Mobile Networks and Applications
https://doi.org/10.1007/s11036-023-02156-0

Misreporting Attacks Against Load Balancers in Software-Defined Networking

Quinn Burke[1] · **Patrick McDaniel**[1] · **Thomas La Porta**[1] · **Mingli Yu**[1] · **Ting He**[1]

Accepted: 3 December 2021

Abstract
Load balancers enable efficient use of network resources by distributing traffic fairly across them. In software-defined networking (SDN), load balancing is most often realized by a controller application that solicits traffic load reports from network switches and enforces load balancing decisions through flow rules. This separation between the control and data planes in SDNs creates an opportunity for an adversary at a compromised switch to *misreport* traffic loads to influence load balancing. In this paper, we evaluate the ability of such an adversary to control the volume of traffic flowing through a compromised switch by misreporting traffic loads. We take a probabilistic approach to model the attack and develop algorithms for misreporting that allow an adversary to tune attack parameters toward specific adversarial goals. We validate the algorithms with a virtual network testbed, finding that through misreporting the adversary can control traffic flow to a high degree by drawing a target amount of load (e.g., + 200%) to within a 2% to 10% error of that target. This is yet another example of how depending on untrustworthy reporting in making control decisions can lead to fundamental security failures.

Keywords Network Security · SDN · Load Balancing

1 Introduction

Today's dynamic, cloud-centric marketplace demands faster and more reliable services. In order to meet these demands and maintain a specified quality of service, scaling out infrastructure has become a necessity. Key network functions, like load balancing, then provide the support necessary to keep these larger networks working efficiently. Load balancers split traffic fairly across equivalent backend servers or links to enable more efficient use of available network resources. In software-defined networking (SDN), however, load balancing typically manifests as a distributed system. The load balancer is divided into two components: the controller application that runs the load balancing algorithm and the network switches that enforce the load balancing decisions via flow rules. Here, chosen network switches report traffic loads (switch statistics) to the controller application which decides how to route incoming flows. Hence, efficient load balancing requires distributed trust among the switches in reporting accurate traffic loads. The distributed nature of the load balancer therefore creates an opportunity for an adversary at a compromised switch to misreport traffic loads to influence load balancing.

In this paper, we evaluate an adversary's ability to control the amount of traffic flowing through the com-promised switch (for eavesdropping and traffic analysis) by misreporting traffic loads (here, under-reporting). We take a probabilistic approach to model the attack and develop algorithms for misreporting that allow the adversary to tune attack parameters toward specific adversarial goals. We introduce two attacks against SDN load balancers to draw a target volume of traffic through the com-promised switch: the *trivial* attack that naively misreports zero load, and the *stealthy* attack that elusively misreports realistic load values. We then evaluate them against four widely used load balancing algorithms: *least-loaded*, *weighted least-loaded*,

✉ Quinn Burke
qkb@cs.wisc.edu

Patrick McDaniel
mcdaniel@cse.psu.edu

Thomas La Porta
tfl12@psu.edu

Mingli Yu
mxy309@psu.edu

Ting He
tzh58@psu.edu

[1] Pennsylvania State University State College, State College, PA, USA

Published online: 09 January 2024

least-connections, and *weighted least-connections*, which are included in the widely used Floodlight's [1] and OpenDaylight's [2] load balancing modules, and relied upon by several other specialized load balancing solutions [3–8]. We note that most dynamic load balancers in practice inevitably perform some form of *least-X* selection (e.g., least-loaded in bytes, least-connections) to select the most suitable path or endpoint for a flow [9, 10]. The wide reliance on this calculation provides motivation for evaluating its effectiveness in a setting where the load balancer is subject to malicious inputs—in the form of false load reports.

Additionally, as the network traffic characteristics depend on the services offered by a subnetwork, we consider in our analyses two distinct traffic models that are representative of workloads most commonly found in modern cloud and datacenter networks: short and long flows (in terms of flow duration) [11, 12]. The adversary must therefore calibrate the attack parameters appropriately based on the environment. We validate the attack algorithms with a virtual network testbed, finding that through misreporting the adversary can control traffic flow to a high degree by drawing a target amount of load (e.g., +200%) to within a 2% error of that target for short flows and within 10% error for long flows (with reduced error by using suggested alternative attack strategies). Thus our attack model is an effective tool for defensive analysis but also provides a means of planning attacks on real SDNs. This demonstrates that misreporting extends to other services beyond those discussed in prior work.

This is yet another example of how depending on collecting faithful information from untrustworthy sources leads to vulnerabilities, the results here being potentially disastrous, besides being difficult to detect in real-time. Our key contributions are:

- An attack model for analysis and planning of misreporting attacks against SDN-based load balancers.
- Development of two attacks against SDN load balancers that allow an adversary to control the volume of traffic through a compromised switch.
- Evaluation of misreporting attacks against four widely used load balancing algorithms and two distinct traffic patterns.

Prior work has partially addressed the issue of compromised switches with regards to eavesdropping, message integrity, and malicious link-discovery messages [13–15]; however, they have not considered the effects of malicious control messages in the context of load balancing. Here, we evaluate the performance of SDN-based dynamic load balancers in the presence of compromisedswitches who may misreport traffic loads (by under-reporting them). Several questions are raised concerning the performance of dynamic load balancers in adversarial settings: (1) *To what extent* can an adversary degrade the performance of load balancers by misreporting? (2) *When* must the adversary misreport? And (3), by *how much* must they misreport in order to accomplish their goal? We seek to address these key questions to highlight and quantify adversarial capabilities with regards to critical SDN services such as load balancers.

2 Background

Software-defined networks provide a framework that allows a more reliable and scalable alternative to traditional hardware and software-based load balancers which sit in front of network resources. In the following, we discuss how load balancing is typically realized in SDNs.

2.1 Load-balancing algorithms

Existing load balancing solutions for traditional networks come in two categories: static and dynamic. Static solutions implement proactive techniques for splitting incoming flows evenly across network resources (i.e., servers or links). Since the client mappings are known ahead of time, these techniques cannot exploit run-time knowledge of bandwidth utilization, often resulting in a negative impact on network performance (e.g., un-derutilization, increased latency). Common implementations of static load balancing include Randomized, Round-Robin, and hash-based solutions like equal-cost multipath (ECMP) [3, 16, 17]. In contrast, dynamic solutions implement various reactive techniques for connection assignment and provide a means for connection affinity by maintaining a per-connection state. They allow more flexible and favorable decision making by exploiting knowledge about resource utilization learned during normal operation of the network. Widely used implementations of dynamic load balancers include least-response-time, least-loaded, and least-connections, along with their weighted counterparts [1, 6, 18, 19].

2.2 Load-balancing architecture in SDN

Dedicated software-based load balancers offer scalability and reliability benefits over traditional hardware-based load balancers, which are often expensive and suffer from poor horizontal scalability [20]. Previous work has already demonstrated the ability of load balancers to be implemented as software running on commodity hardware [21–23]. In SDNs, however, load balancing typically manifests slightly differently. The load balancer is abstracted from the physical infrastructure by moving the load balancing logic to the control plane and distributing decisions to network switches in the form of flow rules.

To enable dynamic load balancing in SDNs, the network administrator defines a *pool* : a set of *switch ports*

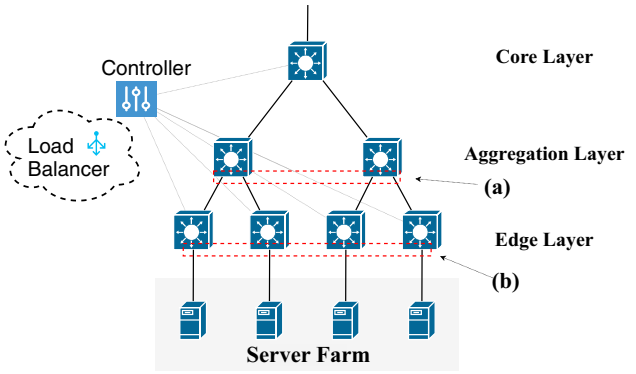

Fig. 1 The SDN-based load balancing architecture showing the pool members for balancing across (**a**) links or (**b**) servers

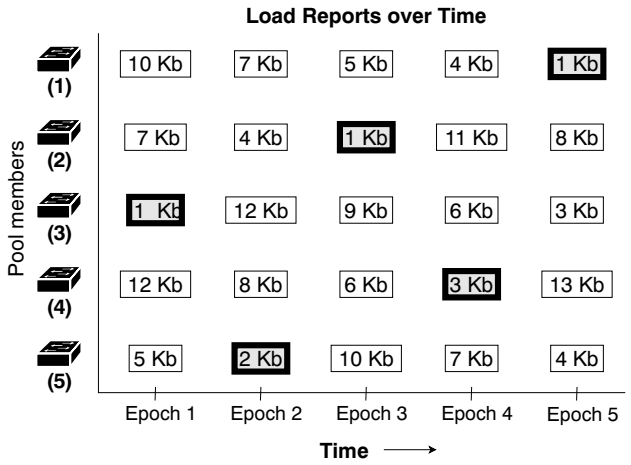

Fig. 2 Load reports (R_t^i) used for routing new incoming flows. Bolded reports are where switches reported the minimum load to the load balancer

connected to (aggregation or edge/access) links which are being balanced (see Fig. 1). The load balancer then requests traffic load reports from each pool member at each time epoch t. We refer to the epoch length, the time between load-report collections, as the *collection interval*, which may be one or more seconds long.

Under the OpenFlow [24] protocol, the reports come in the form of *switch statistics*. The loads represent the total activity at the switch ports since the last report, and may be measured in terms of Kb, number of active flows (or connections), etc., depending on the algorithm in use. The loads are then used by the controller to route new incoming flows (that are destined for the resources offered by the pool) according to the load balancing algorithm; for example, with a variant of *least-X* selection. Note that end-hosts in general do not run OpenFlow agents and therefore edge-switch load is instead commonly used as a proxy for server load [5].

As shown in Fig. 2, when a switch reports the minimum load at any epoch, the load balancer will temporarily route new flows through it. For example, switch (3) reports 1 Kb of activity in the first epoch, has new flows routed through it to a backend server or link, and reports 12 Kb of activity in the following epoch. Importantly, in the general case of the considered algorithms, all incoming flows are routed through the same pool member until the next load report is collected[1]; as the load balancer is removed from the data plane, it can only respond to the information given in load reports.

2.3 Notation for load balancing

Consider a network composed of N pool members, where the load balancer requests a load report R_t^i at each time epoch t for each member $1 \leq i \leq N$. For the case of least-loaded and least-connections [6], the load balancer temporarily routes new flows through the member who reported the minimum load (in bytes or number of active flows/connections), until the next load report is collected. More formally, the new flows will be routed through some member m in epoch t if:

$$R_t^m = \min_{1 \leq i \leq N} R_t^i, \tag{1}$$

If multiple members report the minimum load, random selection is done to choose between those members.

For weighted least-loaded and weighted least-connections, an exponentially weighted moving average (EWMA [25]) of loads is used for balancing. Weights are applied to the historical load values (α, where $0 \leq \alpha \leq 1$) and the current load value $(1 - \alpha)$ which are then summed together to smooth out sudden bursts that may lead to inefficient balancing. Then, the new load $R_t^{i'}$ computed for each member at time t is:

$$R_t^{i'} = \alpha R_{t-1}^{i\ J} + (1 - \alpha) R_t^i, \tag{2}$$

and new flows will be temporarily routed through the member with the minimum load as in (1), with R_t^m and R_t^i replaced by R_t^{mJ} and $R_t^{i'}$. Again, random selection is applied in the case of multiple members with the same minimum value.

3 Attacking the load balancer

Misreporting switch statistics allows adversaries to directly control the volume of traffic flowing through a compromised switch for larger-scale eavesdropping and traffic analysis, which have been established as significant threats in modern

[1] We leave to future work analyzing more specialized variants of these algorithms.

cloud networks (e.g., to uncover browsing histories [26]). Here, we introduce two attack methods against SDN-based load balancers.

3.1 Threat model

We assume switches report aggregate (i.e., port-level) statistics for themselves to a trusted load balancer, as balancing is typically done at a coarser level than individual flows [27]. Of these switches, we assume that one becomes compromised (to evaluate a lower bound on attacker capabilities, as several compromised switches would naturally allow an attacker to launch more sophisticated attacks). If the pool members are just the ports on the compromised switch, then load balancing integrity is clearly lost. We therefore consider the situation where ports on multiple edge switches form a pool[2] and report for statistics for themselves (as edge switch load is commonly used as a proxy for server load [5]). We note that attacks against edge-switch-based load balancing are independent of the actual internal network structure (as the edge-load reports do not provide insight on how load is to be distributed among internal links) and thus are not limited to just tree-based datacenter networks.

Switches may be compromised by insider or external adversaries [28–31]; while methods for compromising switches are outside the scope of this work, we note that prior work has demonstrated how an adversary may take control over a network switch—from exploiting weakly protected admin web interfaces to bugs in the switch operating system software and hardware backdoors [32]. For example, many organizations leverage open-source switch software such as Open vSwitch [33] and an adversary could fuzz the source code to identify exploitable bugs such as a buffer overflows, from which they can achieve privilege escalation and thereafter assume administrator control over the switch [34].

In the context of load balancing, we define the general adversarial goal as drawing a large fraction of network traffic to perform large-scale eavesdropping or denial-of service attacks. This would potentially enable the adversary to access sensitive control-plane messages (e.g., topology discovery messages) or sensitive client traffic in a multi-tenant datacenter network, besides causing availability problems—which have been demonstrated to be significant threats to SDNs [15, 32, 35]. The adversary accomplishes this by misreporting to induce the load balancer into sending a *target* volume of traffic (on average) through the compromised switch. The adversary's capabilities are limited to recording its own load reports and sending misreports. Note that misreporting is necessary to draw more traffic regardless of if packets on the switch ports are actually dropped; nevertheless, the adversary may drop an equivalent amount of traffic to evade detection systems that may leverage downstream switches to find inconsistencies in reports. We focus on adversaries under-reporting their true load to obtain an unfairly large proportion of traffic. We note that over-reporting loads may be useful for denying service to other switches in the pool, however, such an requires a different attack formulation and therefore we defer it to future work (see Section 5).

3.2 Overview

Studies of modern datacenter and cloud networks have observed that datacenter network traffic can be modeled by positive-skewed and heavy-tailed distributions of network flow sizes and flow durations [11, 12, 36]. For example, the flow distribution may consist of a majority of small (in bytes) and short (in seconds) flows that exist in the network only for a few seconds. This traffic is representative of applications such as web servers. In contrast, the flows may consist of a majority of relatively longer and larger flows that persist in the network for several seconds or minutes, for example, for applications like video streaming. Importantly, our preliminary observations of these traffic patterns across a pool of servers reveals a new threat vector for an adversary to compromise the load balancer: since pool members serve similar kinds of services, they observe similar traffic characteristics, which agrees with observations from prior work [11]. Therefore, the adversary can use their observed behavior as an approximation for what other switches in the pool observe (see Fig. 3), and tune their attack strategy to meet specific goals.

In this context, in steady-state if the adversary reports the minimum load (i.e., *"wins"* the epoch) for $X\%$ of the time epochs, they will observe $X\%$ of the total load in the system—as they draw $X\%$ of the total flows that arrive. In terms of load balancing fairness, we therefore formalize misreporting in terms of a tar- get fraction of load in the system to draw through the compromised switch (equivalently, a target fraction of epochs to *"win"*). We will refer to this target as T. Note that here the load in the system refers only to the client traffic destined for the pool. We then introduce two misreporting attacks with respect to T. In the *trivial* attack, the adversary's goal is to maximize the probability of winning the epoch each time a misreport is sent by reporting a load of zero. In the *stealthy* attack, the adversary misreports to nonzero loads that still have a high probability of being the minimum amount reported, allowing them to reduce their likelihood of being detected by an anomaly detection system while still obtaining the target load.

The challenge here is determining the required frequency of misreports (denoted by M) to draw the target load

[2] Note that switches may have multiple pool members (ports), but here we just consider a single pool member per switch and use *switch* and *pool member* interchangeably.

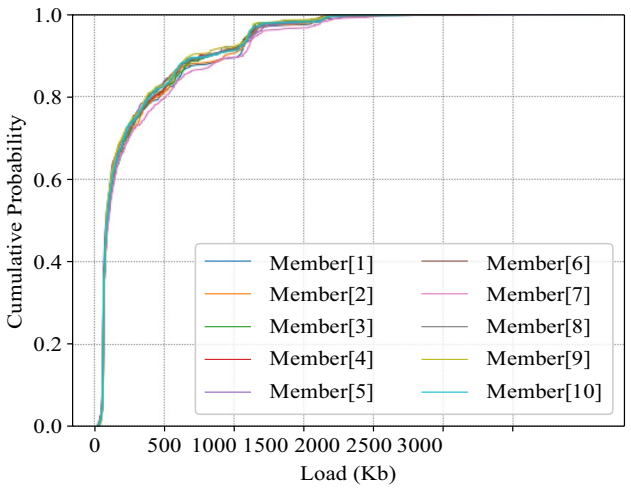

Fig. 3 Distribution of load reports collected once per second from switches in a 10-member load balancing pool over 10 min. Load distributions show small differences between pool members, enabling an adversary to estimate the traffic characteristics at other switches using their own observations

(equivalently, to win the target fraction of epochs). The adversary must therefore calibrate δ the misreported amount (in Kb or number of flows) sent to the load balancer. The choice of δ affects the misreporting success probability (i.e., if the load balancer immediately begins routing new flows through the switch) and therefore the required frequency of misreports to *accurately* draw the target load.

3.3 Attack model

Here we introduce an attack model from which the adversary will derive attack parameters M and δ for a given target T, generalizing the model introduced in our prior work [37].

3.3.1 Computing the required misreporting frequency

There are two components that determine the expected probability of winning an epoch: the probability of winning when not sending a misreport (reporting honestly) and the probability of winning when sending a misreport. At epochs when the adversary reports honestly, our heuristic approach is to assume the adversary has a fair chance, i.e., the probability of winning is $1/N$. When misreporting, guaranteeing a 100% misreporting success rate is difficult unless simply sending a load of zero in each misreport. However, sending a load of zero in each misreport (i.e., the trivial attack) may likely raise alarms, especially if the target load is very high (e.g., 75% of the load in the system) which will necessarily require misreporting more frequently. A more elusive approach is for the adversary to *simulate* activity at the switch by misreporting (setting δ) to very low loads *which have been observed previously* and which misrehave probability of success comparable to sending a load of zero. This is less likely to raise flags as it would be difficult to discern a legitimate report from a falsified one.

To this end, we approximate the load distribution observed at other pool members by that observed by the adversary before the attack. Without loss of generality, we first denote the compromised switch by switch N. If we let p denote a cumulative probability of the load distribution (see Fig. 3), then there is an associated load value p_L in Kb or number of flows) with that cumulative probability: a p fraction of observed loads falls within $[\mathrm{o}, p_L]$. If the adversary simulates activity at the port by misreporting to random loads within the bottom pth percentile of previously observed loads (e.g., the bottom 10th percentile loads), then the probability (approximately) of all other switches reporting a load value higher than the adversary can be estimated as:

$$\Pr\left[\bigcup_{i\neq N}\left(R_t^i > R_t^N\right)\right] \approx (1-p)^{N-1} \tag{3}$$

Note that if the adversary has knowledge of the size of the load balancing pool, they can directly evaluate the expression. However, if they do not, they can estimate it based on their own load reports. If we denote the steady-state load in the system by L, the adversary can compute from their own load reports the average flow duration of t epochs, arrival rate of R flows at winning epochs, and average flow rate of b bps. Then, they can compute the steady-state load in the system as: $L{=}t{\times}R{\times}b$. If under normal conditions, each pool member observes approximately a fair share $S{=}L/N$ load, then the adversary can estimate the pool size by dividing their fair share by the computed load in the system: $N{=}L/S$. Other methods for estimating pool size may leverage known techniques to map the load balancing pool in SDNs (or the network in general), such as probing the subnets identified in flow rules or listening for ARP packets [38, 39].

At a misreporting frequency of M, we use the two probabilities compute the expected fraction of load in the system to observe at any epoch during the attack (or equivalently, the expected winning probability):

$$\mathrm{E}[\mathrm{load}] = 1/N \times (1-M) + (1-p)^{N-1} \times M. \tag{4}$$

For the expected load to be the target load in the system T, we can rearrange Eq. (4) to solve for the *required* misreporting frequency M (where $0 < M < 1$) for the target:

$$M = \frac{T - 1/N}{(1-p)^{N-1} - 1/N}. \tag{5}$$

We further denote the attack window as W epochs long (e.g., from time epochs 500-1000), and can compute the actual number of misreports sent as the product: $M \times W$. We provide parameter descriptions in Table 1.

Table 1 Parameter descriptions for the attack model

Param	Description
N	Number of pool members
T	Target fraction of load in the system
p	Misreporting load upper threshold
M	Misreporting frequency
δ	Chosen misreported load
W	Attack length (in epochs)
R_t^i	Load report for member $1 \leq i \leq N$ at time t

3.3.2 Selecting a load value

The adversary will then randomly set δ to a previously observed load in $[0, p_L]$ for each misreport, to achieve the expected winning probability. Note that it is up to the attacker to assess the environment and decide what an appropriate undetectable load would be, i.e., how much load can they misreport before they are observable to some detection system. Thus, what we provide here is a method for configuring the attack such that the adversary can target a specific load (to within reasonable bounds) that they have decided as stealthy.

3.4 Launching the attack

In this section, we present two attack strategies for the adversary to draw the target load through the switch port: the *trivial* attack and the *stealthy* attack.

3.4.1 Trivial attack

In this attack, the goal of the adversary is to maximize the winning probability under a given target. To maximize the probability that the misreported load will be the minimum in Eq. (1), the adversary will select $p=0$, which corresponds in general to a load of zero (as switches generally always observe some background traffic), and thus the adversary will send a new load $R_t^{N'J'}$ in each misreport: $R_t^{N'J'} = \delta = 0$. They will then compute the required misreporting frequency from Eq. (5) and begin sending the misreports with zero load. Note that misreports can be sent at a fixed period $1/M$ or at random epochs during the attack with an average period of $1/M$. Moreover, the choice of a smaller p requires a smaller misreporting frequency, which is generally beneficial, however, consistent reports of zero load may become readily observable.

3.4.2 Stealthy attack

In this attack, the adversary relaxes the load values to which they misreport to (i.e., δ) in order to manage their detectability. After assessing the environment to determine an appropriate threshold p, the first step is to perform reconnaissance for a configurable period of time (e.g., 600 s, or 10 min), from which they record their load observations and prepare to send misreports. Note that they may also estimate the pool size during reconnaissance if necessary, as discussed in Section 3.3. The adversary will select an appropriate load for each misreport: $R_t^{N'J'} = \delta$, where $\delta \in [0, p_L]$. After the reconnaissance period is complete, the adversary computes the required misreporting frequency from Eq. (5) and begins sending the misreports with an appropriate load value. Here, misreports can also be sent at a fixed period $1/M$ or at random epochs during the attack with an average period of $1/M$. Further, the choice of a smaller p will result in a higher misreporting success probability and therefore require a smaller misreporting frequency. The inverse is also true, which leads to limited parameter flexibility for the stealthy attack. Thus, there is a natural tradeoff between misreporting success and detectability (discussed in Section 4).

3.5 Assessing the impact

To assess the effects of the proposed attacks, we first want to measure the direct impact of misreporting. We then evaluate the bounds on attacker capabilities under both the trivial and stealthy attacks to understand the extent to which an adversary can cause imbalance in the load balancing pool.

3.5.1 Attack effectiveness

To describe the direct impact of the attack with regards to drawing more traffic through the switch, we define a damage metric D. It represents the ratio of the average load on the compromised switch during the attack window to the average load observed under normal conditions. If we denote the average load during the attack by L_a, and under normal conditions by L_n, then the relative damage is:

$$D = \frac{L_a}{L_n} - 1. \tag{6}$$

To concretely quantify misreporting effectiveness, we introduce a potency metric P that represents the ratio of damage per percent of misreported epochs:

$$P = \frac{D}{M}. \tag{7}$$

We also measure the frequency, as well as the success rate of misreporting, which describes how often a misreport resulted in more traffic being routed through the compromised switch.

3.5.2 Capturing victim traffic

As the general adversarial goal is to draw more traffic on which to eavesdrop, we also discuss the success of

Table 2 Experimental network results with the Floodlight [1] SDN controller. We evaluate the effectiveness of the attack against four different load balancing algorithms and two traffic patterns. For a target of 30% across each attack scenario, we observe that the adversary can misreport to draw the target volume of traffic to within a small error

		Short flows				Long flows			
		LL	WLL	LC	WLC	LL	WLL	LC	WLC
Control	Average load	141.4 Kb/s	141.8 Kb/s	10.9 flows/s	11.2 flows/s	1542.3 Kb/s	1535.9 Kb/s	142.2 flows/s	138.9 flows/s
	Average load %	10%	10%	10%	10%	10%	10%	10%	10%
Trivial	Target load (T)	30%	30%	30%	30%	30%	30%	30%	30%
	Average load %	32%	31%	32%	32%	23%	23%	22%	22%
	Average load	450.9 Kb/s	445.4 Kb/s	34.6 flows/s	35.8 flows/s	3728.5 Kb/s	3556.6 Kb/s	316.3 flows/s	310.3 flows/s
	Misreport frequency	23%	24%	24%	26%	22%	23%	24%	24%
	Misreport success	100%	100%	100%	100%	100%	100%	100%	100%
	Damage	+219%	+214%	+217%	+220%	+142%	+132%	+122%	+123%
	Potency	+9.5%	+8.9%	+9.1%	+8.4%	+6.44%	+5.7%	+5.1%	+5.1%
	Model Error	+2%	+1%	+2%	+2%	-8%	-7%	-8%	-8%
Stealthy	Target load (T)	30%	30%	30%	30%	30%	30%	30%	30%
	Average load %	33%	29%	31%	32%	24%	21%	21%	20%
	Average load	467.9 Kb/s	416.5 Kb/s	33.4 flows/s	35.9 flows/s	3667.1 Kb/s	3270.9 Kb/s	302.1 flows/s	281.2 flows/s
	Misreport frequency	26%	22%	25%	24%	25%	23%	25%	24%
	Misreport success	99%	99%	93%	97%	98%	98%	97%	97%
	Damage	+231%	+194%	+206%	+221%	+138%	+113%	+112%	+102%
	Potency	+8.9%	+8.8%	+8.3%	+9.2%	+5.5%	+4.9%	+4.5%	+4.3%
	Model Error	+3%	-1%	+1%	+2%	-6%	-9%	-9%	-10%

our attack in the context of an adversary targeting a specific flow (or traffic type) that they have knowledge of existence (e.g., they know users are currently browsing a specific website). If an $X\%$ proportion of the load in the system (in number of flows or bits) is passing through the switch, then an equivalent description is that the adversary has an $X\%$ chance of capturing a victim flow. We therefore assess the ability to capture victim traffic by analyzing the flexibility in parameter choice under a given target load and the bounds on adversary capabilities under both attack strategies.

4 Evaluation

With the formulation of the attack model, here, we explore the effects on the performance of the load balancer in several scenarios (shown in Table 2) and address the last research question: to what extent can the adversary degrade the performance of the load balancer? We consider 4 widely used load-balancing algorithms: least-loaded, weighted least-loaded, least-connections, and weighted least-connections. We then provide analyses of the effectiveness of the two attacks in each scenario, the ability to capture victim traffic, and the detectability against state-of-the-art SDN detection systems.

4.1 Experimental setup

4.1.1 Network setup

For experimentation, we employ the latest version of the widely used Floodlight [1] SDN controller, along with its load balancing module. To configure the virtual network, we use the popular Mininet emulator [40] to create a similar topology of virtual switches and hosts to that shown in Fig. 1. New flows will originate from a source connected to the "top-most" switch in the figure, which represents a common gateway from which flows split paths in the network (e.g., an aggregation switch in a three-tiered network). Each switch runs the latest version of Open vSwitch (v2.12.0) and is invoked to connect to and receive forwarding instructions from the Floodlight controller. The directly connected hosts act as sinks for the incoming network flows. The attacks are then carried out by designating one switch as the adversary.

We configure the load balancer to have a single pool consisting of 10 SDN-enabled switches, which is a realistic pool size for small clusters based on real configurations used in the wild [41]. We note that our experimentation with larger pool sizes yielded qualitatively similar results, where the load is scaled proportionately. The switches are directly connected to a single backend resource (which can represent either servers, or more switches). We also configure the load

balancer to have a load-report collection period of 1 s, which is suitable for providing reasonably low load-error rates [27, 42, 43]. We set the attack window to $W = 300$ epochs, and we set the misreporting percentile to $p = 0.01$ for the stealthy attack and $p = 0.0$ for the trivial attack. Simulations are averaged over 10 independent executions. Without loss of generality, the adversary is designated by switch number N.

4.1.2 Traffic models

In evaluating our attacks, we draw from prior work to generate packet traces for each of the short and long traffic patterns. The sizes and durations of flows are randomly distributed amongst the probability distribution defined by two pareto curves, which are widely accepted approximations for network traffic behavior [36, 44]. The average flow durations from the traces are ≈ 1 s for short flows and ≈ 25 s for long flows. The average flow data rate had no detectable effect on the attack effectiveness, and we use 1500-byte packets for each flow. We then consider an average arrival rate of about 50 flows per second. Note that smaller or larger arrival rates yielded qualitatively similar results.

4.2 Attack effectiveness

In the first part of the evaluation, we measure the effectiveness of the attack for both traffic types. In Table 2, compared to the average load observed under normal conditions (the control experiment), running the trivial attack against the load balancer was able to effectively draw on average 31.7% (for short flows) and 22.5% (for long flows) of the load in the system toward the compromised switch. In fact, the trivial attack proved to be successful across all four of the considered load balancing algorithms, always drawing $> 200\%$ additional load for short flows and $> 120\%$ additional load for long flows, than under normal conditions. The misreporting frequency was on average 24% (with a required $M = 22\%$ for the specified target) of the attack window, and since the misreported load was zero (and loads must be non-negative), the misreporting success rate was also maximal (i.e., each misreport resulted in arriving flows for the next epoch being scheduled through the compromised switch). The resulting effect was a potency of 8.9% for short flows and 5.6% for long flows, to within a 3.5% error and 6.8% error for short and long flows, respectively.

The stealthy attack showed similar results with respect to misreporting success. The misreporting frequency was on average 24.2% (with a required $M = 25\%$ for the target), with an average success rate of 97.3% across all algorithms and traffic types. The adversary maintained on average 31.2% (for short) and 21.5% (for long) of the load in the system, to within a 2% error of the target for short flows and 10% error for longer flows. It follows that the potency of misreports was on average 8.9% for short flows and 4.8% for long flows, revealing that stealthy misreporting (i.e., simulating activity on the switch port) still does in fact have a non-negligible effect on load balancing fairness as simply reporting a load of zero does with the trivial attack, and almost double the effectiveness compared to when the average flow duration is much longer.

Interestingly, the misreporting success remained the same even in the case of weighted load balancing. Even with a significantly high weight factor α for weighted balancing (e.g., $\alpha = 0.5$) [45], where the misreported load only has half the significance toward the smoothed value, the adversary was able to misreport low enough for the load balancer to consider it the minimum and begin routing flows through it. Certainly, a much higher α would place more weight on the historical load value and thus dampen the effects of misreporting.

Takeaway Stealthily attacking the load balancer proved to be just as effective as trivially reporting a load of zero, without having to misreport at a much higher frequency than the trivial attack would require (although, that difference increases as p does). Neither algorithm proved to be more or less resistant to the stealthy nor trivial misreporting attack, and the results show that even for short attacks (here, 5 min), the adversary can control traffic flow to within a reasonable error.

4.3 Capturing victim traffic

In the second part of the evaluation, we explore the parameter flexibility under a specified target load and the bounds on attacker capabilities.

4.3.1 Parameter flexibility

As discussed in Section 3.3, the probability (approximately) of all other switches reporting a load value higher than the adversary can be estimated as $(1\ p)^{(N-1)}$. Independent of the traffic pattern, higher choices of the misreporting threshold p (to reduce detectability) will cause the adversary to misreport to within a large range of values. Then, the misreporting success rate (the expected load when misreporting to the pth percentile loads) decreases with a power-law relationship to the threshold p. Hence, misreporting within a larger range requires a significantly higher misreporting rate for the same target load.

For example, Eq. (5) from our attack model demonstrates in Fig. 4 that relaxing p to even the $p = 0.1$ percentile already requires a misreporting frequency of almost 60% for an expected load (target) of $\approx 25\%$, which is nearly $3\times$ that required when $p = 0.01$ (requires $\approx 19\%$). Similarly, in general for any target load, relaxing p to anything higher than

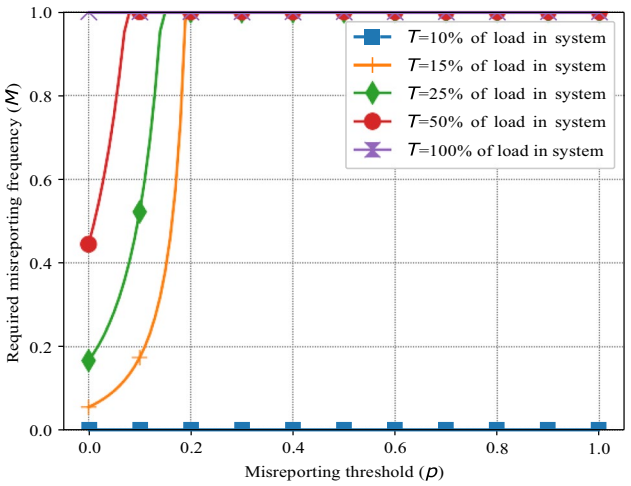

Fig. 4 Required misreporting frequency M as a function of the misreporting threshold p (from the attack model in Eq. (4)). Slight relaxations on p (to potentially reduce detectability) require significantly more misreporting

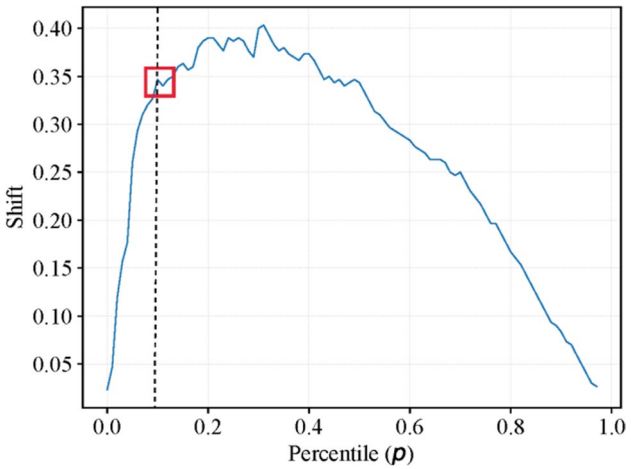

Fig. 5 The percentile shift from the perspective of an honest switch. As the adversary draws more traffic, the distribution at the honest switch changes, where they see more *low* load values. Thus, the pth percentile is in fact much higher than the adversary expects, resulting in a lower winning probability than expected (e.g., the 10th percentile shifts to the 35.th percentile)

the 20% percentile load requires effectively misreporting every time epoch. Thus, we observe that for an effective attack, the flexibility in parameter choice is limited. While still able to (approximately) draw the target load, too-frequent misreporting is unsustainable from an attacker's perspective. However, this observation is critical for future defense systems that protect the fairness in load balancing, as potential attackers must maintain somewhat-static behavior to have an effective attack (i.e., adaptivity is limited, especially for higher targets).

4.3.2 Bounds on attacker capabilities

This flexibility is further limited by an artifact of the network traffic behavior: Fig. 5 describes (at a misreporting frequency of 100%) how the perceived pth percentile that the adversary misreports *to* for long flows is shifted to a higher percentile (from the perspective of an honest switch) once the adversary begins to draw traffic away from the honest switches. As the adversary misreports more, the distribution at an honest switch changes more (as they see more relatively lower loads) and thus the pth percentile load is in fact larger than expected, leading to lower misreporting success rates and therefore a less load gained by the adversary. For example, the 10th percentile shifts by 35% when under attack (i.e., to the 45%th percentile), and thus the perceived misreporting success probability $(1-0.1)^{(N-1)}=0.9^{(N-1)}$ is in fact $(1-0.45)^{(N-1)}=0.55^{(N-1)}$ in steady-state.

In Fig. 6, we show how the load obtained during the attack begins to diverge from the attack/analytical model in certain scenarios. Note that we observed quantitatively similar results for the least-connections and weighted algorithms. First, the analytical model accurately describes the trivial attack to within a 2% error for short flows and 8% error for long flows (and approaches 0% error as the misreporting frequency increases). Moreover, as shown in the top figure for short flows and small p (i.e., $p=0.01$), the model is very accurate to within 2% error. For longer flows, the model is accurate to within 8% error (for $p=0.01$) and 2% error (for $p=0.1$) up to a misreporting rate of 25%. However, the nonlinear change in distribution (i.e., percentile shifts) from misreporting causes the adversary to overestimate how much load they will obtain at higher misreporting frequencies.

However, for short flows, when $p=0.1$, while there is no loss (i.e., the model only underestimates how much load the adversary truly obtains), the nonlinear change in distribution also plays a role in the higher model prediction error. We hypothesize that the average flow duration (and therefore the average amount of load in the system) has differing effects on the probability of successful misreporting (and therefore expected load). For example, eventual starvation of other pool members may naturally mitigate misreporting attacks (a feature which may not generalize across other SDN services). Thus, while the proposed attack model can capture the effects of misreporting for the trivial scenario, it does not accurately capture the phenomena when the adversary attempts to be very stealthy (at $p=0.1$) at higher misreporting frequencies. Accounting for these effects is the subject of future work. Nevertheless, the model itself is still useful for defensive analysis of as it describes the upper bounds of a potential attack by the most aggressive adversaries (i.e., when $p=0.0$).

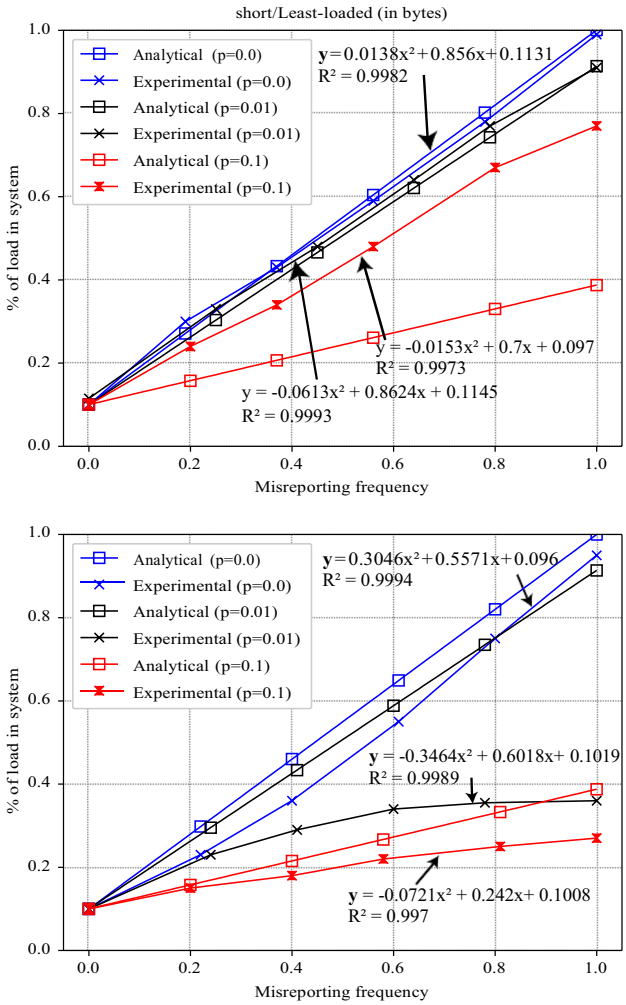

Fig. 6 Drawn load as a function of misreporting frequency (from the attack model in Eq. (4)) for the least-loaded algorithm against short flows (top) and long flows (bottom). The experimental results deviate from the model at higher misreporting frequencies, but are still accurate at lower frequencies (i.e., more stealthy, yet effective attacks)

However, we perform a quadratic regression analysis to understand the diminishing effect of misreporting as the frequency increases. We arrive at the formulas shown in Fig. 6, which accurately fit the experimental results, and demonstrate that for larger p and longer duration flows, attacker capabilities exhibit a slower growth and lower limit on how much load can be obtained. Combined with the already limited parameter flexibility, this further establishes that an adversary is confined to limited parameter choice in launching a successful attack (i.e., very small p); any relaxations on p will require misreporting effectively every time epoch. Thus, the adversary's behavior might reveal patterns that may be feasibly detected.

The key insight here is that a fixed misreporting frequency becomes less effective as p is relaxed or the average flow duration increases. However, trivially reporting a load of zero is generally always effective. A stealthy adversary may temporarily resort to the trivial strategy if they detect unsuccessful misreports. Moreover, this does not take away from the fact that an adversary can take an adaptive approach to misreporting by simply increasing the misreporting frequency until they achieve their desired load. Any attacker must understand however that there is a ceiling on how much load they can obtain (for non-zero p) as the misreporting frequency increases. Regardless, the adversary can achieve the same effect in realistic settings, as detecting misreporting attacks in SDN remains largely an open problem (see Section 4.4).

Takeaway The trivial attack can be accurately modeled, but misreporting tends to have diminishing returns at higher misreporting frequencies for other choices of p, and we resort to a regression analysis to understand the complexities in these scenarios. An adversary can however take a dynamic strategy by analyzing the effects of misreporting and re-calibrating p and M to directly manage their detectability or increase the accuracy in approximating the target load.

4.4 Detectability

In the third part of the evaluation, we examine state- of-the-art detection systems in SDN, and we apply two change detection techniques (the Kolmogorov–Smirnov two-sample test and the Median test [46, 47]) to identify how detectable the attack may be by solely examining load reports.

4.4.1 Sphinx

The Sphinx [48] detection system addresses attacks focused on corrupting network topology and data-plane forwarding state. It leverages logical network flow graphs to validate network state updates (e.g., validate that a new link-discovery message is legitimate). With respect to forwarding state, the core of the system relies on a proposed *similarity index*: a metric used to compare flow-statistics reports along a flow path. For example, if a switch along a flow path reports too low of a value in the flow statistics, Sphinx raises an alert for the switch (i.e., consider it to be dropping traffic).

The first problem (as explicitly stated in the paper) is that the system is unable to validate the accuracy of load reports from ingress or egress switches (i.e., edge switches), which is where our misreporting attack is staged. A second problem relates to load balancing: the system operates at flow-level, while load balancing is typically done at a coarser granularity (i.e., at port-level). This is a design limitation of Sphinx that makes it not applicable to the proposed attacks. Moreover, the similarity test cannot be applied equivalently to port-level statistics as flow paths combine and split at the switches and therefore no single path can be traced back to compare a set of port statistics under the current design.

4.4.2 FlowMon

The FlowMon [49] system addresses packet droppers and packet swappers (adversaries who send packets out of the wrong switch ports). With respect to misreporting (packet droppers), the authors introduce a threshold-based test to identify if a switch is transmitting a different number of packets than it is receiving:

$$\left| \frac{\sum_{\forall i \epsilon P_k} T_k^i - \sum_{\forall i \epsilon P_k} R_k^i}{\sum_{\forall i \epsilon P_k} T_k^i + \sum_{\forall i \epsilon P_k} R_k^i} \right| > \theta \tag{8}$$

where P_k is the set of all ports on switch k, T^k is the number of packets transmitted from port i of switch k in the last time epoch, and R^k is the number of packets received at switch k port i in the last time epoch. A difference indicates packet dropping, and the system raises an alert if the dropping percentage is greater than an allowed threshold θ. A critical aspect of FlowMon is that it relies on the statistics information in the load reports being accurate (otherwise the adversary can always modify the load reports to equate the number of transmitted and received packets). They introduce a second detector using a neighboring switch along a link to validate the accuracy of a load report:

$$\left| T_{ij} + T_{ji} + R_{ij} - R_{ji} \right| > \alpha \left| T_{ij} + T_{ji} \right| \tag{9}$$

where T_{ij} is the number of packets transmitted from switch S_i to S_j in the last time epoch (at a specified switch port), R_{ij} is the number of packets received by switch S_i from S_j in the last time epoch (at the associated switch port), and an alert is raised if the inequality holds. As with Sphinx, the underlying problem here is that the detector cannot be applied to edge links (i.e., those connecting the edge switch to the server, where the adversary is dropping traffic) as one side of the link (the host) does not provide statistics reports in the common SDN load balancing paradigm. Although in general, we posit that it may be possible to leverage a similar strategy to narrow the search for potential malicious switches to those connected by problematic links (as the discounted load in a misreport may be greater than α).

4.4.3 OpenWatch

The OpenWatch system [50] leverages volume-based detectors to identify when a switch's change in load is abnormal over an extended period of time or between consecutive load reports. They consider an anomalous event to be either a report that is greater than the $mean + 3 \times std$ of the load values, or consecutive reports that have an increase in load that is greater than $mean + 3 \times std$ of all increasing load changes. The fundamental limitation with the system is that (as with one of the FlowMon strategies) it relies solely on an individual switch's reports to determine whether the switch is exhibiting anomalous behavior or not. A malicious switch can break either of these detectors by simply ensuring that any load report it sends does not deviate from normal behavior (i.e., is within 3 std of the switch's mean load), and yet achieve the same effect. Since the misreports are to set previously observed values, the intended non-misreporting epochs could simply be set to the higher load values to not stray from normal behavior (from the perspective of the load balancer).

Without resorting to misreporting every epoch to evade suspicion, however, we ran experiments resembling a most extreme case (the trivial attack, where $p = 0.0$) for long flows, with an aggressive misreporting frequency of $\approx 60\%$ over a 5-min attack, and the adversary drawing $\approx 55\%$ of the load in the system. We observed over a 20-min snapshot of load values that the two OpenWatch detectors were only able to correctly identify 23% and 2% of the 60% misreported epochs, respectively. In fact, we observed significantly high false negative rates by the detectors, at 12.5% and 31.1%, respectively. Thus, due to the nature of datacenter network traffic, simple threshold-based detectors are cannot effectively identify misreporting attacks.

4.4.4 Change detection

In the last part of the evaluation, we apply two widely used change detection techniques, the Kolmogorov–Smirnov (KS) two-sample test and the Median test, to identify potential differences in the load distributions of the honest pool members from the compromised switch (which may indicate anomalous behavior). We assume that a detection system operates as a sliding window across load reports (for a 1-min and 5-min detection window). Note that here we consider window sizes close in length to the attack window, as the attacks could easily be hidden in the noise when using very large detection windows. For simplicity, we use the p-value as a proxy for identifying anomalous events, where a very small p-value indicates abnormal behaviors.

We show in Figs. 7 and 8 how the p-value of either test changes as a function of the starting position of the detection window—i.e., it begins to drop as the detection window becomes more aligned in time with the attack window. For an attack occurring between time 900-1200 s, we observe for a detection window of 60 s (and 300 s, respectively) that the p-value for the KS two-sample test drops rapidly when beginning at time 800 s (and 600 s, respectively). Here, we consider a p-value less than 0.05 to be within an acceptable range to reject the null hypothesis of the KS test, in confidence that the adversary's load reports are in fact abnormal. In other words, for either window size it would have taken

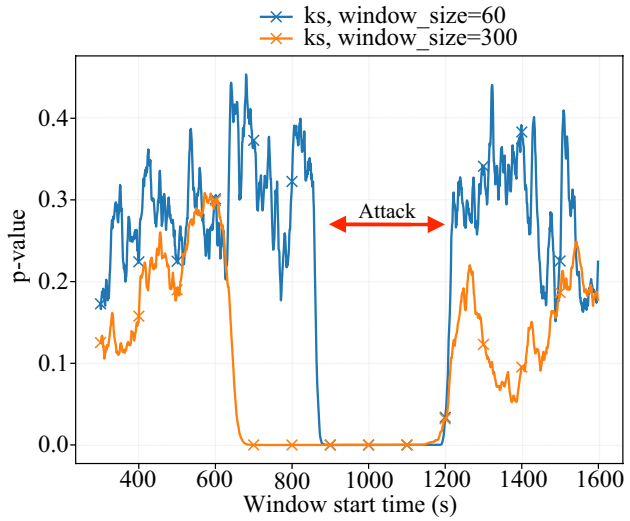

Fig. 7 We use the p-value as a proxy for identifying anomalous events. We show how the p-value of the Kolmogorov–Smirnov (KS) two-sample test changes as a function of the detection window starting time. At several points in time, it shows sudden drops that indicate a change in distribution (i.e., a potential anomalous event)

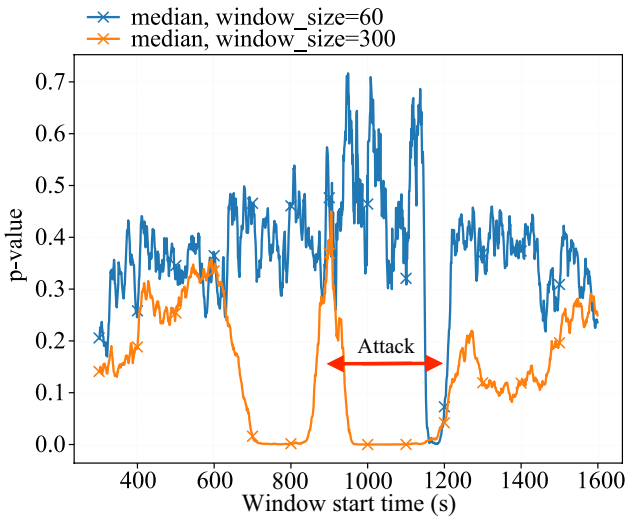

Fig. 8 We show how the p-value of the Median test changes as a function of the detection window starting time. At several points in time, it shows sudden drops that indicate a change in distribution (i.e., a potential anomalous event), with a large spike during the attack that indicates a missed detection for a period of time

only a few seconds for the detector to begin to identify a potential change in load distribution.

We observed similar results for the median test for a window size of 300 s. We, however, observed a spike when the window start time approaches the attack start time at 900 s, which implies a missed detection for a period of time. For a window size of 60 s we found that it would have taken the detector ≈ 5 min from the attack start to reach a similar conclusion (i.e., after the 5-min attack completed). Thus, we find here that in general statistical measures of difference have the potential to identify inconsistencies among pool members. However, window size plays an important role in detection accuracy: too short windows might take too long to identify a potential attack, and too long windows might miss transient attacks.

Takeaway Prior work on SDN defenses do not comprehensively address the problem of misreporting (e.g., cannot detect inconsistencies in aggregate load reports). Statistical methods commonly used for anomaly detection thus far provide the strongest grounds for further development of defenses. However, such a system cannot use the load reports as a source of truth in identifying anomalous behavior, and therefore the statistical measures should be supplemented by other techniques (e.g., probing-based techniques or systematic load report collection) to reduce the potential for false negatives or late detection.

5 Discussion

We have demonstrated that an adversary can control to a high degree the amount of imbalance caused in a load balancing pool by misreporting traffic loads. Although specialized load balancing algorithms may be more or less robust to misreporting than others, the fact stands that validating load reports in SDN is largely an unsolved problem.

While the focus of our work is on under-reporting traffic load statistics, how to leverage over-reporting to undermine load balancing remains an open question. However, there are several unique challenges in crafting an effective over-reporting attack strategy. Since over-reporting causes the load balancer to divert traffic away from the compromised switch and toward other switches, the effects of misreporting become less severe (as the diverted load is distributed to other switches over time according to some unknown distribution). The distribution changes at other pool members then need to be accurately modeled to understand the direct effects of misreporting (in contrast, under-reporting induces direct effects at the compromised switch that can be easily measured and provide feedback for the attacker). Modeling the load distributions on the rest of the pool and crafting an effective attack strategy (that balances the misreporting frequency vs. damage tradeoff) in this setting would therefore require significantly more additions to the current attack framework and thus we consider this to be future work.

We have demonstrated the limitations in existing systems and here discuss potential mitigations against this class of misreporting attacks. Modern datacenter network traffic is characterized by several dynamic network events, including traffic bursts, link failures, and migrating hosts. Quarantining a potential malicious switch incurs a performance and monetary cost to the network and business, and it is therefore necessary that a defense system cross-validate reports with multiple points (including other switch load reports

and other types of reports) before deciding that an event is anomalous and taking a switch offline to investigate. As discussed, current systems cannot account for adversaries (or potentially colluding ones) dropping traffic on edge links, where one end of the link is not SDN-enabled (the host).

A possible solution is to collect statistics from upstream switches/links and use those to indirectly validate the accuracy of the edge load report (i.e., enhancing the FlowMon method).

Another potential solution is to aggregate load reports to identify noticeable changes in traffic volume and use that information to validate the accuracy of the changes claimed by a (compromised) switch's load reports. However, it has already been shown that collecting statistics too frequently or from too many sources causes significant overhead and is impractical [27]. Thus, finding optimal strategies for collecting statistics that are tuned to the purpose of validating load reports remains an open problem. Moreover, quantifying the direct and indirect effects of intelligent misreporting attacks (i.e., those leveraging a reconnaissance stage to attack more effectively) against other SDN-based services will be useful in developing more comprehensive defenses.

6 Related work

The work presented in this paper relates to three well studied areas of research: network load balancing, SDN control-plane security, and distributed trust in networked environments. In Section 2, we discussed how load balancers operate and are realized in SDNs, and in this section we focus on the latter two areas.

6.1 SDN control-plane security

Our work focuses on modeling and evaluating misreporting attacks against load balancers in SDN. In designing the attacks, we draw from prior work studying the security of other SDN services to identify vulnerable points at which to attack the load balancer. Dridi et. al [51] found that adversaries can directly launch denial-of-service attacks against the control plane to saturate various network services at the controller, for example, the service that computes routes for newly arriving flows. Yoon et. al [15] performed a comprehensive analysis of the SDN control plane and found that most controllers have several vulnerabilities, including overflowing of switch tables to evict rules and/or exhaust resources and induce a disconnect, and spoofing switch or host identities to enable eavesdropping attacks. Further, Hong et. al [13] demonstrated that an adversary can inject malicious control messages (specifically, malicious link-discovery messages) to cause the topology discovery service to either eliminate links from the minimum spanning tree or direct traffic through specific links for eavesdropping attacks. Other works [28–31, 38, 52, 53] also demonstrated that an adversary can compromise the control plane through man-in-the-middle attacks over control messages (e.g., modifying other switches' link-discovery packets) and lack of access control at controller application interfaces, among others. The inherent vulnerability the attacks exploit is the fact that the network services are out-of-band and thus lack visibility into the true state of the network, instead relying on untrustworthy sources of information to make control decisions.

A related body of research has partially addressed the vulnerabilities but, as we demonstrated in Section 4.4, are not a fit for all types of network services. The state-of-the-art detection system Sphinx [48] addresses attacks focused on corrupting network topology and data-plane forwarding state. With respect to detecting packet dropping, the core of the system rests on the assumption that edge switches are reporting accurate traffic loads, which breaks down in the case of our misreporting attacks, among others launched from edge switches. Related systems such as Spiffy [54] and OpenWatch [50] specifically target short-term volume-based attacks by malicious hosts, but lack the necessary architectural features that monitor (potentially malicious) switches and/or account for noisy datacenter traffic. While change-detection techniques have been an effective tool for various purposes including offline analysis of data to identify network intrusions [46], anomaly detection in SDN control messages (including load reports) is an online problem with tight time constraints, as a non-trivially long attack against core network services may impose a great cost the network operator.

6.2 Distributed trust in networked environments

The problem that enables the discussed attacks is the controller's lack of visibility into network events. There is an inherent trust assumption between the controller and each switch, as well as among the switches themselves. This problem has been well-studied in other contexts such as wireless communications networks, where a node might similarly report false messages in order to gain additional utility for itself at the expense of others [55]. For example, nodes in mobile ad hoc networks (MANETs) cooperate in forwarding packets from sources to destinations, and thus might report false routing information to blackhole traffic or otherwise avoid having to expense energy to route packets [56]. Nodes in a vehicle ad hoc network (VANET) might falsely report road conditions to clear roads for their own use [57]. Similarly, nodes in wireless sensor networks cooperate in collecting accurate measurements, and thus might report false measurements to corrupt sensor data (e.g., to fabricate physical events) [58].

Trust and reputation management (TRM) systems have been developed alongside anomaly detection systems to allow the network to systematically identify and respond to such misbehaving nodes [59]. These systems use methods for iteratively updating trust and reputation scores to identify malicious behavior, and use methods for filtering out

likely-falsified reports (e.g., a node deliberately flooding the network with bad feedback about its peers) and gradually building trust among peers in order to deter attacks.

While SDN is considered to be centralized, the services still require a distributed form of trust to operate, and therefore may be able to leverage known techniques to comprehensively address the problem of misreporting. Recent work [60, 61] proposed frameworks for establishing trusted control-plane operations based on trust values assigned to controllers (that form a distributed controller) or controller applications. However, these systems do not address the core of the misreporting problem: trust between the control and data plane elements. Therefore, validating control messages in SDN is still largely an unsolved problem, and the extent to which an adversary can exploit control applications from a malicious switch is unknown.

7 Conclusion

As load balancers are a key feature of modern networks, protecting the integrity of their decisions is critical. To provide this, it is necessary that traffic measurements accurately reflect the true state of the network. While prior work has addresses malicious control messages in the context of other services (for example, link discovery), here we proposed an attack model against SDN-based load balancers to control the volume of traffic flowing toward a compromised switch. We found that the adversary could launch an effective misreporting attack (with configurable attack parameters), being able to obtain targeted (unfair) proportions of load in the system. We then observed that the adversary had diminishing returns when misreporting at higher frequencies when flows are generally longer, with no noticeable effect when flows are very short. Further, we found that existing defense systems have several limitations them render them unable to detect such misreporting attacks in practice, although statistical measures combined with systematic approaches to collecting load reports are a potential solution to the misreporting problem. We hope that our results will provide insight into how adversaries might exploit the control/data plane relationship to launch more calculated attacks and lead to th.e development of more comprehensive defenses.

Funding This research was 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. This work was also supported in part by the National Science Foundation under award CNS-1946022.

References

1. Project floodlight (2011) http://www.projectfloodlight.org/floodlight/. [Online; accessed 19-October-2018]
2. Opendaylight project (2013) https://www.opendaylight.org/. [Online; accessed 19-October-2018]
3. Aslam S, Shah MA (2015) Load balancing algorithms in cloud computing: a survey of modern techniques. In: 2015 National Software Engineering Conference (NSEC). IEEE, Rawalpindi, Pakistan, pp 30–35
4. Handigol N, Seetharaman S, Flajslik M, McKeown N, Johari R (2009) Plug-n-serve: Load-balancing web traffic using openflow. ACM Sigcomm Demo 4(5):6
5. Li J, Chang X, Ren Y, Zhang Z, Wang G (2014) An effective path load balancing mechanism based on SDN. In: 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications. IEEE, Beijing, China, pp 527–533
6. Mesbahi M, Rahmani AM (2016) Load balancing in cloud computing: a state of the art survey. Int J Modern Educ Comput Sci 8(3):64
7. Qilin M, Weikang S (2015) A load balancing method based on SDN. In: 2015 Seventh International Conference on Measuring Technology and Mechatronics Automation. IEEE, Nanchang, China, pp 18–21
8. Zhang J, Yu FR, Wang S, Huang T, Liu Z, Liu Y (2018) Load balancing in data center networks: A survey. IEEE Commun Surv Tutorials 20(3):2324–2352
9. Guo Z, Su M, Xu Y, Duan Z, Wang L, Hui S, Chao HJ (2014) Improving the performance of load balancing in software-defined networks through load variance-based synchronization. Comput Netw 68:95–109
10. Neghabi AA, Jafari Navimipour N, Hosseinzadeh M, Rezaee A (2018) Load balancing mechanisms in the software defined networks: A systematic and comprehensive review of the literature. IEEE Access 6:14159–14178. https://doi.org/10.1109/ACCESS.2018.2805842
11. Benson T, Akella A, Maltz DA (2010) Network traffic characteristics of data centers in the wild. In: Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. ACM, New York, NY, pp 267–280
12. Rao A, Legout A, Lim Ys, Towsley D, Barakat C, Dabbous W (2011) Network characteristics of video streaming traffic. In: Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies. ACM, New York, NY, pp 1–12
13. Hong S, Xu L, Wang H, Gu G (2015) Poisoning network visibility in software-defined networks: New attacks and countermeasures. NDSS 15:8–11
14. Khan S, Gani A, Wahab AWA, Guizani M, Khan MK (2016) Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art. IEEE Commun Surv Tutorials 19(1):303–324
15. Yoon C, Lee S, Kang H, Park T, Shin S, Yegneswaran V, Porras P, Gu G (2017) Flow wars: Systemizing the attack surface and defenses in software-defined networks. IEEE/ACM Trans Netw (TON) 25(6):3514–3530
16. Kang N, Ghobadi M, Reumann J, Shraer A, Rexford J (2014) Niagara: scalable load balancing on commodity switches. Tech Rep, Technical Report (TR-973–14), Princeton
17. Wang R, Butnariu D, Rexford J et al (2011) Openflow based server load balancing gone wild. Hot-ICE 11:12–12
18. Mahmood A, Rashid I (2011) Comparison of load balancing algorithms for clustered web servers. In: ICIMU 2011: proceedings of the 5th international Conference on Information Technology & Multimedia. IEEE, Kuala Lumpur, Malaysia, pp 1–6

19. Zhou Y, Zhu M, Xiao L, Ruan L, Duan W, Li D, Liu R, Zhu M (2014) A load balancing strategy of SDN controller based on distributed decision. In: 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications. IEEE, Beijing, China, pp 851–856
20. Arau´jo JT, Saino L, Buytenhek L, Landa R (2018) Balancing on the edge: Transport affinity without network state. In: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), pp. 111–124
21. The netfilter.org project (1998). URL https://www.netfilter.org/
22. Eisenbud DE, Yi C, Contavalli C, Smith C, Kononov R, Mann-Hielscher E, Cilingiroglu A, Cheyney B, Shang W, Hosein JD (2016) Maglev: A fast and reliable software network load balancer. In: 13th USENIX Symposium on Networked Systems Design and Implementation ({NSDI} 16), pp. 523–535
23. Patel P, Bansal D, Yuan L, Murthy A, Greenberg A, Maltz DA, Kern R, Kumar H, Zikos M, Wu H et al (2013) Ananta: Cloud scale load balancing. ACM SIGCOMM Comput Commun Rev 43(4):207–218
24. Openflow switch specification (2015) https://www.opennetworking.org/software-defined-standards/specifications/. [Online; accessed 19-October- 2018]
25. Raghavan B, Vishwanath K, Ramabhadran S, Yocum K, Snoeren AC (2007) Cloud control with distributed rate limiting. In: ACM SIGCOMM Computer Communication Review, vol. 37. ACM, New York, NY, pp 337–348
26. Feghhi S, Leith DJ (2016) A web traffic analysis attack using only timing information. IEEE Trans Inf Forensics Secur 11(8):1747–1759
27. Aslan M, Matrawy A (2016) On the impact of network state collection on the performance of sdn applications. IEEE Commun Lett 20(1):5–8
28. Arbettu RK, Khondoker R, Bayarou K, Weber F (2016) Security analysis of opendaylight, onos, rosemary and ryu sdn controllers. In: 2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks). IEEE, Montreal, QC, pp 37–44. https://doi.org/10.1109/NETWKS.2016.7751150
29. Benzekki K, El Fergougui A, Elbelrhiti Elalaoui A (2016) Software-defined networking (sdn): a survey. Secur Commun Netw 9(18):5803–5833
30. Dargahi T, Caponi A, Ambrosin M, Bianchi G, Conti M (2017) A survey on the security of stateful sdn data planes. IEEE Commun Surv Tutorials 19(3):1701–1725
31. Scott-Hayward S, O'Callaghan G, Sezer S (2013) SDN security: a survey. In: 2013 IEEE SDN For Future Networks and Services (SDN4FNS). IEEE, Trento, Italy, pp 1–7
32. Thimmaraju K, Schiff L, Schmid S (2017) Outsmarting network security with sdn teleportation. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, Paris, France, pp 563–578
33. Pfaff B, Pettit J, Koponen T, Jackson E, Zhou A, Rajahalme J, Gross J, Wang A, Stringer J, Shelar P et al (2015) The design and implementation of open vSwitch. In: 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15). USENIX Association, Oakland, CA, pp 117–130
34. Thimmaraju K, Shastry B, Fiebig T, Hetzelt F, Seifert JP, Feldmann A, Schmid S (2018) Taking control of sdn-based cloud systems via the data plane. In: Proceedings of the Symposium on SDN Research. ACM, New York, NY, pp 1–15
35. Chica JCC, Imbachi JC, Vega JFB (2020) Security in sdn: A comprehensive survey. J Netw Comput Appl 159:102595
36. Benson T, Anand A, Akella A, Zhang M (2009) Understanding data center traffic characteristics. In: Proceedings of the 1st ACM workshop on Research on enterprise networking, pp. 65–72. ACM
37. Burke Q, McDaniel P, Porta TL, Yu M, He T (2020) Misreporting attacks in software-defined networking. In: EAI SecureComm 2020 – 16th EAI International Conference on Security and Privacy in Communication Networks. EAI, pp 1519–1528
38. Achleitner S, La Porta T, Jaeger T, McDaniel P (2017) Adversarial network forensics in software defined networking. In: Proceedings of the Symposium on SDN Research, SOSR '17. ACM, New York, NY, USA, pp 8–20. https://doi.org/10.1145/3050220.3050223
39. Kampanakis P, Perros H, Beyene T (2014) SDN-based solutions for moving target defense network protection. In: Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014. IEEE, Sydney, NSW, pp 1–6
40. De Oliveira RLS, Schweitzer CM, Shinoda AA, Prete LR (2014) Using mininet for emulation and prototyping software-defined networks. In: 2014 IEEE Colombian Conference on Communications and Computing (COLCOM). IEEE, Bogota, Colombia, pp 1–6
41. Qian H, Medhi D (2011) Server operational cost optimization for cloud computing service providers over a time horizon. In: HotICE. USENIX Association, Boston, MA
42. Curtis AR, Mogul JC, Tourrilhes J, Yalagandula P, Sharma P, Banerjee S (2011) Devoflow: scaling flow management for high-performance networks. In: ACM SIGCOMM Computer Communication Review, vol 41. ACM, New York, NY, pp 254–265
43. Greenberg A, Hamilton JR, Jain N, Kandula S, Kim C, Lahiri P, Maltz DA, Patel P, Sengupta S (2009) Vl2: a scalable and flexible data center network. In: ACM SIGCOMM computer communication review, vol 39. ACM, New York, NY, pp 51–62
44. Chandrasekaran SS (2017) Understanding traffic characteristics in a server to server data center network. Thesis, Rochester Institute of Technology
45. Aweya J, Ouellette M, Montuno DY, Doray B, Felske K (2002) An adaptive load balancing scheme for web servers. Int J Network Manage 12(1):3–39
46. Ahmed M, Mahmood AN, Hu J (2016) A survey of network anomaly detection techniques. J Netw Comput Appl 60:19–31
47. Caberera J, Ravichandran B, Mehra RK (2000) Statistical traffic modeling for network intrusion detection. In: Proceedings 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No. PR00728). IEEE, San Francisco, CA, pp 466–473
48. Dhawan M, Poddar R, Mahajan K, Mann V (2015) Sphinx: detecting security attacks in software-defined networks. Ndss 15:8–11
49. Kamisin´ski A, Fung C (2015) Flowmon: detecting malicious switches in software-defined networks. In: Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense. ACM, New York, NY, pp 39–45
50. Zhang Y (2013) An adaptive flow counting method for anomaly detection in SDN. In: Proceedings of the ninth ACM conference on Emerging networking experiments and technologies. ACM, New York, NY, pp 25–30
51. Dridi L, Zhani MF (2016) SDN-guard: Dos attacks mitigation in sdn networks. In: 2016 5th IEEE International Conference on Cloud Networking (Cloudnet). IEEE, Pisa, Italy, pp 212–217
52. Achleitner S, Burke Q, McDaniel P, Jaeger T, Porta TL, Krishnamurthy S (2019) MLSNet: A Policy Complying Multilevel Security Framework for Software Defined Networking. Tech. Rep. INSR-500-TR-0500–2019, Institute of Networking and Security Research, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA. http://patrickmcdaniel.org/papers-ct.html
53. Yu M, He T, McDaniel P, Burke QK (2020) Flow table security in SDN: adversarial reconnaissance and intelligent attacks. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications. IEEE, Toronto, ON, pp 1519–1528
54. Kang MS, Gligor VD, Sekar V et al (2016) Spiffy: Inducing cost-detectability tradeoffs for persistent link-flooding attacks. NDSS 1:53–55
55. Yu H, Shen Z, Miao C, Leung C, Niyato D (2010) A survey of trust and reputation management systems in wireless communications. Proc IEEE 98(10):1755–1772

56. Tan S, Li X, Dong Q (2015) Trust based routing mechanism for securing oslr-based manet. Ad Hoc Netw 30:84–98
57. Zhang J (2011) A survey on trust management for VANETs. In: 2011 IEEE International Conference on Advanced Information Networking and Applications. IEEE, Biopolis, Singapore, pp 105–112
58. Lopez J, Roman R, Agudo I, Fernandez-Gago C (2010) Trust management systems for wireless sensor networks: Best practices. Comput Commun 33(9):1086–1093
59. Buchegger S, Le Boudec JY (2002) Performance analysis of the confidant protocol. In: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing. ACM, New York, NY, pp 226–236
60. Aliyu AL, Bull P, Abdallah A (2017) A trust management framework for network applications within an SDN environment. In: 2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA). IEEE, Taipei, Taiwan, pp 93–98
61. Betg´e-Brezetz S, Kamga GB, Tazi M (2015) Trust support for SDN controllers and virtualized network applications. In: Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft). IEEE, London, UK, pp 1–5