Flow Table Security in SDN: Adversarial Reconnaissance and Intelligent Attacks

Mingli Yu, Ting He, Patrick McDaniel, and Quinn K. Burke
Pennsylvania State University, University Park, PA, USA. Email: {mxy309,tzh58,pdm12,qkb5007}@psu.edu

***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 led to massive adoption of SDN.

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 [1] by using agile structures and policies [2], 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 [3] 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 [4].

This vulnerability has been explored to launch various flow table overflow attacks [5], [6], [7], [8] and control plane saturation attacks [9]. 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. For example, the controller cannot ensure that the data plane executes control instructions as expected [10], controller logic can be exploited to subvert some controllers [11], and the controller presents a single point of failure [12]. However, the weakest component of SDN is the communication bottleneck between the controller and the switches [3]. This bottleneck can be exploited in both active attacks [9], [5], [6], [7], [8] and adversarial reconnaissance [13], [14], [15], [16], [17]. We will exploit this bottleneck for joint reconnaissance and attack. Unlike previous works [15], [16], [17] 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.

Flow table management: Existing techniques can be classified into: (i) rule replacement, (ii) rule compression, and (iii) rule distribution [18]. Among the three, approach (i) is the most 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 [19], and hardware switches usually employ FIFO [6].

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 [20], [21], [22], and the approximation of Bélády's optimal replacement policy [23]. In contrast, very few studies have considered the security of rule replacement policies in an *adversarial environment*.

This research was partly sponsored by the U.S. Army Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

In [6], [7], [8], the feasibility of filling the flow table with the attacker's rules was demonstrated, but the attack was unintelligent. In [5], 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 [14], 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 general caches, but even then existing results are very limited. In [24], 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 [25], two attacks were proposed to replace popular contents in the cache by unpopular contents, but their parameters were not intelligently designed. In [26], algorithms were proposed to infer the size and other parameters of an LRU cache, but 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 and trace-driven simulations that the proposed solution can achieve accurate reconnaissance (with more than 95% accuracy) and efficient attacks despite interference from background traffic.

Roadmap. Section II introduces our models and problem formulation. Sections III–IV present our solutions for reconnaissance and attack design. Section V validates the proposed solutions via simulations. Section VI concludes the paper.

II. Problem Formulation

A. Flow Table Model

We model the flow table at the target switch as a *cache of flow rules*. According to the OpenFlow Switch Specification 1.5.1 [27], 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 [5] 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 [4]. The replacement policy π also comes from a small set of candidate policies. For example, Open vSwitch [19] 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. A study [6] found that certain hardware switches use FIFO. While there are more sophisticated policies proposed by researchers, e.g., [20], [21], [22], [23], 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 flow table hit or miss. This capability has been demonstrated in previous studies [28], [14]. For example, the attacker can measure the round-trip time (RTT) of the probe and compare it with a threshold learned from sure misses (RTTs of packets with randomly generated source IP addresses) and sure hits (RTT for 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 [5] 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

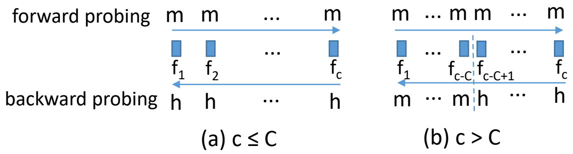


Fig. 1. Forward-backward probing ('h': hit; 'm': miss).

installation time. We assume that the attacker can estimate d_I by measuring the differences between RTTs of hits and misses.

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 [14], [26], the solutions therein do not solve our problem. In [14], 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 [26], 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, 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* [29]. 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 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.

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 $\delta$;
```

This probing mechanism has been used to estimate the size of an LRU cache [26]. 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,

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

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

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

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),

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

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

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

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

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

which proves the theorem. □

We note that although the above analysis is done for Poisson traffic, our algorithm applies to other types of traffic too.

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

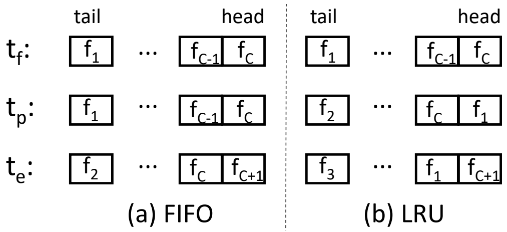
$$\begin{cases} n(7C - 2c_0 + 1) & \text{if } c_0 \leq C, \\ n(c_0 + C + 1) & \text{if } c_0 > C. \end{cases} \quad (6)$$

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 $c_0, 2c_0, \ldots, 2^{m+1}c_0$, where $m = \lfloor \log(C/c_0) \rfloor$. For $c = c_0, \ldots, 2^m c_0$, 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 $n(c_0 + C + 1)$. □



(a) FIFO
(b) LRU

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).

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 $f_1, \ldots, f_C$, 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 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

Algorithm 2: Robust Cache Policy Detection (RCPD)

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

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. Note that RCPD does not rely on this assumption.

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 $(1 - e^{-\lambda T_{C+3}})^N$. □

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 [15], [16], [17] 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 [30].

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* [31], [32]. 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 [32].

In particular, the following has been shown for IRM [33]:

- *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 [32] and stationary ergodic processes [31], for which our approach also applies.

3) Attack Strategy: The idea is that by requesting a new content (via a carefully crafted probe [5]) 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.

This approach can be extended to infer more parameters (e.g., arbitrarily-valued $(\lambda_i)_{i=1}^F$) by performing more probing experiments. 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.

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 [26] 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 more accurate is 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 value 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 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 [5], [6], [7], [8]. 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 [5]. 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 [31], [32].

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. □

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

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

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 $\tau \leq \tau^*$, 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

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

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 $\tau \leq \tau^*$, 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[1] 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}.

[1] 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 $x e^x - e^x + 1 \geq 0$ for all $x \geq 0$, $\lambda_a \tau e^{\lambda_a \tau} - e^{\lambda_a \tau} + 1 \geq 0$ and hence the derivative is non-negative, proving that the left-hand side of (25) is monotone increasing in λ_a.

V. Performance Evaluation

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

A. Simulation Setting

Synthetic simulation: We generate 20 instances of background traffic according to the model in Section IV-A1, where the number of flows $F = 5000$, the skewness $\alpha = 0.9$, 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 [34], and α according to the flow size distribution in a real trace [35]. We use a lower background traffic rate for size/policy inference as they are static parameters that can be inferred during off-peak hours. We set the rate during attacks according to the average rate of the trace [35].

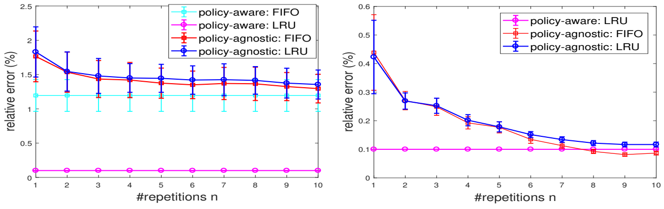
Trace-driven simulation: We generate background traffic according to the packet trace UNI1 from a data center [35], from which we extract 20 subtraces by taking 10000 packets from a random point in time and repeating this for 20 times.

Common parameters: In both types of simulations, we set the cache size $C = 1000$ (rules) according to flow table sizes of commodity switches [4]. 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 $d_I = 20$ (ms) according to measurements on commodity switches [20], [36]. 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.

B. Results on Reconnaissance

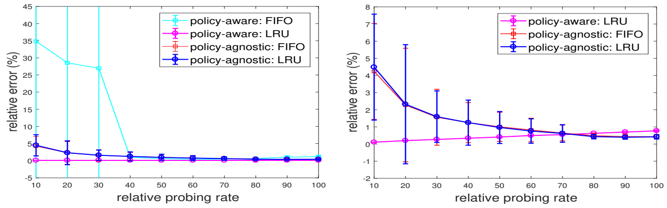
1) Size Inference: We evaluate RCSE (Algorithm 1) in comparison with the size inference algorithms in [14] by the relative error of the estimated cache size: $|\widehat{C} - C|/C$. Note that RCSE is applicable without knowing whether the policy is FIFO or LRU ('policy-agnostic'), while [14] used two different algorithms for FIFO and LRU, and thus must know the policy ('policy-aware'). 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 shows the results for varying the design parameter n in RCSE, and Fig. 4 shows the results for varying the relative probing rate under $n = 1$. The algorithm proposed in [14] for FIFO ('policy-aware: FIFO') incurs about 1000% of relative error for the trace and is hence omitted in Fig. 3–4 (b) for better visibility of the other curves. We see that: (i) although policy-agnostic, RCSE closely matches the accuracy of the existing size inference algorithm under LRU while significantly outperforming that under FIFO; (ii) increasing the number of repetitions n and increasing the probing rate can both improve the accuracy of RCSE; (iii) the inference by RCSE is more accurate for the trace. The last phenomenon is because that the trace exhibits an on-off pattern [34], where the off periods provide opportunities for accurate inference.



(a) synthetic (b) trace

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

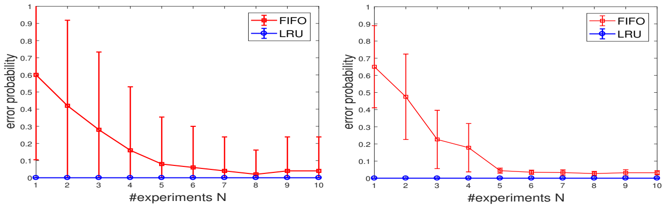


(a) synthetic (b) trace

Fig. 4. Policy-agnostic size inference: vary probing rate

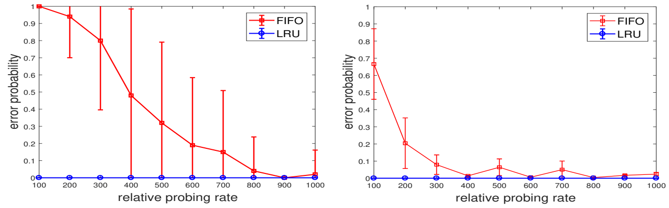
2) Policy Inference: We then evaluate the accuracy of RCPD (Algorithm 2) in terms of the error probabilities under different ground-truth policies ('FIFO' and 'LRU'); see Fig. 5–6. To our knowledge, RCPD is the first of its kind.

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. Fig. 5, under such a probing rate, shows that while a single experiment still has substantial error, repeating the experiment multiple times can reduce the error effectively. Fig. 6 shows 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. 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 to infer the policy with more than 95% accuracy.



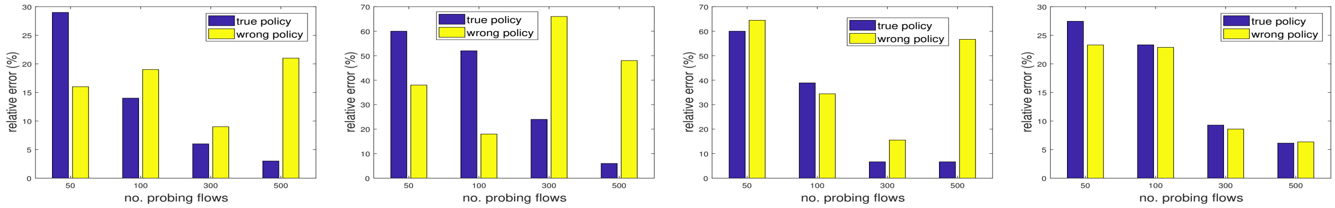
(a) synthetic (b) trace

Fig. 5. Size-aware policy inference: vary #experiments N



(a) synthetic (b) trace

Fig. 6. Size-aware policy inference: vary probing rate



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

Fig. 7. Joint parameter inference under LRU

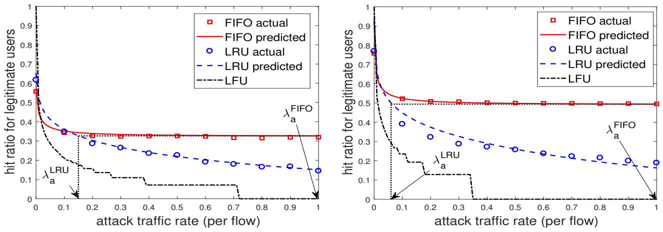
3) Traffic Parameter Inference: We now evaluate the strategy in Section IV-A3 for inferring parameters of the background traffic during normal hours. We set the probing rates $(\lambda_0^{(1)}, \lambda_0^{(2)}, \lambda_0^{(3)})$ to $(0.004, 0.012, 0.324)$ under FIFO and $(0.003, 0.006, 0.023)$ under LRU. In the j-th experiment ($j = 1, \ldots, 3$), lasting 100 seconds, we send multiple probing flows, each of rate $\lambda_0^{(j)}$. Due to space limitation, we only show the results on synthetic traffic with $\lambda = 10$ packets/ms under LRU; similar results exist under FIFO.

We evaluate the accuracy in inferring each parameter as the number of probing flows increases, as shown in Fig. 7 (a–c), where 'true policy' is the result from solving the characteristic time equations for the true policy, and 'wrong policy' is the result from solving the equations for the wrong policy. We see that: (i) the proposed strategy can infer parameters of the background traffic to within 5% of error, (ii) the inference accuracy significantly improves with the number of probing flows, and (iii) the error is much higher under the wrong policy when sending sufficiently many probing flows. We note that although the aggregate probing rate to achieve good accuracy is high, the rate of a single probing flow is well within the normal range, which allows the probing to be performed in a distributed manner (as in DDoS attacks).

We further evaluate the accuracy of inferring the cache size C using this approach, as discussed in Section IV-A4. The results, shown in Fig. 7 (d), show that this approach can also infer C to within 5% of error, but interestingly, the errors under the true policy and the wrong policy are comparable. In comparison, RCSE has $\approx 50\%$ of error in this case (not shown) due to the high background traffic rate.

C. Results on DoS Attack

We have verified Theorem IV.1, i.e., assigning equal rate to all the attack flows is the most effective (plots omitted due to space limitation). We now verify the accuracy of the designed attack rate. Fig. 8 shows both the predicted hit ratio for legitimate users ('FIFO/LRU predicted') given by the TTL approximation, and the actual value ('FIFO/LRU actual') from simulations, as we send $C_a = C$ attack flows and slowly increase the rate of each flow. Note that the attack flows can be sent from distributed locations (i.e., DDoS). We see that the prediction closely follows the actual value, and hence our designed attack rate will be near-optimal, i.e., close to the minimum rate for achieving the targeted hit ratio. We also have two important observations from this result:



(a) synthetic (b) trace

Fig. 8. DoS attack: accuracy of attack rate design

Observation 1. *Knowing the policy is crucial for accurate attack design.* As illustrated in Fig. 8, for synthetic traffic, the minimum attack rate to achieve $\bar{h} = 0.33$ is 0.15 under LRU but 1 under FIFO; for the trace, the minimum attack rate to achieve $\bar{h} = 0.5$ is 0.06 under LRU but 1 under FIFO. This demonstrates the need of knowing the policy in planning intelligent DoS attacks. Meanwhile, we have verified that the prediction accuracy is not sensitive to errors in the inferred cache size and traffic parameters.

Observation 2. Contrary to the popular belief that LRU is a better replacement policy than FIFO, *FIFO is actually better than LRU in terms of resilience to DoS attacks.* Fig. 8 shows that as the attack rate increases, the hit ratio of LRU decays quickly while the hit ratio of 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. To better illustrate this point, we add to Fig. 8 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 can all be attack flows. As many replacement policies are designed to approximate LFU [32], this indicates a need to redesign replacement policies for better attack resilience.

VI. Conclusion

Observing that many studies of flow table security are based on simplistic attack models, we develop a model of 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 show that an intelligent attacker can use simple primitives to accurately infer the internal parameters of the flow table (size, policy, and load), based on which he can plan attacks more efficiently. In demonstrating the capabilities of such attackers, we also identify the need of new designs and defenses.

References

[1] 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.

[2] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[3] 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.

[4] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, 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.

[5] 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.

[6] J. Weekes and S. Nagaraja, "Controlling your neighbour's bandwidth for fun and for profit," in *Security Protocols*, 2017.

[7] Y. Qian, W. You, and K. Qian, "Openflow flow table overflow attacks and countermeasures," in *IEEE EuCNC*, 2016.

[8] 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.

[9] 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.

[10] T. Sasaki, C. Pappas, T. Lee, T. Hoefler, and A. Perrig, "SDNsec: Forwarding accountability for the SDN data plane," in *IEEE ICCCN*, August 2016.

[11] C. R opke and T. Holz, "SDN Rootkits: Subverting network operating systems of software-defined networks," in *RAID*, November 2015.

[12] M. Azab and J. A. Fortes, "Towards proactive sdn-controller attack and failure resilience," in *IEEE ICNC*, 2017, pp. 442–448.

[13] S. Shin and G. Gu, "Attacking Software-Defined Networks: a first feasibility study," in *ACM HotSDN*, August 2013.

[14] 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.

[15] S. Achleitner *et al.*, "Adversarial network forensics in software defined networking," in *ACM Symposium on SDN Research (SOSR)*, 2017.

[16] J. Sonchack, A. Dubey, A. Aviv, J. Smith, and E. Keller, "Timing-based reconnaissance and defense in software-defined networks ," in *ACM ACSAC*, 2016.

[17] S. Liu, M. K. Reitner, and V. Sekar, "Flow reconnaissance via timing attacks on sdn switches," in *IEEE ICDCS*, 2017.

[18] 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.

[19] "Open vSwitch 2.11.90 Documentation," https://docs.openvswitch.org/en/latest/.

[20] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Cacheflow: Dependency-aware rule-caching for software-defined networks," in *SOSR*, 2016.

[21] X. Li and W. Xie, "CRAFT: A cache reduction architecture for flow tables in software-defined networks," in *IEEE ISCC*, July 2017.

[22] 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.

[23] H. Li, S. Guo, C. Wu, and J. Li, "FDRC: Flow-driven rule caching optimization in software define networking," in *ICC*, 2015.

[24] 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.

[25] Y. Gao, L. Deng, A. Kuzmanovic, and Y. Chen, "Internet cache pollution attacks and countermeasures," in *IEEE ICNP*, 2006.

[26] M. Dehghan, D. Goeckel, T. He, and D. Towsley, "Inferring military activity in hybrid networks through cache behavior," in *MILCOM*, 2013.

[27] "OpenFlow switch specification," Open Networking Foundation, Version 1.5.1, March 2015.

[28] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in *HotSDN*, 2013.

[29] A. Abel and J. Reineke, "Measurement-based modeling of the cache replacement policy," in *RTAS*, 2013.

[30] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "Leveraging Zipf's law for traffic offloading," in *SIGCOMM*, 2012.

[31] 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.

[32] 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.

[33] M. Dehghan, B. Jiang, A. Dabirmoghaddam, and D. Towsley, "On the analysis of caches with pending interest tables," in *ICN*, September 2015.

[34] 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.

[35] T. Benson, "Data set for IMC 2010 data center measurement," http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html.

[36] D. Y. Huang, K. Yocum, and A. C. Snoeren, "High-fideltiy switch models for software-defined network emulation," in *HotSDN*, August 2013.