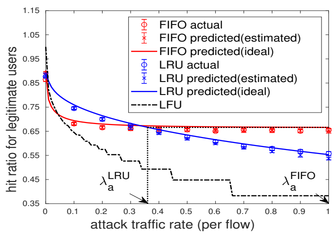
2) Accuracy of Rate Design: We now verify the accuracy of the designed attack rate. To this end, we gradually increase the



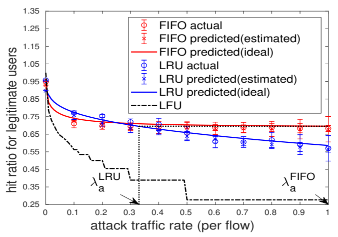
(a) synthetic (b) trace

Fig. 15. DoS attack: effectiveness of equal attack rate allocation

rate of each attack flow (totally $C_a = C$ attack flows). Fig. 16 shows a comparison of: the actual hit ratio for legitimate users from simulations ('FIFO/LRU actual'), the ideal prediction given by the TTL approximation based on the true parameters ('FIFO/LRU predicted (ideal)'), and the actual prediction obtained by the attacker, given by the TTL approximation based on the estimated parameters from previous experiments ('FIFO/LRU predicted (estimated)'). We see that (i) 'predicted (ideal)' closely follows the actual value, and (ii) 'predicted (estimated)' also closely follows the actual value, implying that our designed attack rate will be near-optimal, i.e., close to the minimum rate for achieving the targeted hit ratio.



(a) synthetic



(b) trace

Fig. 16. DoS attack: predicted and actual hit ratios for background traffic

Moreover, we can observe the following from these results:

Observation 1. *Knowing the policy is crucial for accurate attack design.* As illustrated in Fig. 16, for synthetic traffic, the minimum attack rate to achieve $\bar{h} = 0.67$ is $\lambda_a^{\text{LRU}} = 0.36$ under LRU but $\lambda_a^{\text{FIFO}} = 1$ under FIFO; for the trace, the minimum attack rate to achieve $\bar{h} = 0.70$ is $\lambda_a^{\text{LRU}} = 0.33$ under LRU but $\lambda_a^{\text{FIFO}} = 1$ under FIFO. This result demonstrates the need of knowing the policy in planning intelligent DoS attacks.

Observation 2. Contrary to the common belief that LRU is a better replacement policy than FIFO, *FIFO is actually better than LRU in terms of resilience to DoS attacks.* Fig. 16 shows that as the attack rate increases, the hit ratio for legitimate users under LRU decays quickly while this value under FIFO stays stable. This is because by design, LRU favors larger flows and hence will favor the attack flows as they become large; in contrast, FIFO treats all the flows equally, and is hence more resilient to large attack flows (in fact, the performance of FIFO is provably equivalent to that of the RANDOM policy that randomly selects the rules to evict [37]). To better illustrate this point, we add to Fig. 16 the hit ratio for Least Frequently Used (LFU), which deterministically stores the rules for the largest C flows. Despite being the optimal policy when there is no attack, LFU is even more vulnerable to DoS attacks, as it only serves the largest C flows, which may all be attack flows when the attack rate is sufficiently high. As many replacement policies are designed to approximate LFU [37], this indicates a need to redesign replacement policies for better attack resilience.

VI. Experiments

We repeat our key experiments in a virtual SDN created in Mininet [42], running OpenFlow 1.5, Open vSwitch 2.14.0, and Ryu controller. The virtual SDN runs in a virtual machine with 3.00GHz Intel Xeon Gold 6136 CPU and 64GB memory. In these experiments, we use large timeout values to rule out the effect of timeouts in order to focus on the effect of replacement policies[7].

The network topology is shown in Fig. 17, where h_1 is the only malicious host. Every link bandwidth is 10 Gbps. Different flows are created by varying the source/destination port numbers. Attack traffic consists of TCP packets sent from h_1 to h_2, while background traffic consists of TCP/UDP packets sent from h_3 to h_2 according to the traces introduced in Section V-A. The packets in the trace that trigger responses (e.g., TCP, ICMP) will be represented by TCP packets; the packets in the trace that do not trigger responses will be represented by UDP packets. The use of TCP packets as attack traffic allows h_1 to measure RTTs and infer whether the packets incur hits/misses in the flow table of the switch. However, this means that every attack flow will result in two flow rules being installed, one for itself and the other for the response flow, which is the primary difference between our experiments and simulations. To make the results in experiments and simulations comparable, we reduce the relative probing rate λ_a/λ in the experiments by half. Another difference is in packet rate: as the controlled experiment is slower in sending packets than production networks[8] , we slow down the packet sending rate by 10 times. This change does not affect our results as it is applied to both attack traffic and background traffic. The average rule installation delay in all Mininet experiments is around 0.9 ms.

Table II shows the experiment results for size inference with $\lambda_a/\lambda = 50$ and $n = 10$, which corresponds to the last data points in Fig. 4. Table III shows the experiment results for policy inference with $\lambda_a/\lambda = 500$ and $N = 10$, which corresponds to the last data points in Fig. 8. Both experiments are repeated for 90 Monte Carlo runs (two runs of attack traffic generation for each of the 45 subtraces). The experiment results closely match the simulation results, except that the required number of probes is reduced by half due to the use of TCP packets as probes that each requires two flow rules. We note that the experiments yield slightly lower accuracy than the simulations. This is because the background traffic also contains some TCP packets, which causes the same background traffic to result in the installation of more flow rules in the experiments than in the simulations, hence reducing the effective relative probing rate (recall that the probing rate is already reduced by half to offset the effect of TCP probes).

We then test the traffic parameter inference as in Fig. 13–14 (a–c) and the impact of DoS attack as in Fig. 16 (b) in

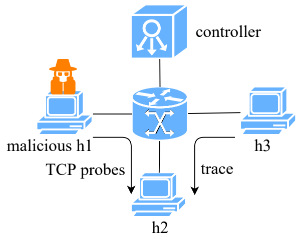


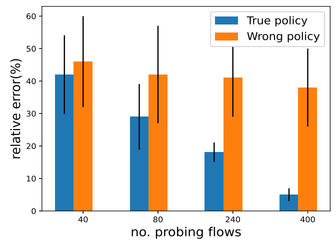
Fig. 17. Topology of the virtual SDN used in the experiments.

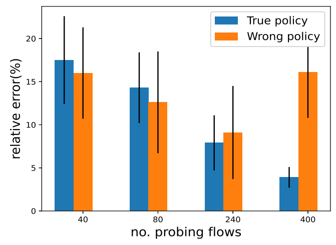
TABLE II
Experiment Results of Size Inference

metric	FIFO aware	FIFO agnostic	LRU aware	LRU agnostic
error mean (%)	421.27	0.1	0.47	0.1
error std (%)	344.18	0.04	0.81	0.05
#probes mean	4867	$1.151e4$	$2.360e5$	$1.121e4$
#probes std	24.89	2.786	2075	1.987

[7]Under LRU, we use an idle timeout of 65535s and no hard timeout; under FIFO, we use a hard timeout of 65535s and no idle timeout.

[8]This is because we adopt Tcpreplay [43] to send packets in our experiments and the max rate is around 1000 packets/ms when the link bandwidth is 10 Gbps. Considering that the max ratio of λ_a / $\lambda = 500$ during policy inference, we slow down the entire traffic by 10 times. In this way, the background traffic rate is changed from around 10 packets/ms to 1 packet/ms, and the maximum attack traffic rate is around 500 packets/ms.


(a) estimate λ


(b) estimate F

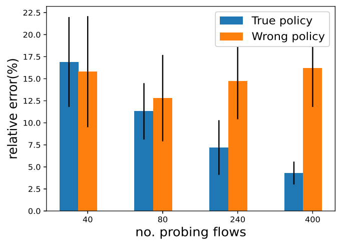
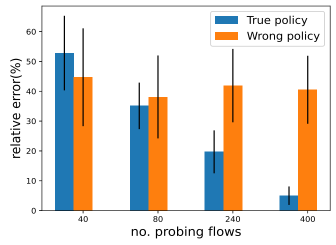
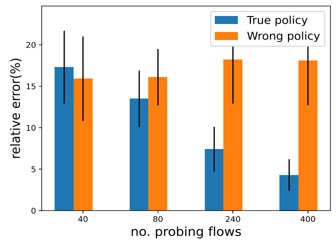

(c) estimate α

Fig. 18. Joint parameter inference under FIFO in Mininet


(a) estimate λ


(b) estimate F

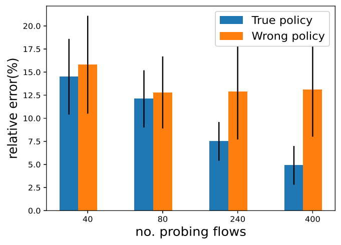

(c) estimate α

Fig. 19. Joint parameter inference under LRU in Mininet

TABLE III
EXPERIMENT RESULTS OF POLICY INFERENCE

metric	FIFO	LRU
error probability	0.0991	0
#probes (‘mean (std)’)	2004 (513)	5030 (0)

a single experiment, where the parameters (λ, F, α) of the background traffic are inferred from the first β fraction of a trace, and then used to predict the hit ratio under various DoS attacks for the rest of the trace; we set $\beta = 0.8$, but similar results are observed under other comparable β values.

To repeat the experiments on traffic parameter inference in Fig. 13–14 (a–c), we tune the probing rates based on the partial traces, resulting in probing rates of 0.00007–0.0016 packets/ms under FIFO and 0.00006–0.0015 packets/ms under LRU. The resulting inference errors are shown in Fig. 18–19, which closely match the results from simulations.

To repeat the DoS experiments in Fig. 16 (b), we set $C_a = 0.5C$ due to the use of TCP packets as attack traffic. The results under varying attack rate are shown in Fig. 20, which shows that the hit ratio predicted based on the traffic parameters estimated from the first part of the trace (’predicted (estimated)’) closely approximates the actual hit ratio measured from the second part of the trace (‘actual’). This validates that the inferred parameters about background traffic can be used to design effective DoS attacks (e.g., achieving a targeted hit ratio using the minimum attack rate), despite the natural dynamics in the background traffic. Meanwhile, the experiment results still support our previous observations. For example, we see that knowledge of the policy is crucial for accurate attack design, e.g., the minimum attack rate to achieve $\bar{h} = 0.70$ is $\lambda_a^{\text{LRU}} = 0.38$ under LRU but $\lambda_a^{\text{FIFO}} = 1$ under FIFO, which is similar to our observations from simulations.

VII. CONCLUSION

Observing that many studies of flow table security are based on simplistic attack models, we developed a model of

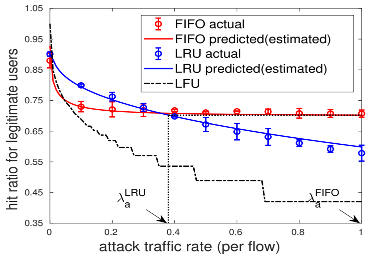


Fig. 20. DoS attack: predicted and actual hit ratios for background traffic in Mininet experiments

intelligent attackers that exploit the cache-like behaviors of the flow table to perform sophisticated reconnaissance and attacks. By developing explicit inference algorithms and attack strategies, we showed that an intelligent attacker can use simple primitives to accurately infer the internal parameters of the flow table (size, policy, and load characteristics), based on which he can plan attacks more efficiently. In demonstrating the capabilities of such attackers, we also identified the need of new designs and defenses, the detailed investigation of which is left to future work. Besides SDN, our results are also applicable to inference and attacks in general cache-based systems.

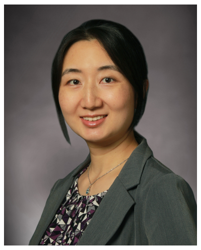
REFERENCES

[1] M. Yu, T. He, P. McDaniel, and Q. K. Burke, “Flow table security in SDN: Adversarial reconnaissance and intelligent attacks,” in *IEEE INFOCOM*, 2020.
[2] M. de Vivo, E. Carrasco, G. Isern, and G. O. de Vivo, “A review of port scanning techniques,” *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 2, pp. 41–48, April 1999.
[3] H. Kim and N. Feamster, “Improving network management with Software Defined Networking,” *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.
[4] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “Devoflow: Scaling flow management for high-performance networks,” *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 254–265, 2011.
[5] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, January 2015.
[6] J. Cao, M. Xu, Q. Li, K. Sun, Y. Yang, and J. Zheng, “Disrupting SDN via the data plane: A low-rate flow table overflow attack,” in *SECURECOMM*, 2017.

[7] J. Weekes and S. Nagaraja, "Controlling your neighbour's bandwidth for fun and for profit," in *Security Protocols*, 2017.
[8] Y. Qian, W. You, and K. Qian, "Openflow flow table overflow attacks and countermeasures," in *IEEE EuCNC*, 2016.
[9] B. Yuan, D. Zou, S. Yu, H. Jin, W. Qiang, and J. Shen, "Defending against flow table overloading attack in software-defined networks," *IEEE Transactions on Services Computing*, vol. 12, no. 2, pp. 231–246, March-April 2019.
[10] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks," in *ACM CCS*, November 2013.
[11] A. Akhunzada, E. Ahmed, A. Gani, M. K. Khan, M. Imran, and S. Guizani, "Securing software defined networks: taxonomy, requirements, and open issues," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 36–44, 2015.
[12] S. Hong *et al.*, "Poisoning network visibility in software-defined networks: New attacks and countermeasures." in *NDSS*, 2015.
[13] Z. Hu, M. Wang, X. Yan, Y. Yin, and Z. Luo, "A comprehensive security architecture for sdn," in *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*. IEEE, 2015.
[14] S. M. Mousavi and M. St-Hilaire, "Early detection of ddos attacks against sdn controllers," in *Computing, Networking and Communications (ICNC), 2015 International Conference on*. IEEE, 2015, pp. 77–81.
[15] Q. Burke, P. McDaniel, T. L. Porta, M. Yu, and T. He, "Misreporting attacks in software-defined networking," in *SECURECOMM*, 2020.
[16] Y. Zhou, K. Chen, J. Zhang, J. Leng, and Y. Tang, "Exploiting the vulnerability of flow table overflow in software-defined networks: Attack model, evaluation, and defense," *Security and Communication Networks*, pp. 1–15, January 2018.
[17] S. Achleitner *et al.*, "Adversarial network forensics in software defined networking," in *ACM Symposium on SDN Research (SOSR)*, 2017.
[18] J. Sonchack, A. Dubey, A. Aviv, J. Smith, and E. Keller, "Timing-based reconnaissance and defense in software-defined networks ," in *ACM ACSAC*, 2016.
[19] S. Liu, M. K. Reitner, and V. Sekar, "Flow reconnaissance via timing attacks on sdn switches," in *IEEE ICDCS*, 2017.
[20] J. Cao, Q. Li, R. Xie, K. Sun, G. Gu, M. Xu, and Y. Yang, "The CrossPath attack: Disrupting the SDN control channel via shared links," in *USENIX Security*, 2019.
[21] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: detecting security attacks in Software-Defined Networks," in *NDSS*, 2015.
[22] X.-N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Rules placement problem in openflow networks: A survey," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 2, pp. 1273–1286, 2016.
[23] "Open vSwitch 2.11.90 Documentation," https://docs.openvswitch.org/en/latest/.
[24] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Cacheflow: Dependency-aware rule-caching for software-defined networks," in *SOSR*, 2016.
[25] X. Li and W. Xie, "CRAFT: A cache reduction architecture for flow tables in software-defined networks," in *IEEE ISCC*, July 2017.
[26] J.-P. Sheu and Y.-C. Chuo, "Wildcard rules caching and cache replacement algorithms in software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 19–29, March 2016.
[27] H. Li, S. Guo, C. Wu, and J. Li, "FDRC: Flow-driven rule caching optimization in software define networking," in *ICC*, 2015.
[28] N. Laoutaris, G. Zervas, A. Bestavros, and G. Kollios, "The cache inference problem and its application to content and request routing," in *IEEE INFOCOM*, 2007.
[29] Y. Gao, L. Deng, A. Kuzmanovic, and Y. Chen, "Internet cache pollution attacks and countermeasures," in *IEEE ICNP*, 2006.
[30] M. Dehghan, D. Goeckel, T. He, and D. Towsley, "Inferring military activity in hybrid networks through cache behavior," in *MILCOM*, 2013.
[31] "OpenFlow switch specification," Open Networking Foundation, Version 1.5.1, March 2015.
[32] "Open vswitch database schema." [Online]. Available: https://man7.org/linux/man-pages/man5/ovs-vswitchd.conf.db.5.html
[33] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in *HotSDN*, 2013.
[34] A. Abel and J. Reineke, "Measurement-based modeling of the cache replacement policy," in *RTAS*, 2013.
[35] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "Leveraging Zipf's law for traffic offloading," in *SIGCOMM*, 2012.
[36] B. Jiang, P. Nain, and D. Towsley, "On the convergence of the TTL approximation for an lru cache under independent stationary request processes," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 3, no. 4, September 2018.
[37] M. Garetto, E. Leonardi, and V. Martina, "A unified approach to the performance analysis of caching systems," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 1, no. 3, May 2016.
[38] M. Dehghan, B. Jiang, A. Dabirmoghaddam, and D. Towsley, "On the analysis of caches with pending interest tables," in *ICN*, September 2015.
[39] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010.
[40] T. Benson, "Data set for IMC 2010 data center measurement," http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html.
[41] D. Y. Huang, K. Yocum, and A. C. Snoeren, "High-fideltiy switch models for software-defined network emulation," in *HotSDN*, August 2013.
[42] "Mininet." [Online]. Available: http://mininet.org/
[43] "Tcpreplay." [Online]. Available: https://tcpreplay.appneta.com/
[44] "Openflow tutorial," https://homepages.dcc.ufmg.br/~mmvieira/cc/OpenFlow Tutorial - OpenFlow Wiki.htm.
[45] T. Xie, N. Nambiar, and T. He, "Configurable rule replacement policies in SDN: Implementation in Open vSwitch," https://github.com/SophieCXT/SDN-Implementation-in-Open-vSwitch, 2021.
[46] H. Cui, G. O. Karame, F. Klaedtke, and R. Bifulco, "On the fingerprinting of software-defined networks," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 10, pp. 2160–2173, 2016.
[47] T. Xie, T. He, P. McDaniel, and N. Nambiar, "Attack resilience of cache replacement policies," in *IEEE INFOCOM*, May 2021.

Mingli Yu (M'20) received the B.S. degree in Computer Science from Tsinghua University and M.S. degree in Computer Science from Pennsylvania State University. He is currently a Ph.D. student in Computer Science at Pennsylvania State University, advised by Prof. Thomas La Porta. His research interest includes computer networking, network security and machine learning.

Tian Xie (M'20) received the B.Eng. degree in Software Engineering from Wuhan University, China in 2019, awarded as the 'Top Ten Outstanding Luojia Scholars of the Year 2018'. She is a Ph.D candidate in Computer Science at Pennsylvania State University, advised by Prof. Ting He. Her research interests lie in the area of network security and network modeling and optimization.



Ting He (SM'13) is an Associate Professor in the School of EECS at Pennsylvania State University, University Park, PA. Her work is in the broad areas of computer networking, network modeling and optimization, and statistical inference. Dr. He is a senior member of IEEE, an Associate Editor for IEEE Transactions on Communications (2017-2020) and IEEE/ACM Transactions on Networking (2017-2021), and an Area TPC Chair of IEEE INFOCOM (2021). She received multiple Outstanding Contributor Awards from IBM, and multiple paper awards from ITA, ICDCS, SIGMETRICS, and ICASSP.

Quinn Burke (M'20) received his B.S. and M.S. degrees in Computer Science from the Pennsylvania State University with a focus on computer security. He is currently pursuing a Ph.D. in Computer Science at Pennsylvania State University. His research interests include network and systems security, software-defined networking, and virtualization technologies.

VIII. Further Considerations

A. Rule Timeouts

One limitation of our previous model of the flow table is that it ignores the possibility that rules may come with idle and/or hard timeouts set by the controller, which can also cause rules to be removed from the flow table in addition to evictions. Nevertheless, most of our results remain applicable in the presence of timeouts as explained below. We will use τ_I to denote the idle timeout and τ_H the hard timeout.

1) Size inference: For RCSE (Algorithm 1) to work properly under timeout, it suffices for the subroutine `forward-backward-probing` to sense hits for the last C probes in the forward pass (when $c \geq C$). As sending the last C probes, waiting for the corresponding rules to be installed, and testing their existence in the flow table takes $T_{2C}+d_I$ time, where T_c denotes the time to send c probes, the timeouts will have no impact if $\min(\tau_I, \tau_H) > T_{2C} + d_I$, which is usually satisfied in practice. Specifically, under a realistic setting (e.g., $\lambda_a = 100$ packets/ms, $C = 1000$, and $d_I = 0.9$ ms), $T_{2C} + d_I$ is only 20.9 milliseconds, but $\min(\tau_I, \tau_H)$ is at least several seconds (e.g., 60 seconds for the default Mininet controller).

2) Policy inference: For RCPD (Algorithm 2) to work correctly under timeout, it suffices for the subroutine `flush-promote-evict-test` to finish before timeout occurs, which is satisfied if $\min(\tau_I, \tau_H) > T_{C+3} + d_I$. As $C \gg 1$, this condition will be satisfied if the above condition for size inference is satisfied.

3) Traffic parameter inference: Our solution for traffic parameter inference (Section IV-A) is based on TTL approximation, where the existing formulas have not considered timeouts. Nevertheless, as FIFO is already approximated by a TTL-based eviction policy with a non-reset timer τ (the characteristic time) [38], the existing TTL approximation formula (8) can be easily extended to accommodate a hard timeout τ_H:

$$h_i^{\text{FIFO}} = \frac{\lambda_i \min(\tau, \tau_H)}{1 + \lambda_i(d_I + \min(\tau, \tau_H))}, \tag{26}$$

as the hard timeout effectively sets another non-reset timer. Similarly, as LRU is already approximated by a TTL-based eviction policy with a reset timer τ [38], the existing formula (10) can be extended to accommodate an idle timeout τ_I:

$$h_i^{\text{LRU}} = \frac{e^{\lambda_i \min(\tau, \tau_I)} - 1}{\lambda_i d_I + e^{\lambda_i \min(\tau, \tau_I)}}, \tag{27}$$

as the idle timeout effectively sets another reset timer. For FIFO with idle timeout, LRU with hard timeout, or either policy with both types of timeouts, the TTL approximation will be a hybrid TTL-based policy with both a non-reset timer and a reset timer, which has not been analyzed before. We leave these cases to future work.

The above analysis implies that our approach of inferring parameters of background traffic in Section IV-A3 remains valid in the presence of hard timeout under FIFO or idle timeout under LRU, as long as the probing rates are designed such that the characteristic time given by (12) or (14) is no larger than the externally-imposed timeout value.

TABLE IV
EXPERIMENT RESULTS OF SIZE INFERENCE WITH TIMEOUTS

metric	FIFO aware	FIFO agnostic	LRU aware	LRU agnostic
error mean (%)	411.86	0.1	0.43	0.1
error std (%)	312.71	0.05	0.77	0.05
#probes mean	4920	$1.114e4$	$2.410e5$	$1.102e4$
#probes std	28.71	2.458	2082	1.756

TABLE V
EXPERIMENT RESULTS OF POLICY INFERENCE WITH TIMEOUTS

metric	FIFO	LRU
error probability	0.0987	0
#probes ('mean (std)')	2004 (488)	5030 (0)

4) DoS attack: Our results in Section IV-B2 remain valid in the presence of hard timeout under FIFO or idle timeout under LRU, as in these cases, the TTL approximation formulas (26) and (27) have the same form as before. Specifically, Theorem IV.1 still holds under FIFO with hard timeout τ_H for any meaningful attack target $\bar{h} < \sum_{i=1}^{F} \frac{\lambda_i}{\lambda} \cdot \frac{\lambda_i \tau_H}{1+\lambda_i(d_I+\tau_H)}$ (which is an upper bound on the average hit probability due to the hard timeout), as in this case (17) still holds, and hence (18) and the conclusion therein still hold. Similarly, Theorem IV.1 still holds under LRU with idle timeout τ_I for any $\bar{h} < \sum_{i=1}^{F} \frac{\lambda_i}{\lambda} \cdot \frac{e^{\lambda_i \tau_I}-1}{\lambda_i d_I + e^{\lambda_i \tau_I}}$ (upper bound on the average hit probability due to the idle timeout). For the same reason, the attack design methods proposed in Section IV-B2 remain valid under any meaningful attack target $\bar{h}$. We leave attack design under other combinations of policy and timeout to future work.

5) Experiments: We validate the impact of timeouts by repeating the experiments in Section VI under the timeouts of $\tau_I = 60$s and $\tau_H = 600$s. Although the timeouts have different default values for different controllers, 60s of idle timeout is the default choice by Mininet [44] (no hard timeout), and we add a hard timeout to test the effect of both timeouts. However, the default policy in Open vSwitch is neither LRU nor FIFO in the presence of both timeouts. Therefore, we modify its code to implement the desired replacement policies, available at [45]. We repeat the experiments in Section VI, as shown in Tables IV–V and Fig. 21–23 (ignoring the idle timeout under FIFO and the hard timeout under LRU in computing the TTL approximation). The results are very similar to those in Section VI, implying that our solutions remain applicable under realistic timeout values.

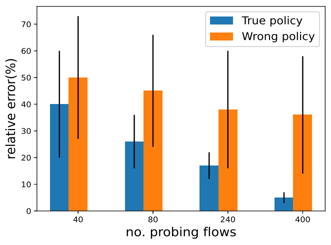
B. Possible Defenses

As the proposed attacks are based on the two primitives in Section II-B that have been validated on current SDN implementations, one strategy to defend against the identified attacks is to modify the implementation to invalidate at least one of the primitives.

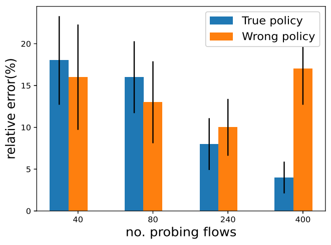
One approach is to make all the hits and misses indistinguishable in terms of response time. However, such a strategy comes at a high performance cost, as it will slow down packets that result in hits (which are normally the majority of cases) to the level of packets that result in misses. In [46], an alternative strategy was proposed, where after a flow has not been active (i.e., generating packets) for a period of time T_{th}, the switch will delay new packets of this flow within a small window W to mimic the response times under table misses, even if these packets result in hits. From the attacker's perspective, adopting this strategy on top of an existing replacement policy is equivalent to adding an "idle timeout" of T_{th} and a "rule installation delay" of $d_I = W$ after the first miss following a timeout, as packets arriving during this delay will be detected as misses. As explained in Section VIII-A, this defense has no impact on our size/policy inference algorithms as long as $T_{th} > T_{2C} + d_I$. Furthermore, it will not affect our methods for traffic parameter inference or DoS attack under LRU as long as T_{th} is no smaller than the characteristic time.

To test the efficacy of this defense, we repeat the experiments of RCSE and RCPD in Section VI while implementing the strategy in [46] with $W = 100$ ms (as recommended by [46]) and various values of T_{th}. The results, shown in Fig. 24–25, demonstrate a tradeoff between defense efficacy and communication performance, measured by the percentage of legitimate packets resulting in hits that are delayed by the defense mechanism. To notably increase the error of size/policy inference, T_{th} has to as small as a few milliseconds, causing a significant portion of legitimate packets to be delayed. Fundamentally, timing attacks (including the proposed attacks) exploit the performance difference between packets handled within the data plane and those involving the control plane, and thus cannot be completely eliminated as long as such difference exists. Techniques designed to destroy the statistics used in a specific attack (e.g., [46]) may not be effective for other attacks. It remains open whether there exists a universal defense with tolerable performance penalty against all possible timing attacks in SDN.

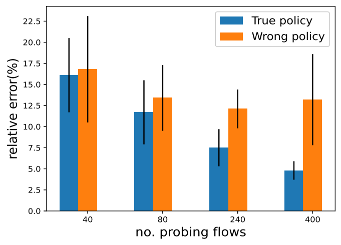
When DoS attacks are launched, the most effective defense is to quickly detect the attacks (e.g., [21]) and block the attack traffic. In addition, it is also desirable to have graceful performance degradation when the detection is not successful. Motivated by Observation 2 in Section V-C, our recent results [47] suggested that different policies degrade differently under a given attack strategy, wherein an adaptive policy selection scheme was proposed to select the most resilient policy for the perceived type of attack.



(a) estimate λ

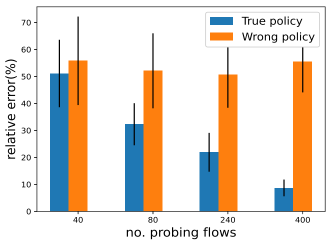


(b) estimate F

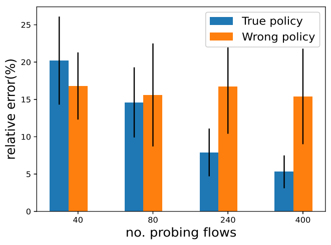


(c) estimate α

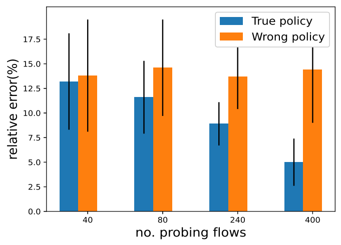
Fig. 21. Joint parameter inference under FIFO in Mininet with timeouts



(a) estimate λ



(b) estimate F



(c) estimate α

Fig. 22. Joint parameter inference under LRU in Mininet with timeouts

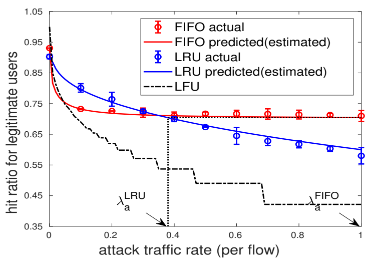
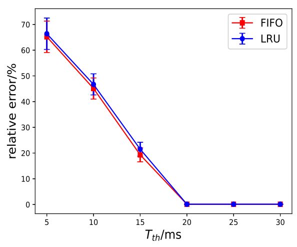
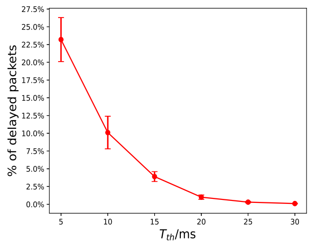


Fig. 23. DoS attack: predicted and actual hit ratios for background traffic in Mininet with timeouts

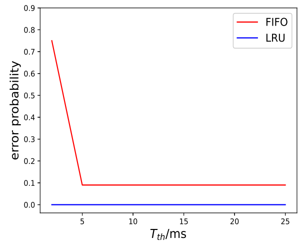


(a) relative error

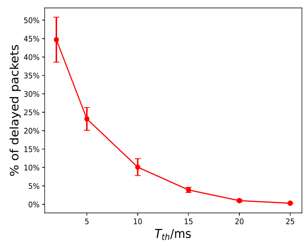


(b) delayed legitimate packets

Fig. 24. Defense: policy-agnostic size inference under varying T_{th} (n=10, $\lambda_a/\lambda = 50$)



(a) error probability



(b) delayed legitimate packets

Fig. 25. Defense: size-aware policy inference under varying T_{th} ($N = 10$, $\lambda_a/\lambda = 500$)