Host-based Flow Table Size Inference in Multi-hop SDN

Tian Xie*, Sanchal Thakkar*, Ting He*, Novella Bartolini†, Patrick McDaniel‡
*Pennsylvania State University, University Park, PA, USA. Email: {tbx5027,sjt5721,tinghe}@psu.edu
†Sapienza University of Rome, Rome, Italy. Email: novella@di.uniroma1.it
‡ University of Wisconsin, Madison, WI, USA. Email: mcdaniel@cs.wisc.edu

***Abstract*—As a novel network paradigm, Software Defined Networking (SDN) has greatly simplified network management, but also introduced new vulnerabilities. One vulnerability of particular interest is the *flow table*, a data structure in every SDN-enabled switch that caches flow rules from the controller to bridge the speed gap between the data plane and the control plane. Prior works have shown that an adversary-controlled host can accurately infer parameters of the flow table at its directly-connected edge switch, which can then be used to launch intelligent attacks. However, those solutions do not work for flow tables at internal switches. In this work, we develop an algorithm that can infer the different flow table sizes at internal switches by measuring the Round Trip Times (RTTs) of a path traversing these switches from one of its endpoints. A major challenge in this problem is the lack of an inferable relationship between the RTTs and the flow table hits/misses at the traversed switches. Our solution addresses this challenge by experimentally identifying the inferable information and designing an inference algorithm that combines carefully designed probing sequences and statistical tools to mitigate measurement noise and interference. The efficacy of our solution is validated through experiments in Mininet.**

***Index Terms*—software defined network, adversarial reconnaissance, flow table size inference.**

I. Introduction

Software Defined Networking (SDN) has revolutionized the way networks are constructed and managed. A key feature of SDN is the separation of the control plane from the data plane, which has resulted in greater communication flexibility and easier network management. However, the increasing prevalence of SDN has raised security concerns, because although SDN facilitates defense against traditional IP-network attacks, it also introduces new vulnerabilities.

A particularly important vulnerability is the *flow table*, a data structure in every SDN-enabled switch that caches flow rules received from the controller. Flow tables are typically small, holding only a few thousand rules, due to the high cost and power consumption of the underlying storage medium. A packet whose header does not match any of the existing rules has to be handled by the controller, which slows down the forwarding and imposes load on the control plane. This vulnerability has been exploited to launch various flow table overflow attacks [1], [2], [3], [4] and control plane saturation attacks [5]. Apart from those, [6] introduces algorithms that can explicitly deduce the size, the replacement policy, and the state of a flow table from probes sent by a compromised host. However, all these attacks target on edge switches directly connected to adversary-controlled hosts.

In this work, we deepen the vulnerability analysis to internal switches, by studying the possibility to infer the flow table sizes of multiple switches traversed by a multi-hop path originating from an adversary-controlled host. Given the ability to craft packets that can trigger new rule installation as shown in [1], we develop an inference algorithm that uses the Round Trip Times (RTTs) of specially-designed sequences of probes to deduce information about the involved flow table sizes. This capability will enable a host-based adversary to map out the resource limitations along each candidate attack path and plan more intelligent attacks similar to [6]. The focus of this work is on inferring the flow table sizes; how to exploit or protect this information is left to future work.

A. Related Work

General cache inference: As the flow table acts as a cache of flow rules, flow table inference can be viewed as a special case of general cache inference. There are, however, few positive results for general caches. In [7], a procedure was proposed to infer the cache replacement policy and input parameters from observations of all the misses, which is not applicable in our context as the adversary-controlled host can only observe the results of its own packets. In [8], an algorithm was proposed to infer the size of a Least Recently Used (LRU) cache, but it did not consider the problem that the replacement policy can be different and unknown.

Flow table inference: More results are available on flow table inference, where the notably different communication performance under flow table hit/miss has been used to infer various characteristics of the target flow table, mostly in the context of adversarial reconnaissance. For example, [9], [10], [11] showed the feasibility of inferring the presence of certain flow rules, [1] showed a probing scheme to infer the timeout configurations of flow rules and the bitmasks in the match fields, and [12] proposed two different algorithms to infer the flow table size under First In First Out (FIFO) and LRU rule replacement policy, but the policy must be known to apply the correct algorithm. Most of these works use RTTs as their observations. In our previous work [6], RTT measurements were used to infer both fixed parameters of the flow table such as size and replacement policy, and transient parameters such as # active flows and their rates. However, all these works only

This work was supported by the National Science Foundation under awards CNS-1946022 and CNS-2106294.

targeted the flow table at the edge switch directly connected to the probing host. In this work, we extend the scope to the flow tables at switches multiple hops away from the probing host. The main challenge in this case is the increased randomness in RTT measurements, which makes it impossible to accurately infer the hit/miss experienced by a probe at each of the traversed switches (see Section III-A).

Attack modeling: Due to their small sizes, flow tables are known to be vulnerable to denial-of-service attacks that either overwhelm their capacity [1], [3], [4], [5] or occupy them with unpopular content [2], [6], both degrading the quality of service for legitimate users. Most of these works mounted the attacks on edge switches directly exposed to malicious hosts. The reconnaissance method developed in this work can potentially enable a host-based adversary to plan intelligent attacks at switches deeper in the network. We leave the detailed study of such attacks and their defenses to future work.

B. Summary of Contributions

Our contributions are summarized as follows:

1) We formulate the *Multi-Switch Size Inference (MSSI) Problem* to jointly infer the flow table sizes at multiple switches from RTTs measured by a single probing host.
2) By experimentally analyzing the relationship between RTTs and flow table hits/misses, we develop an algorithm that can accurately estimate the set of flow table sizes along the probing path by combining specifically designed probes with customized statistical tools.
3) We demonstrate via trace-driven Mininet experiments that the proposed algorithm can achieve 10 times higher accuracy than existing solutions.

Roadmap. Section II introduces our problem formulation. Section III presents our solution. Section IV validates the proposed solution via experiments. Section V concludes the paper. *Further experiments and results are provided as supplementary materials in [13].*

II. Problem Formulation

A. Network Model

Consider a multi-hop SDN as illustrated in Fig. 1. We model the flow table of each switch as a *cache* of flow rules. According to the OpenFlow Switch Specification [14], each rule contains several fields, including match, priority, counter, action, and timeout, which determine which packets the rule applies to and how to process them. Although in practice a switch may contain multiple flow tables organized into a pipeline, the default setting is to store all the rules in a single table, which fits our cache-based model.

Each flow table, modeled as a cache of flow rules, has two fundamental parameters: (1) the *size*, which determines the maximum number of flow rules that can be stored, and (2) the *replacement policy*, which governs the evictions of rules when the table reaches its capacity. The former parameter is akin to the cache size, while the latter parameter is similar to the cache replacement policy. While OpenFlow provides the option to remove rules proactively based on timeouts, their impact

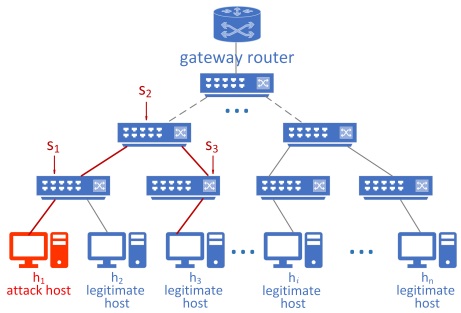

Fig. 1. Sample topology showing with the attack host and the target switches.

is negligible compared to reactive rule replacements when the flow table becomes saturated. The replacement policy is typically an approximation of FIFO or LRU, e.g., Open vSwitch [14] implements an approximation of LRU when rules only have idle timeouts or FIFO when rules only have hard timeouts. There are also more sophisticated replacement policies proposed by researchers, such as those approximating the Least Frequently Used (LFU) policy [15], [16], [17]. As our focus is on the inference of flow table size, we do not make any assumption on the replacement policy, except that: (i) *each newly requested rule will be inserted into the table*, and (ii) when the table is full, *the next rule to evict will be the oldest rule if all the rules are requested only once*. These assumptions are naturally satisfied by FIFO and LRU, and are consistent with LFU.

B. Threat Model

We assume that the adversary controls one compromised host that can measure the RTTs along routing paths originating from it by generating round trip probes such as ping or TCP packets. We assume the adversary to know the routing paths to each destination. For example, in Fig. 1, the attacker knows that the path from a compromised host h_1 to another host h_3 traverses switches s_1, s_2, and s_3. Moreover, we assume that through the technique in [1], the adversary can identify the header fields used in rule matching and hence knows how to craft packets to trigger new rule installations.

Our focus is on the inference of flow table sizes along one probing path, which can then be repeated for all the paths the compromised host can probe. Let K denote the number of switches on the path under consideration, and $C_1, C_2, ..., C_K$ denote the flow table sizes of these switches, measured by the number of flow rules they can each store. The solution to this basic problem can be used as a building block in addressing more general cases with multiple compromised hosts and multiple probing paths, which is left to future work.

C. Multi-Switch Size Inference (MSSI) Problem

The goal of the attacker is to use the measured RTTs to infer as much information as possible about the flow table sizes $(C_i)_{i=1}^K$ of multiple switches traversed by the probing path, referred to as the MSSI problem.

The above problem is an extension of the single flow table size inference problem addressed in [6]. Assuming the ability to accurately detect whether a probe incurs hit or miss at the target flow table, [6] developed an algorithm called

Robust Cache Size Estimation (RCSE) that can infer the size of this flow table. A straightforward solution to MSSI is thus to apply RCSE to each of the target switches based on inferred hits/misses at this switch. However, as shown later (Section III-A), the RTT of a multi-switch path does not permit accurate inference of whether a hit or miss has occurred at each of the switches. Thus, a different solution is needed for MSSI.

III. Algorithm Design

We first conduct a preliminary experiment to identify what information can be extracted from RTT measurements and then design our inference algorithm accordingly.

A. Preliminary Experiment

Compared to flow table inference based on probing a single switch, the main challenge in performing inference by probing a sequence of switches is the lack of an inferable relationship between the RTTs and the hits/misses incurred at the traversed switches. We start by quantifying this challenge through a preliminary experiment based on Mininet [18], which is an SDN emulator running real kernel, switch, and application code. The code for our experiment is available at [19].

Methodology: We emulate the round-trip communications along a single path with K switches by sending TCP packets. Below we present the results obtained under $K = 2$, but similar results have been obtained under other values of K (see [13]). Let the traversed switches be s_1 and s_2. Since each probe invokes two flow rules at each switch (one for the probing packet and one for its acknowledgement), each probe invokes 4 flow rules, leading to $2^4 = 16$ possible scenarios of hits/misses as each invoked rule may or may not be in the flow table. We profile the RTT distribution under each scenario by configuring the desired flow table content from the controller and then sending a probe to measure its RTT. For example, to create the scenario of 'm+h+h+h' (miss at the first traversal of s_1 and hits elsewhere), we send a TCP packet, remove the rule for forwarding this packet at s_1, and send the same packet again to measure its RTT. We repeat this procedure to collect 1000 RTT measurements under each scenario.

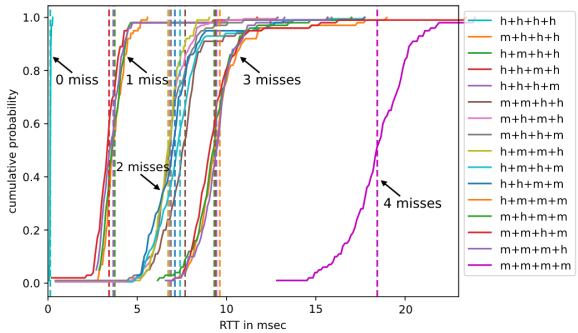

Fig. 2. RTT distributions under all hit/miss scenarios (the vertical lines indicate the average RTTs in each scenario).

Result: Fig. 2 shows the profiling result in terms of the empirical Cumulative Distribution Function (CDF) of the RTT under each hit/miss scenario. These results reveal that: (i) there is a clear positive correlation between the number of misses and the RTT, with notable gaps between the mean RTTs under different numbers of misses, but (ii) there are no detectable differences between the RTT distributions under different locations of the same number of misses, and (iii) the ranges of RTTs under adjacent numbers of misses have nonempty overlap.

Observation: The above results show that by measuring the RTTs, it is possible to estimate the number of incurred misses but not their specific locations, and the estimation based on a single RTT measurement will contain non-negligible error. This observation motivates our proposed solution as follows.

B. Proposed Algorithm

The preliminary experiment suggests that we cannot uniquely associate the inferred flow table sizes with the switches, as a change in the order of traversal will not notably change the RTT distribution. Therefore, we can at most hope to infer the set of flow table sizes $\{C^{(1)}, \ldots, C^{(K)}\} = \{C_1, \ldots, C_K\}$, where $C^{(1)} \le C^{(2)} \le \cdots \le C^{(K)}$.

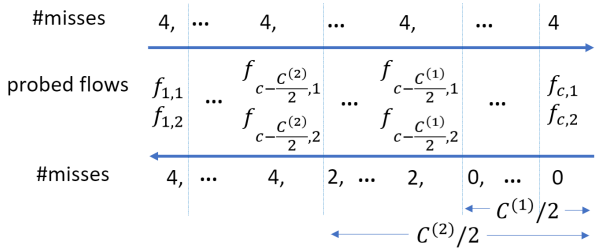

Fig. 3. Forward-backward probing for a two-switch path ($C^{(1)}$: size of the smaller flow table, $C^{(2)}$: size of the larger flow table).

1) Vanilla Solution: Assuming accurate estimation of #misses incurred by each probe, we can directly extend a previous algorithm `forward-backward-probing` in [6] to infer the flow table sizes at multiple switches.

The key idea of `forward-backward-probing` [6] is to use a long sequence of probes, each requiring a different rule, to flush the target flow table, and then use the same sequence of probes, sent in the reverse order, to measure the number of rules stored in the flow table, which reveals its size. We can extend this idea to reveal the sizes of multiple flow tables as follows. Consider the case of probing a two-switch path with even-sized flow tables, as illustrated in Fig. 3. Let $f_{i,1}$ denote the en route flow for the i-th probe and $f_{i,2}$ the corresponding return flow. In the "forward probing" process, we generate a sequence of c distinct back-to-back probes for a sufficiently large parameter c ($c > C^{(2)}/2$) to flush the flow tables. In absence of background traffic, the smaller flow table will store the rules generated by the last $C^{(1)}/2$ probes, and the larger flow table will store the rules generated by the last $C^{(2)}/2$ probes. In the "backward probing" process, we repeat the probes in the reverse order, which will generate 0 miss for each of the first $C^{(1)}/2$ probes, 2 misses for each of the next $C^{(2)}/2 - C^{(1)}/2$ probes, and 4 misses for each of the remaining probes. In the case that $C^{(1)}$ (or $C^{(2)}$) is an odd number, there will be one probe that incurs 1 miss (or 3 misses) during backward probing. Thus, given the #misses incurred by each probe, we can use the number of probes incurring each #misses during backward probing to estimate $C^{(1)}$ and $C^{(2)}$. The idea can be easily extended to the case of $K \ge 2$ switches.

However, the above solution hinges on accurate estimation of *the #misses incurred by each probe*, which is nontrivial to obtain due to the overlapping ranges of RTTs under different #misses as shown in Fig. 2. Directly applying this solution will lead to poor accuracy due to the notable error in estimating #misses from individual RTT measurements, as shown in Table I.

2) Improved Solution: To improve the robustness to the randomness in RTTs, we propose to combine forward-backward probing with change point detection. The observation is that while individual RTTs can fluctuate randomly, the sequence of RTTs during backward probing can be decomposed into multiple sub-sequences as illustrated in Fig. 3, each corresponding to a specific #misses. Thus, if we can detect the boundaries between these sub-sequences, we can aggregate the RTT measurements in each sub-sequence to estimate the #misses with higher accuracy.

Overall algorithm: The overall algorithm *Multi-Switch Size Inference (MSSI)* is shown in Algorithm 1. Starting from an initial value of the sequence length c (line 2), it exponentially increases c (line 13) to look for a sequence length large enough to flush the largest flow table, indicated by the largest estimated size $\hat{C}^{(K)}$ being less than $2c$ (line 10). The algorithm relies on a subroutine called `Robust-CuSum-Probing` (Algorithm 2) to perform the probing experiment and compute a noisy estimate $\tilde{\boldsymbol{C}}$ of the flow table sizes. The subroutine is repeated N times, and the results are aggregated by taking the maximum for each flow table size (line 7). This is because the presence of background traffic can cause the rules for probing packets to be evicted earlier, leading to underestimation of the flow table sizes, and taking the maximum over multiple trials can reduce the underestimation error. Here, N is a design parameter controlling the tradeoff between the robustness against background traffic and the probing cost. Note that since only the largest numbers are recorded (line 7), once $\hat{C}^{(1)} = 2c$ (line 8), all the estimates have reached their maximum value under c distinct probes, and hence there is no need to further repeat the experiment with the current value of c.

Subroutine: The subroutine, presented in Algorithm 2, implements a variation of the forwarding-backward probing process, with three primary steps:

1) Perform change point detection on the sequence of RTTs measured during backward probing using CuSum, and record the change point in an array if changes are detected for τ consecutive measurements (lines 2–29).
2) Based on the detected change points in descending order, divide the RTT measurements into sub-sequences and estimate the #misses for each sub-sequence based on the average RTT (lines 30–33).
3) Infer the flow table sizes in ascending order from the #probes experiencing each #misses (lines 34–36) based on the idea described in Section III-B1.

We now provide detailed explanations for the above steps.

Step 1 - Change Detection: CuSum (cumulative sum) [20] is a sequential change point detection technique for detecting changes in mean. In our context, we apply CuSum to the sequence of RTTs measured during backward probing, stored as an array with $RTT[i]$ being the RTT for probe p_i (line 6). CuSum maintains a cumulative sum $S \leftarrow \max(0, S + \tilde{x} - \omega)$ (line 15), where $\tilde{x}$ is the latest RTT measurement $RTT[i]$ normalized by the estimated mean $\bar{x}$ and variance σ^2 (line 10). We estimate the mean and variance from the measurements since the last detected change at $RTT[b]$ by

$$\bar{x} = mean(RTT[i:b]) := \frac{1}{b-i+1}\sum_{t=i}^{b} RTT[t], \qquad (1)$$

$$\sigma^2 = var(RTT[i:b]) := \frac{1}{b-i}\sum_{t=i}^{b} (RTT[t]-\bar{x})^2. \qquad (2)$$

Originally, CuSum detects a change if S meets a detection threshold T, where ω and T are design parameters controlling the tradeoff between false alarm and miss detection. To overcome the randomness in a single RTT measurement, we modify the decision rule by introducing a robustness parameter τ such that a change is detected only if $S \geq T$ for τ consecutive measurements, based on the same initial cumulative sum $\tilde{S}$ (line 13). Once a change is detected, we will record the change point in an array B and restart the computation of CuSum from the latest change point (lines 23–26). We can repeat this step (lines 2–29) multiple times and aggregate the detected change points (e.g., via majority vote) to improve the robustness against measurement noise.

Step 2 - #Misses Estimation: As the RTT measurements between consecutive change points are expected to be associated with the same #misses, we can use the change points to partition the RTT measurements into sub-sequences, and use the measurements within a sub-sequence as i.i.d. samples to obtain a more accurate estimation of the #misses. In our evaluation, we use an intuitive decision boundary given by

$$t_{m-1,m} := \frac{\underline{\mu}_m + \overline{\mu}_{m-1}}{2}, \quad m = 1, \ldots, 2K, \qquad (3)$$

Algorithm 1: Multi-Switch Size Inference (MSSI)

input : Number of switches on the probing path K, number of repetitions per probing experiment N, change detection parameters (ω, T, τ)
output: Estimated flow table sizes in increasing order $\hat{\boldsymbol{C}} := (\hat{C}^{(i)})_{i=1}^{K}$

1 $\hat{\boldsymbol{C}} \leftarrow zeros(1, K)$;
2 $c \leftarrow 1$;
3 **while** *true* **do**
4 **foreach** $i = 1, \ldots, N$ **do**
5 $\tilde{\boldsymbol{C}} \leftarrow$ `Robust-CuSum-Probing`(K, c, ω, T, τ);
6 **foreach** $j = 1, \ldots, K$ **do**
7 $\hat{C}^{(j)} \leftarrow \max(\hat{C}^{(j)}, \tilde{C}^{(j)})$;
8 **if** $\frac{\hat{C}^{(1)}}{2} = c$ **then**
9 break;
10 **if** $\frac{\hat{C}^{(K)}}{2} < c$ **then**
11 break;
12 **else**
13 $c \leftarrow 2 \times c$;
14 return $\hat{\boldsymbol{C}}$;

Algorithm 2: Robust CuSum Probing

input : number of switches K, number of probes c, change detection parameters (ω, T, τ)
output: Estimated flow table sizes in increasing order $\hat{\boldsymbol{C}}$

1 $\hat{\boldsymbol{C}} \leftarrow zeros(1, K)$;
2 $S \leftarrow 0$, $b \leftarrow c$, $B \leftarrow [c]$, $\kappa \leftarrow 0$;
3 **foreach** $i = 1, \ldots, c$ **do**
4 send probe p_i;
5 **foreach** $i = c,\ c-1, \ldots, 1$ **do**
6 send probe p_i, and record its RTT in $RTT[i]$;
7 **if** $i = b$ **then**
8 $\tilde{x} \leftarrow 0$;
9 **else**
10 $\tilde{x} \leftarrow \frac{(RTT[i]-\bar{x})}{\sigma}$, where $\bar{x} = mean(RTT[i:b])$, $\sigma^2 = var(RTT[i:b])$;
11 $S_{pre} \leftarrow S$;
12 **if** $\kappa > 0$ **then**
13 $S \leftarrow \max(0, \tilde{S} + \tilde{x} - \omega)$;
14 **else**
15 $S \leftarrow \max(0, S + \tilde{x} - \omega)$;
16 **if** $S >= T$ **then**
17 $\kappa \leftarrow \kappa + 1$;
18 **if** $\kappa = 1$ **then**
19 $S' \leftarrow 0$, $\tilde{S} \leftarrow S_{pre}$;
20 **else**
21 $\tilde{x}' \leftarrow \frac{(RTT[i]-\bar{x}')}{\sigma'}$, where $\bar{x}' = mean(RTT[i:i+\kappa-1])$, $\sigma'^2 = var(RTT[i:i+\kappa-1])$;
22 $S' \leftarrow \max(0, S' + \tilde{x}' - \omega)$;
23 **if** $\kappa = \tau$ **then**
24 $b \leftarrow i + \tau - 1$;
25 append b to B;
26 $S \leftarrow S'$, $\kappa \leftarrow 0$;
27 **else**
28 $\kappa \leftarrow 0$;
29 append 0 to B;
30 $\hat{\boldsymbol{c}} \leftarrow zeros(1, 2K)$;
31 **foreach** $j = 1, \ldots, |B| - 1$ **do**
32 $m \leftarrow$ estimated #misses for RTT $mean(RTT[B[j+1]+1 : B[j]])$;
33 $\hat{c}_m \leftarrow B[j] - B[j+1]$;
34 $\hat{C}^{(1)} \leftarrow 2\hat{c}_0$;
35 **foreach** $k = 1, \ldots, K-1$ **do**
36 $\hat{C}^{(k+1)} \leftarrow \hat{C}^{(k)} + 2\hat{c}_{2k}$;
37 return $\hat{\boldsymbol{C}}$;

where $\underline{\mu}_m$/$\overline{\mu}_m$ is the minimum/maximum average RTT under a hit/miss scenario with m misses, and estimate the #misses as m if the mean RTT falls in $[t_{m-1,m}, t_{m,m+1})$ ($t_{-1,0} := 0$, $t_{2K,2K+1} := \infty$). Other estimators can be used as well.

Step 3 - Size Inference: The results of change point detection and #misses estimation provide $\hat{c}_m$, the number of probes estimated to incur m misses ($m \in \{0, \ldots, 2K\}$). Based on the analysis in Section III-B1, we can then compute the flow table sizes using the recursive formula given in lines 34 and 36.

3) Performance Analysis: The accuracy of our solution is guaranteed as follows (see proof in [13]).

Theorem III.1. If (i) there is no background traffic during probing, (ii) all flow table sizes are even, and (iii) each estimate $\hat{c}_m$ is the true #probes incurring m misses ($m = 0, \ldots, 2K$), then the inference result $\hat{\boldsymbol{C}}$ of Algorithm 2 will be equal to the ground truth $\boldsymbol{C} := (C^{(k)})_{k=1}^{K}$.

Theorem III.2. Assume that (i) there is no background traffic during probing, (ii) all flow table sizes are even, (iii) each RTT measurement is an independent Gaussian random variable with distribution[1] $\mathcal{N}(\mu_m, \sigma_m^2)$, where m is the incurred #misses, and (iv) the sample mean $\bar{x}$ and variance σ^2 used to normalize $RTT[i]$ approximate the true mean and variance at the last change point before $RTT[i]$. Then the probability P_F for Algorithm 2 to falsely detect a change at $RTT[i]$ when there is no change in $RTT[i-\tau+1:i]$ is bounded by

$$P_F \leq (1 - \Phi(\omega))^{\tau}, \tag{4}$$

and the probability $P_M(m)$ for Algorithm 2 to miss a change from m misses to $m+2$ misses is bounded by

$$P_M(m) \leq \tau \Phi\left(\frac{\sigma_m(\omega + T) - \mu_{m+2} + \mu_m}{\sigma_{m+2}}\right), \tag{5}$$

where $\Phi(\cdot)$ is the CDF of $\mathcal{N}(0,1)$.

Remark: The assumption of even flow table sizes is not critical. It just implies an additional error of ± 1 if the flow table sizes are odd. Theorem III.1 implies that without background traffic, the accuracy of Algorithm 2 is mostly determined by the accuracy of change point detection, as the estimation of #misses for each sub-sequence is highly accurate (see Section IV). Theorem III.2 quantifies the accuracy of change point detection as a function of the design parameters (ω, T, τ). The interference from background traffic is mitigated by repeating Algorithm 2 multiple times as in Algorithm 1.

IV. Experimental Evaluation

Experiment Setting: We use Mininet as in Section III-A. Due to space limitation, we will only present results for $K = 2$ and $(C^{(1)}, C^{(2)}) = (2000, 3000)$, but similar results have been obtained under other settings (see [13]). We profile the mean RTT under each hit/miss scenario as in Section III-A using 1000 measurements. To simulate a realistic environment, we generate background traffic on each link according to the dataset UNI2 from [21], which contains 9 packet traces from a data center.

Results on #misses estimation: Table I shows the accuracy of estimating #misses using the thresholds in (3) based on different numbers of i.i.d. RTT measurements. Each entry corresponding to true #misses $= m$ and estimated #misses $= m'$ is the fraction of time the mean RTT generated under m misses falls into the decision interval corresponding to m' misses. The results show that (i) a single RTT is not a reliable indicator of #misses, but (ii) the mean of multiple RTTs (incurring the same #misses) is a reliable indicator.

Results on change point detection: The results in Table I indicate that we can accurately estimate the #misses for each sub-sequence of probes incurring the same #misses, if we can correctly detect the change points. The proper tuning of the

[1]Although the RTT distribution depends on both the number and the locations of the misses, for a given configuration of flow table sizes along the probing path, the locations are fixed once the number is determined.

TABLE I
DISTRIBUTION OF ESTIMATED #MISSES (500 RTTS PER SCENARIO)

	estimated #misses from single/500 RTTs				
true #misses	0	1	2	3	4
0	**0.99/1**	0.01/0	0/0	0/0	0/0
1	0.005/0	**0.9325/1**	0.0475/0	0.01/0	0.005/0
2	0.0025/0	0.0025/0	**0.765/1**	0.1975/0	0.0325/0
3	0/0	0/0	0.065/0	**0.8325/1**	0.1025/0
4	0/0	0/0	0.12/0	0.32/0	**0.68/1**

design parameters (ω, T, τ) are crucial in balancing false alarm and miss detection. To this end, we experimentally explore various values of ω and τ, and set T to be slightly smaller than the minimum value of S at the true change points.

Given a sufficiently large #probes c ($c > C^{(K)}/2$), the true change points during backward probing are $(c - C^{(1)}/2, \ldots, c - C^{(K)}/2)$. The mean absolute errors (MAE) of the detected change points[2] $(B[1], \ldots, B[K])$ are shown in Table II, measured as $(|c - C^{(1)}/2 - B[1]|, \ldots, |c - C^{(K)}/2 - B[K]|)$. Although the detection performance is insensitive to the value of τ in this experiment, our larger scale experiments in [13] indicate the necessity of having $\tau > 1$.

Based on these results, we set $\omega = 2, T = 5$, and $\tau = 3$.

TABLE II
MAE OF CHANGE POINT DETECTION ($c = 2048$, 5 MONTE CARLO RUNS)

	$\omega = 0, T = 18$	$\omega = 2, T = 5$	$\omega = 10, T = 2$
$\tau = 1$	$(975, 1425)$	$(2, 3.4)$	$(4, 1250)$
$\tau = 3$	$(861, 1407)$	$(2, 2.2)$	$(20.4, 1309)$

Results on flow table size inference: We compare our overall solution in Algorithm 1 with the vanilla solution in Section III-B1, which represents the state of the art [6]. We find that under the adopted background traffic, the proposed solution already returns accurate results even if $N = 1$ (i.e., running Algorithm 2 only once for each value of c), if we repeat the change point detection (lines 2–29 in Algorithm 2) multiple times (10 times here) and aggregate the results by majority vote. Table III shows the normalized mean absolute error $|C^{(k)} - \hat{C}^{(k)}|/C^{(k)}$ in percentage ($C^{(k)}$: ground truth, $\hat{C}^{(k)}$: estimate). The result shows that our proposed solution achieves 10 times higher accuracy than a direct application of the existing solution. We note that although $N = 1$ suffices here, a larger value of N may become necessary under heavier background traffic. We have validated this conclusion under other experimental settings, with results deferred to [13].

V. CONCLUSION

Observing that most adversarial reconnaissance studies in SDN only considered the edge switch, we presented a novel algorithm for inferring the flow table sizes of internal switches by measuring the RTTs of a multi-hop path from a compromised host. The proposed algorithm combined carefully designed probing sequences and statistical tools to infer the set of flow table sizes for the switches on the probing path, while being robust to measurement noise and interference from background traffic. To the best of our knowledge, this is the first algorithm that can reveal detailed parameters of internal switches of an SDN from the perspective of a host-based adversary. The efficacy of our algorithm has been demonstrated through Mininet experiments under realistic settings. Our result reveals a new vulnerability of SDN that signals the importance of protecting not just the edge switches but also switches deeper in the network from malicious hosts.

[2]According to Algorithm 2, $B[0] = c$. It is possible that the number of detected change points $\hat{K} \neq K$. In computing MAE, we pad zeros if $\hat{K} < K$ and ignore the extra change points if $\hat{K} > K$.

TABLE III
NORMALIZED MAE OF FLOW TABLE SIZE INFERENCE

normalized MAE	$C^{(1)}$	$C^{(2)}$
vanilla solution	14.5%	19%
improved solution	1.9%	1.5%

REFERENCES

[1] 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.

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

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

[4] 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.

[5] 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.

[6] M. Yu, T. Xie, T. He, P. McDaniel, and Q. K. Burke, "Flow table security in sdn: Adversarial reconnaissance and intelligent attacks," *IEEE/ACM Trans. Netw.*, vol. 29, no. 6, p. 2793–2806, dec 2021. [Online]. Available: https://doi.org/10.1109/TNET.2021.3099717

[7] 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.

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

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

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

[11] S. Achleitner, T. La Porta, T. Jaeger, and P. McDaniel, "Adversarial network forensics in software defined networking," in *ACM Symposium on SDN Research (SOSR)*, 2017.

[12] 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.

[13] T. Xie, S. Thakkar, T. He, and P. McDaniel, "Host-based flow table size inference in multi-hop SDN," April 2023. [Online]. Available: https://sites.psu.edu/nsrg/files/2023/04/Host-based-Flow-Table-Size-Inference-in-Multi-hop-SDN-Report.pdf

[14] "Open vSwitch 2.14.90 Documentation," https://docs.openvswitch.org/en/latest/.

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

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

[17] 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.

[18] "Mininet." [Online]. Available: http://mininet.org/

[19] T. Xie, "Cache Size Inference: RTT Collection ." [Online]. Available: https://github.com/SophieCXT/Flow-Table--Adversarial-Inference-RTT

[20] O. A. Grigg, V. T. Farewell, and D. J. Spiegelhalter, "Use of risk-adjusted CUSUM and RSPRT charts for monitoring in medical contexts," *Statistical Methods in Medical Research*, vol. 12, no. 2, pp. 147–170, 2003.

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