Misreporting Attacks in Software-Defined Networking

Quinn Burke, Patrick McDaniel,
Thomas La Porta, Mingli Yu, and Ting He

The Pennsylvania State University, State College, PA 16801, USA
qkb5007@psu.edu,mcdaniel@cse.psu.edu,
{tfl12,mxy309}@psu.edu,t.he@cse.psu.edu

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 use a queuing theoretic 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 draw nearly all of the load in the subnetwork (+750%, or 85% of the load in the system), or an adversary-desired amount of load (a target load, e.g., +200%) to within 12% 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 afloat. 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 differently. The load balancer is divided into two components: the application logic (e.g., load balancing algorithm) that does the decision making and the network switches that enforce the decisions via flow rules. Here, the network switches are employed to report traffic loads (switch statistics) to the controller

application to decide where to route incoming flows. While offering scalability and reliability benefits, this separation also creates an opportunity for an adversary at a compromised switch to misreport the traffic loads to influence load balancing.

In this paper, we evaluate an adversary's ability to control the amount of traffic flowing through the compromised switch (for eavesdropping and traffic analysis) by misreporting traffic loads. We take a queuing theoretic 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: the *max-flooding* attack to draw as much load as possible and the *stealthy* attack to draw a target amount (an adversary-desired amount) of traffic through the compromised switch. We then evaluate them against four widely used load balancing algorithms: *least-loaded*, *weighted least-loaded*, *least-connections*, and *weighted least-connections*, which are included in the widely used Floodlight's [2] and OpenDayLight's [3] load balancing modules, and relied upon by several other specialized load balancing solutions [45, 34, 7, 25, 32, 38]. 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 [35, 24]. The wide reliance on this calculation provides motivation for 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) [10, 40]. 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 draw 750% additional load (85% of the load in the subnetwork) through aggressive misreporting, or draw a target amount of additional load to within 12% error of that target. We also find that the queuing model accurately describes the network behavior in response to misreporting to within 12% of the predicted throughput and 7% of the predicted number of misreports. Thus it is an effective tool for performing reconnaissance and 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 [44, 30, 26]; 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 compromised switches 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.

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., underutilization, increased latency). Common implementations of static load balancing include Randomized, Round-Robin, and hash-based solutions like equal-cost multipath (ECMP) [7, 29, 42]. 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 [34, 33, 47, 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 [5]. Previous work has already demonstrated the ability of load balancers to be implemented as software running on commodity hardware [19, 36, 43, 1]. In SDNs, however, load balancing typically manifests slightly differently. The load balancer is abstracted from the physical infrastructure that it is deployed on by moving the load balancing logic to the control plane and distributing decision to network switches in the form of flow rules.

To enable dynamic load balancing in SDNs, the network administrator first defines a *pool*: a list of *switch ports* connected to links to load balance for (see Figure 1). These switch ports, or pool members, then become participants in

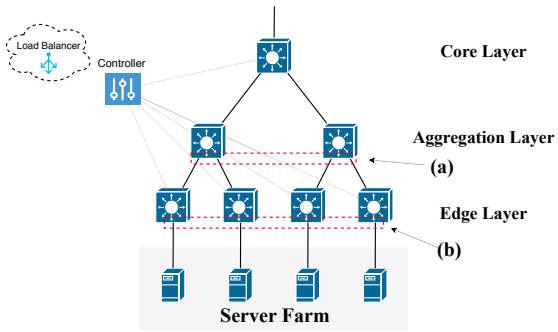

Fig. 1. Pool members for SDN-based load balancing across (a) links and (b) servers.

the load balancing of the pool. The load balancer requests traffic load reports from them at each time epoch t, where epochs may be separated by one or more seconds. We will refer to this epoch length, the time between load-report collections, as the *collection interval*.

Under the OpenFlow [4] 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 to fairly route new incoming flows (that are destined for the resources offered by the pool); for example, with a variant of *least-X* selection.

As shown in Figure 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 1Kb of activity in the first epoch, has new flows routed through it to a backend server or link, and reports 12Kb 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.

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 [34], 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.

For weighted least-loaded and weighted least-connections, an exponentially weighted moving average (EWMA [39]) of loads is used for balancing. Weights are applied to the historical load values (α, where $0 \leq \alpha \leq 1$) and the current

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

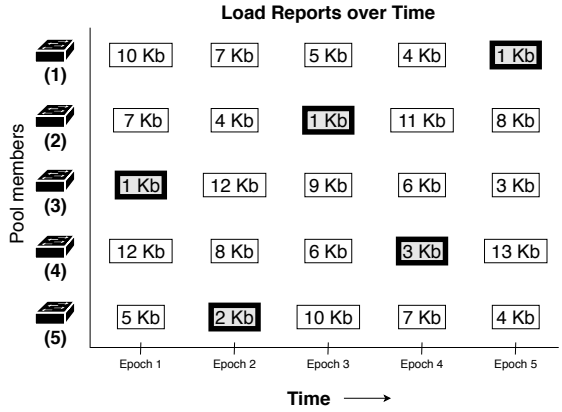

Fig. 2. Load reports (R_t^i) used for routing new incoming flows. Bolded reports are where switches reported the minimum load to the load balancer.

load value $(1 - \alpha)$, which are then summed together, allowing the load balancer to smooth out sudden bursts which may lead to inefficient balancing. Then, the new load $R_t^{i'}$ computed for each member at time t is:

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

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 $R_t^{m\prime}$ and $R_t^{i'}$. Again, random selected is applied in the case of multiple members with the minimum.

Related work. This work focuses on modelling and evaluating misreporting attacks against load balancers in SDN. We draw from prior work on the security of SDN services [15, 6, 41, 12] to identify vulnerable points in the control plane, which observe that an adversary at a compromised switch can manipulate link-discovery (LLDP) packets [26] to poison the topology information used by the controller for tracking network hosts and performing routing functions. Moreover, other works have found that adversaries can directly launch denial-of-service attacks against the control plane to saturate functions at the controller, for example, the service that computes routes for incoming flows [18]. Other works evaluate other vulnerabilities, including data modification and leakage, malicious control applications, lack of security features such as encryption, etc. [17]. Recent work also proposed a load-balancer attack in more traditional network architectures which requires sending probes from a network host [23]. Our work differs in that we consider misreporting (switch statistics) in the context of load balancing. problem Lastly, others have proposed defense systems to protect against some of these different classes of attacks [31, 28, 17, 46], but these systems are not applicable to this attack scenario.

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 cloud networks (e.g., to uncover browsing histories [20]). Here, we introduce two attack methods against two distinct network traffic patterns.

3.1 Threat Model and Overview

Threat model. We assume switches report aggregate (i.e., port-level) statistics to a trusted load balancer, as balancing is typically done at a coarser level than individual flows [8]. Of these switches, we assume that one becomes compromised. If there is a single switch reporting for an entire pool (as with dedicated load balancers) and the switch becomes compromised, then load balancing integrity is clearly lost. We consider the situation where multiple switches faithfully report statistics for the pool and one becomes compromised[2]. Switches may be compromised by either an insider or external adversary [15, 6, 41, 12]; however, methods for carrying out attacks are outside the scope of this work. The adversary may also be located at either the edge (balancing across servers) or aggregation layer (balancing across links) of the network.

In the context of load balancing, we define the general adversarial goal as 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 if packets on the switch ports are actually dropped; although, 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 unfair proportion of traffic, and we leave over-reporting attacks (to deny service downstream or overload other switches) to future work.

Overview. Studies of modern datacenter and cloud networks have identified two distinct patterns in network traffic among different cloud services. The first consists of a majority of small (in bytes) and short (in seconds) flows that exist only momentarily in the network. This traffic is representative of applications such as web servers. The second consists 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. We draw from these studies [10, 40] to generate packet traces for each pattern, consisting of flows with sizes and durations randomly selected along two pareto curves (Section 4). Preliminary observations shown in Figure 3 with the Floodlight [2] SDN controller are representative of well-known traffic loads observed in the wild [10, 40]. Note that load balancing occurs on a per-pool basis, and since pool members would be serving similar kinds of services, they would see similar traffic characteristics [10]. Nonetheless, our preliminary observations of these traffic patterns across a pool of servers reveal two threat vectors for an adversary to compromise the load balancer.

Short flows. In the context of short flows, a majority (>80%) of flows entering the network lasts less than one second [10, 27]. The result is network switches

[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.

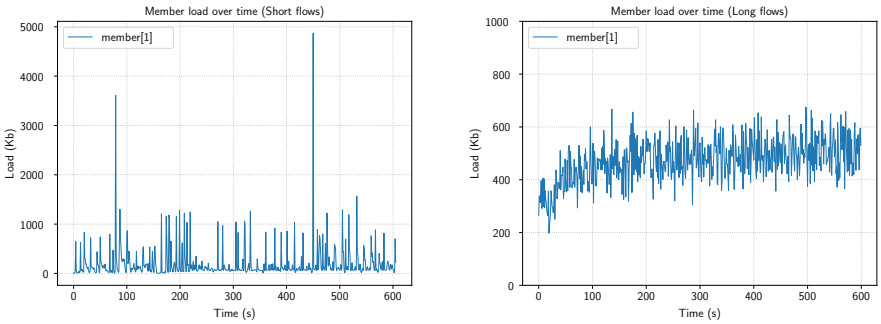

Fig. 3. 10-minute captures of load reports of a single switch in two different scenarios. With traffic dominated by short flows (left) the switch observes momentary load spikes, and for long flows (right) a more stable load over time as active flows persist.

periodically observing momentary *load spikes* [10] as batches of incoming flows are temporarily routed through them. The load at such a switch will fall back down to normal levels (i.e., only overhead from control traffic) within just a few epochs (and load reports) as the flows expire quickly, and while the load balancer selects a different member as the minimum. This can be seen in Figure 3 (left), where the load may be very high at one epoch (e.g., 1000Kb) and then very low by the next (e.g., <10Kb).

The key insight here is that the load reported by pool members is constituted by these momentary load spikes, as opposed to showing a more stable (or flatter) observed load over time. The rest of the load reports will show low activity, until more flows are routed through the switch. Thus, for an adversary to draw more traffic through the compromised switch, they must misreport (under-report) to induce more load spikes. Misreporting can exploit the load balancer's *least- X* calculation to cause the load balancer to immediately begin routing new flows through the switch, creating another load spike. The challenge is determining an appropriate number of misreports to draw the target load through the switch.

Long flows. In the context of long flows, the observed loads of pool members are dominated by persistent activity of longer-lived flows rather than momentary bursts. As a result, pool members observe a steadier (or flatter) load distribution over time. The behavior in Figure 3 (right) is consistent with this. The key insight here is that since the load reported by pool members is constituted by longer-lived flows, drawing more load is based on increasing the number of flows actively sending data through the compromised switch. The challenge here then becomes finding a suitable number of times to misreport to induce a certain number of active flows traversing the switch at any given moment.

We formalize misreporting in terms of a target utilization at a port on the compromised switch. We will refer to this target as ρ_{tar}. We then introduce two misreporting attacks with respect to ρ_{tar}. In the trivial attack, the *max-flooding* attack, the goal is to draw as much traffic as possible (e.g., $\rho_{tar} = 1.0$ utilization) through the switch port. In the *stealthy* attack, the goal is to draw a target volume of traffic (e.g., $\rho_{tar} = 0.2$) through the switch. This allows the adversary

to manage the risk they are exposing themselves to by only misreporting the necessary amount to increase the utilization to the target.

With this formulation, the adversary must calibrate two parameters to draw the target utilization. $\overline{L}$ is the average load, the meaning of which differs slightly depending on the considered traffic pattern. It is used to determine *how many* misreports must be sent in order to draw ρ_{tar} through the switch port. The second parameter is δ, the misreported amount (e.g., in Kb) sent to the load balancer. It determines *by how much* to misreport by, and the choice of which will affect the success rate of misreporting; i.e., if the load balancer immediately begins routing new flows through the switch.

3.2 Attack Model

We introduce an M/D/1-based discrete-time queueing model (following prior work [23]) to approximate the behavior of the output switch port, and later validate the model accuracy on an experimental network. Here, we assume flow arrivals are determined by a Poisson process and service times are fixed (to transmit each bit). The model allows the adversary to derive attack parameters from model parameters for a given ρ_{tar}, and also serves to assess the effect of the attack on network performance. We provide a table of parameters in Appendix A.

Under an M/D/1 model, the utilization ρ of the switch port is given by:

$$\rho = \frac{\lambda}{\mu}, \tag{3}$$

where λ is the arrival rate (in bits per second, or bps) at the network interface card on the port, and μ is the service rate (in bps) of the card, fixed across all pool members as they serve similar services. For a given target utilization ρ_{tar}, there exists some target arrival rate of bits λ_{tar} that the adversary wishes to draw through the switch port:

$$\rho_{tar} = \frac{\lambda_{tar}}{\mu} \tag{4}$$

The adversary must then estimate the necessary number of misreports to draw λ_{tar} through the switch port.

Short flows. Let M denote the number of misreports required to achieve the goal. As network traffic dominated by short flows is characterized by momentary load spikes, whenever the member reports the minimum load, a load spike will occur (see Figure 3). Note that a load spike occurs whether the reported value was a misreport or not. However, a single successful misreport therefore corresponds with a single load spike. Then, the required number of misreports M to draw λ_{tar} can be approximated with knowledge of the amount of load (in bits or number of flows) contained in a load spike, or its *amplitude*. If $\overline{L}$ is the average load spike amplitude, then the number of misreports necessary to draw an average of λ_{tar} on the switch port over an attack window of W epochs is given by:

$$M = \frac{\lambda_{tar} \times W}{\overline{L}} \tag{5}$$

For offline analysis, we can approximate $\overline{L}$ for the least-connections algorithms by first considering an average flow inter-arrival rate of R flows per second [10]. Note that the adversary will compute the actual value at runtime. Since all of the flows are temporarily routed through the compromised switch for the next epoch, then the average load spike amplitude $\overline{L} = R$ flows. For the least-loaded algorithms, we also consider that the network flows have an average size of $\overline{f}$ bytes, based on characteristics of network flows observed in prior work [10]. Then, the average load contained in any load spike is $\overline{L} = 8R\overline{f}$ bits.

Long flows. For network traffic dominated by long flows, the load in the network depends more on the number of active flows sending data through a switch rather than the amplitude of momentary load spikes, which are not as significant in this scenario compared to the number of active flows. Here, a single successful misreport corresponds with a set of long-lived flows being scheduled through the compromised switch. We therefore propose a heuristic method to drawing λ_{tar} on the port: *batch misreporting*. Specifically, the adversary will report consecutively a fixed number of times starting at the beginning of every t_{long}-second time slot, where t_{long} represents the average duration of the long-lived flows [40]. By misreporting in batches, or in consecutive epochs, the adversary can influence the load balancer to schedule an additional set of flows through the switch whose lifespan will nearly overlap in time. Then, the next batch of misreports will replace those expiring flows with new ones.

If we let $\overline{L}$ represent the average load observed at the switch port, the number of consecutive misreports to send, or the batch size B, is computed as the multiplicative factor of λ_{tar} over $\overline{L}$:

$$B = \frac{\lambda_{tar}}{\overline{L}} \tag{6}$$

For example, if the target load is 2Mb/s, the average load is 1Mb/s, and flows have an average duration of 10 seconds, the adversary will misreport in batches of 2 at the beginning of every 10-second time slot to double the number of active flows traversing the switch. The required number of misreports M is:

$$M = \frac{B}{t_{long}} \times W \tag{7}$$

where the adversary misreports B times out of every t_{long} seconds, for the duration of the attack window.

We can approximate $\overline{L}$ for the least-connections algorithms by $\overline{L} = R \times t_{long}/N$ flows, where N is the number of pool members, and $\overline{L}$ represents the steady-state average load at any pool member. Note that for short flows we assume an average flow size $\overline{f}$ as the entire flow is consumed before the next epoch. If network flows have an average flow rate of $\overline{p}$ bytes [10, 40], then for the least-loaded algorithms, we have similarly: $\overline{L} = 8R\overline{p} \times t_{long}/N$ bits. Note that the adversary will compute the actual value of $\overline{L}$ at runtime.

3.3 Max-flooding Attack

In this attack, the goal of the adversary is to maximize the volume of traffic flowing through the compromised switch ($\rho_{tar} = 1.0$). The adversary can trivially

perform the attack by misreporting every time the load balancer requests a load report. Specifically, here the number of misreports is $M = W$, each epoch for the entire duration of the attack window. Without loss of generality, we denote the compromised switch by switch N. To maximize the probability that the misreported load will be the minimum in (1), the adversary will set δ to zero (0 bytes, 0 flows, etc.), sending a new load ($R_t^{N\prime\prime}$) in each epoch:

$$R_t^{N\prime\prime} = \delta = 0 \tag{8}$$

The goal is to draw all flows arriving during the attack window through the compromised switch for larger scale eavesdropping and traffic analysis, and also may create congestion at the server connected by the switch port. Although feasible, the attack may also become readily observable.

3.4 Stealthy Attack

In the second attack, we generalize the max-flooding approach to allow the adversary to more stealthily attack the load balancer. A stealthy attack is one in which the adversary manages their detectability by drawing a λ_{tar} that is less than the maximum (less than maximum utilization). 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 is stealthy. Then, to meet the idea of stealthy, the adversary must reduce the amount of misreporting to only that required to draw λ_{tar} on the switch port. To accomplish this, we divide the attack into two phases: the reconnaissance phase and the attack phase.

Phase 1: Reconnaissance. In contrast to the max-flooding attack, here, $\overline{L}$ plays the critical role in determining the number of misreports that must be sent for a given λ_{tar}. Depending on what traffic conditions are present in the network—which we assume the adversary has some knowledge of or can infer from its own load distribution—$\overline{L}$ is computed in one of two ways. With an estimate for $\overline{L}$, the adversary must then find an appropriate δ.

Estimating $\overline{L}$: Short flows. To estimate the amplitude of any load spike, we propose a heuristic method for detecting load spikes, then take the average as the estimate. The adversary first must select a threshold P at which an observed load should be considered a load spike. For example, if an observed load is greater than the 99th percentile of all observed loads seen thus far, it will be considered a load spike, since the majority of observed loads are much lower (and spikes are short-lived). We first let the adversary perform a warm-up phase (e.g., 10 minutes) to fill a list *observed_loads* of observed loads before detecting load spikes. Then, the adversary maintains a list S of load values considered load spikes (see Alg. 1, lines 5-9, in Appendix B), until D load spikes have been detected. The average is then taken as the estimated amount of load concentrated in any momentary load spike whenever a pool member reports the minimum load. If the list of detected

load spikes S has size D, then the average load spike amplitude $\overline{L}$ is given by:

$$\overline{L} = \frac{\sum_{i=1}^{D} S_i}{D} \tag{9}$$

Given the average load spike amplitude (Kb or number of new flows) calculated during reconnaissance, the adversary computes the required number of misreports with (5). Note that misreports can be spaced out evenly with a misreporting period of T_m, or randomized with an average misreporting period of T_m:

$$T_m = \frac{\overline{L}}{\lambda_{tar}} \tag{10}$$

After the period is set, the adversary exits the reconnaissance phase (see Alg. 1, line 13, in Appendix B).

Estimating $\overline{L}$: Long flows. In this network setting, to draw more load, the adversary must increase the number of active flows sending packets through the switch. $\overline{L}$ can similarly be computed as simply the average load observed over an arbitrary window of time. If this window for reconnaissance is D epochs long, then $\overline{L}$ is given by:

$$\overline{L} = \frac{\sum_{i=1}^{D} observed\_loads_i}{D} \tag{11}$$

Given this, the number of misreports required to draw λ_{tar} through the switch is then given by (7). Unlike the network dominated by short flows, here misreports must be batched to have the target number of flows active as soon as possible.

Estimating δ. We previously assumed that whenever the adversary sent a misreport, the load balancer would certainly begin routing new flows through the compromised switch. Although guaranteeing a 100% misreporting success rate is difficult, sending a load of zero in each misreport will provide the highest probability of success. However, sending a load of zero in each misreport may likely raise alarms, especially if the desired load is very high (e.g., +500% load than usual) and thus so is the misreporting frequency. To meet the idea of stealthy, a better 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 have nearly the same probability of drawing a load spike as 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 in this case.

To this end, we first observe that the observed-load distributions of all pool members (for either network traffic pattern) show small differences, which reflects observations made in prior work [11, 10] of network switches observing similar traffic characteristics. Given this, we approximate the load distribution observed at other pool members by that observed and recorded by the adversary during reconnaissance. Then, if we first let U denote a cumulative probability of the load distribution, then there is an associated load value U_L (in Kb/s or number of flows) with that cumulative probability: U percent of observed loads fall within $[0, U_L]$. Then, if there are N pool members, we can express the number of switches

expected to report within some $[0, U_L]$ at any given time epoch as a binomial experiment on random variable X, with probability U of reporting within $[0, U_L]$:

$$E(X) = (N-1) * U, \quad U > 0 \tag{12}$$

For example, if $N = 100$ and $U = 0.01$, approximately 1 switch (not the adversary) is expected to report within the given range (or percentile) any time the load balancer requests a report. The goal of the adversary then becomes selecting a sufficiently low U to misreport to within, to reduce the expectation and therefore have a higher probability of (mis)reporting the minimum load. The adversary will randomly set δ to a previously observed load in $[0, U_L]$. Note that the adversary may not know the pool size; in this case, they should assume a large pool (and a small U), which will still be advantageous if the pool is actually smaller.

Phase 2: Sending the misreport. After determining $\overline{L}$, then with a suitable misreporting period T_m or batch size B, along with a proper δ that should provide a reasonably high misreporting success rate, the adversary can then send the misreport. The adversary would first verify that the current load report collection time epoch falls at the beginning of a t_{long} time-slot for the case of long flows, or if it falls on a misreporting period (T_m) boundary for short flows. If so, the adversary may update the load to an under-reported value satisfying $\delta \in [0, U_L]$:

$$R_t^{N\prime\prime} = \delta \tag{13}$$

If the actual load at the switch port is already below the determined threshold load U_L, the adversary will simply report that amount instead of modifying it. The key idea here is that that the adversary can significantly reduce (to nearly a minimum) the amount of misreporting that must be done to reach the target load rate via simple analyses of the steady-state behavior.

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 effects of the attack on network performance across the compromised switch using an analytical attack model.

Measuring 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_{attack}, and under normal conditions by L_{normal}, then the relative damage is:

$$D = \frac{L_{attack}}{L_{normal}} - 1 \tag{14}$$

To concretely quantify misreporting effectiveness, we introduce a potency metric P that represents the average load increase obtained per misreport:

$$P = \frac{D}{\#\ misreports} \tag{15}$$

Note that M is an upper bound for the number of misreports, as the actual load may be within the misreporting range and the adversary can send the report without modification. Nonetheless, we also measure the rate and success rate of misreporting, which describes how often a misreport resulted in more traffic being routed through the compromised switch.

Measuring the impact on network performance. A natural effect of shifting a large volume of traffic onto the switch port is congestion at the port, which will have a large impact on the throughput of flows traversing the switch port. However, as the utilization is significantly lower (less than 5%) on the servers or links in subnetworks dominated by many short flows, even shifting all of the traffic onto the link will not cause measurable impacts on throughput. Here, we just consider the effect of the attack on the changes in throughput for the long traffic pattern, where the servers and links are continuously being stressed by persistent flows. We measure the throughput as a function of the target ρ_{tar}, as well as a function of the number of misreports.

For a specified ρ_{tar}, the average time spent waiting in an M/D/1 system for each bit (delay per bit) is given by:

$$T_w = \frac{1}{\mu} + \frac{\rho_{tar}}{2\mu(1-\rho_{tar})} \tag{16}$$

Since we know the number of misreports in this network scenario from (7), we can now measure the throughput changes as the target utilization increases.

4 Evaluation

With the formulation of the reconnaissance and attack phases, here, we explore the effects on the performance of the load balancer in several scenarios (shown in Table 1) 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 an analysis of the effectiveness of the two attacks in each scenario and the effects on the network performance when considering long-lived flows (for example, in video streaming applications).

4.1 Experimental Setup

Network setup. For experimentation, we employ the latest version of the widely used Floodlight [2] SDN controller, along with its load balancing module. To configure the virtual network, we use the popular Mininet emulator [16] to create a similar topology of virtual switches and hosts to that shown in Figure 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. And 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 [37]. We note that our experimentation with larger pool sizes yielded qualitatively similar results, where the load is scaled proportionately for the same arrival rate of flows. The switches are directly connected to a single backend resource (which represent either servers, or more switches). We also configure the load balancer to have a load-report collection period of 1 second, which is suitable for providing reasonably low load-error rates [8, 22, 14]. We then consider an average arrival rate of 250 flows/s and 100 flows/s for short and long flows, respectively. Note that smaller or larger arrival rates yielded qualitatively similar results. We set the load spike detection percentile for the short-flow traffic pattern to $P = 0.9$, the 90th percentile load. We set the load threshold for misreporting $U = 0.01$, meaning the adversary will misreport to within the bottom 1th percentile of loads (over a training window of 10 minutes). Note that the success rate decreases with a power-law relationship to the threshold U, and therefore flexibility in parameter choice is limited (see Appendix C). We also set the attack window to $W = 300$ epochs. Simulations are averaged over 25 independent executions. Without loss of generality, the adversary is designated by switch number N.

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 [11, 13]. Following these prior works, for the short traffic pattern we generate a set of flows with an average size of about 1KB and average duration of about 100ms. For longer flows, we generate flows with an average duration of about 10 seconds (for a flow size of about 10KB). Note that experiments with longer flows (and therefore larger in terms of total size) yielded qualitatively similar results. Flow packets are transmitted at an average rate equal to the flow size divided by the duration.

Note that while switches may observe many flows of different types and patterns at any given time, load balancing is application-based (or switch-port based). Therefore, the load reports are only collected from those switch ports in the load balancing pool. Thus, we assume the same traffic pattern across pool members (i.e., either short or long flows) as downstream resources may serve similar services and should therefore see similar traffic patterns [10].

4.2 Experimental results

Short flows. In the first part of the evaluation, we considered short traffic flows. In this scenario, the adversary performed reconnaissance on the load spike amplitude (averaged over 10 load spikes) to compute the required number of misreports to draw the target load through the compromised switch. In Table 1, compared to the average load observed under normal conditions (the control experiment), running the max-flooding attack against the load balancer (using the least-loaded algorithm) was able to effectively draw nearly 85% of the load in the system (i.e., across the pool members) toward the adversary. In fact, the

		Short flows				Long flows			
		LL	WLL	LC	WLC	LL	WLL	LC	WLC
Control	Average load	251.42 Kb/s	247.64 Kb/s	26 flows/s	27 flows/s	1017 Kb/s	929.6 Kb/s	83 flows/s	84 flows/s
Max-flooding	Average load	2142 Kb/s	2104 Kb/s	206 flows/s	209 flows/s	8277 Kb/s	8210 Kb/s	694 flows/s	688 flows/s
	Misreport rate	100%	100%	100%	100%	100%	100%	100%	100%
	Success rate	100%	100%	100%	100%	100%	100%	100%	100%
	Damage	+752%	+749%	+692%	+674%	+714%	+783%	+736%	+719%
	Potency	+2.51%	+2.49%	+2.31%	+2.25%	+2.38%	+2.61%	+2.45%	+2.39%
Stealthy	Target load	750 Kb/s	750 Kb/s	75 flows/s	75 flows/s	3000 Kb/s	3000 Kb/s	250 flows/s	250 flows/s
	Average load	866.70 Kb/s	834.67 Kb/s	93 flows/s	72 flows/s	2630 Kb/s	2650 Kb/s	223 flows/s	220 flows/s
	Misreport rate	32.3%	32.0%	30.3%	29%	23.7%	23%	24.8%	23.1%
	Success rate	96.9%	94.1%	98.6%	96.1%	97.1%	94.3%	98%	97.4%
	Damage	+245%	+237%	+244%	+167%	+159%	+157%	+169%	+162%
	Potency	+2.53%	+2.47%	+2.69%	+1.92%	+2.23%	+2.23%	+2.27%	+2.34%

Table 1. Experimental network results with the Floodlight [2] SDN controller.

max-flooding attack proved to be successful across all four of the considered load balancing algorithms, drawing from 600-800% additional load through the switch compared to normal conditions. The misreporting rate for each was 100% of the attack window, and since the misreported load was zero (and loads must be non-negative), the misreporting success rate was also maximal. This means that each misreport resulted in at least one new flow being routed through the compromised switch, although multiple pool members may have all shared the minimum load at some time epochs.

The stealthy attack showed similar results with respect to misreporting success. Nearly all misreports resulted in the load balancer routing new flows through the compromised switch, allowing the adversary to maintain approximately the target amount of load at the switch port for the duration of the attack, to within 13% of the target (and almost always above the target). We note that the target load was specified to be three times that observed under normal conditions, although the adversary is not restricted to just that. Depending on the choice of P, the computed load spike amplitude may have been lower or higher, resulting in either more or less misreports (respectively). A persistent adversary may take a dynamic approach to misreporting by analyzing the effects and re-calibrating P appropriately to better meet the target.

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., $\alpha = 0.5$) [9], 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.

In terms of direct damage to the system, the results indicate that the attack was successful in drawing approximately the target amount of traffic through the

switch. Regardless of the algorithm in use, the adversary computed a required number of misreports (or alternatively, the misreporting period) that was nearly the same across all algorithms at approximately 30% of the attack window. It follows that the potency of misreports was also approximately the same across all algorithms at about 2% average increase in load per misreport, revealing that neither attack proved to be more or less resistant to the misreporting attack.

Long flows. Next we consider a network dominated by longer flows. At runtime the adversary computed a batch size $B = 3$ from the given target and average load observed. As with short flows, the max-flooding attack shifted nearly 83% of the load in the system toward the compromised switch for an increase of more than 700% load than under normal conditions. The success rate was also at a maximum against each algorithm.

The stealthy attack in the context of long flows also proved to be successful against all four load balancing algorithms. In this case, the adversary was able to draw an amount of load through the switch to within 12% of the target. As with short flows, using the results as a feedback mechanism for a more dynamic attack is possible for adjusting parameters to better meet the goal. However, there is one difference here from the case of short flows: the average load was always under the target. In contrast to short flows, where the attack exploited the fact that load was concentrated in load spikes, here the fact that a fixed (average) number of flows arrive each second means that the batch misreporting will take longer to reach steady-state at the target load. This delayed effect of misreporting also scales up as the target (and therefore batch size) increases. Regardless, with a longer attack, the adversary would be able to reduce the error rate, although our evaluated attack window proves to be effective still.

The misreporting success showed a similar pattern across all of the algorithms, where none proved to be more or less resistant to misreporting. The potency of misreports also aligned with that observed under the short flows scenario.

4.3 Effects on Network Performance

As flow throughputs can be significantly larger in a long-flow environment (average throughputs generally >5Mbps [21]), for example when streaming media, utilization on the switch port increases significantly relative to the available capacity (typically 100Mb [11]). Higher utilizations begin to impose non-negligible delay overheads for active flows. Therefore, the adversary can directly control the congestion at the server connected by the switch port. Note that the goal is not to cause denial-of-service at the switch itself.

As the imposed delay increases with a power-law (under an M/D/1 system) as utilization increases, the throughputs for flows traversing the port thus decrease similarly as the adversary draws more load toward the compromised switch. We configure a network with a larger arrival rate of 10K new flows per second and show in Figure 4 (left) how the average throughput for flows changes as the adversary's target utilization increases and more traffic is shifted onto the compromised switch. We also plot the predicted changes according to the proposed queuing model, demonstrating that the model is a reasonable approximator of the shape and scale of the plot from the experimental results.

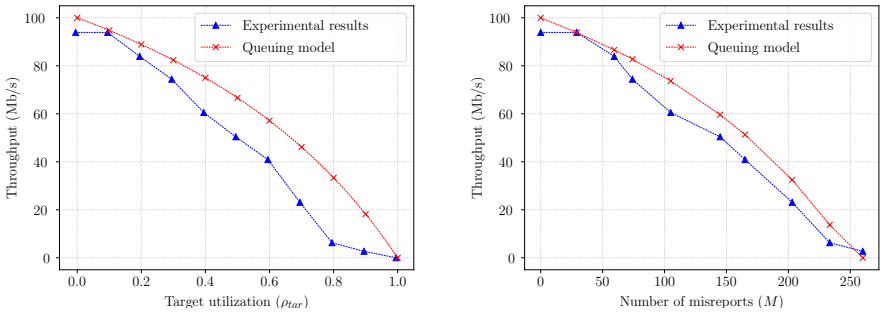

Fig. 4. Throughput vs. target utilization and number of misreports for least-loaded.

The results demonstrate that throughput loss becomes significant quickly. At 20% utilization, flows suffer a nearly 20% throughput loss; similarly, at 40% utilization, which is not uncommon in modern cloud and datacenter networks [10], nearly 40% of throughput is lost. Although, the extent to which an adversary can degrade the throughput for active flows depends heavily on the number and arrival rate of flows in the network. Nonetheless, the ability of the adversary to impose throughput losses on flows in the network exists. In Figure 4 (right), we measure the throughput changes as a function of the number of misreports sent. Note that pool members observed an average utilization of 10% under normal conditions in this scenario, therefore a goal of either 0% or 10% utilization resulted in approximately the same throughput although a single misreport was sent once every t_{long} seconds when the goal is 10% utilization.

Throughput loss is related to utilization. For example, the adversary misreports in batches of 2 consecutive misreports at the beginning of every 10-second time window (t_{long}), for a total of about 60 misreports over the 300-second attack window to draw nearly 20% utilization and induce a 20% throughput loss. As the steady-state load is already significantly high (about 10% utilization), misreporting will indeed draw more and heavier flows through the switch, meaning that the throughput loss per misreport is significant. In contrast, in networks with a lower flow inter-arrival rate and a smaller number of active flows in the network, the throughput loss per misreport is significantly less, and thus the adversary would have to misreport significantly more to cause a similar effect.

The key insight from Figure 4 is that the proposed attack model (with the approximated $\overline{L}$) accurately describes the network behavior as a function of misreporting, to within 12% and 7% of the predicted behavior, respectively. In the least-connections case in Figure 5, the model was within 10% and 5% error, respectively. In this way, we show that the attack model is an effective tool for performing reconnaissance and planning attacks on real networks, besides providing a means for further analysis defensively.

4.4 Discussion

The key insight from our experimentation is that an adversary can in fact feasibly subvert the load balancer by sending false load values in the load reports collected

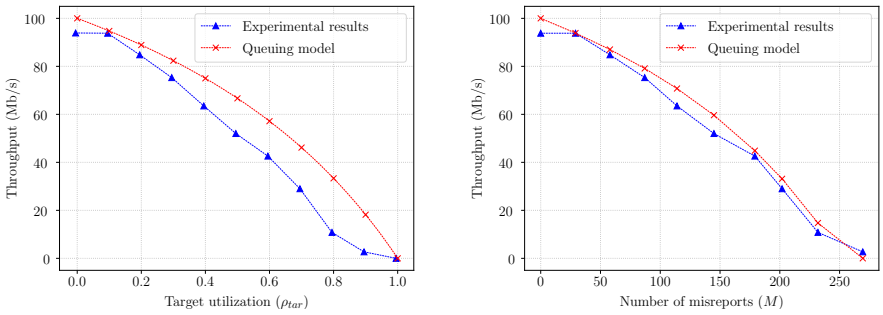

Fig. 5. Throughput vs. target utilization and number of misreports for least-connections.

by the load balancer. Note that the chosen heuristics (for computing the number of misreports) are not necessarily optimal, and a persistent threat can tune attack parameters dynamically to meet their goals. However, the chosen heuristics still prove to be effective as a first exploration into misreporting attacks in general.

We also found that the analytical model accurately reflects what we observed experimentally, which makes the model an effective tool for both planning attacks and defensive analysis (without having to test the attack in a real network). Further, the need for a general security framework becomes obvious. Recent advances in SDN-based anomaly detection have tried to address this problem, however, the approaches are not designed to detect this attack [31, 28, 17, 46]. The state-of-the-art detection system Sphinx [17] relies on trusting edge switches to detect inconsistencies along flow paths, and it operates only at flow-level where load balancing is typically done at port-level (and thus inconsistencies cannot be traced along specific paths due to many flows combining and splitting at switch ports). Other defenses are designed to thwart specific attacks, namely: DDoS, link-flooding, or topology poisoning attacks [26]. Additionally, the systems have design constraints (e.g., monitoring only *hosts* for malicious behavior) that make them not applicable to the proposed misreporting attacks.

As discussed in Appendix C, flexibility in parameter choice is limited for the adversary, as small changes lead to a less-effective attack. Therefore, adversarial strategies are constrained to nearly-static behavior for a fixed target load, which serves as a starting point for identifying misreporting attacks. Another potential avenue is leveraging switch neighbors to vet the accuracy of reports.

5 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. In this paper, we proposed a new model and methods for attacking SDN-based load balancers. Our analytical model very accurately described the network conditions as a function of different attack parameters, providing both a means of planning attacks for the adversary, as well as a tool for analysis defensively.

A Model Parameters

Table 2. Model parameters.

Parameter	Description
ρ_{tar}	Target utilization on switch port
λ_{tar}	Target load (e.g., Kb/s) on switch port
W	Attack length (in epochs)
$\overline{L}$	Average load
M	Number of misreports
T_m	Misreporting period (short flows)
P	Percentile for detecting load spikes
B	Misreport batch size (long flows)
t_{long}	Average flow duration (long flows)
U	Misreporting load threshold
δ	Chosen misreported load
N	Number of pool members
R	Flow arrival rate
R_t^i	Load report value sent

B Reconnaissance Algorithm

Algorithm 1 RecordLoadSpikes ():
Inputs: $current\_load, observed\_loads, load\_spike\_detection\_over, S, \overline{L}, P$
Outputs: n/a

```
1: if load_spike_detection_over then
2:     return
3: end if
4: if S.size() < D then
5:     sort(observed_loads)
6:     s ← S.get(P * observed_loads.size())
7:     if current_load ≥ s then
8:         S.add(current_load)
9:     end if
10: else if S.size() == D then
11:     L̄ ← mean(S)
12:     T_m ← L̄/λ_tar
13:     load_spike_detection_over ← true
14: end if
15: return
```

C Experimental Accuracy and Parameter Sensitivity

Experimental accuracy. The experimental results aligned very closely to what was predicted by the attack model (with the approximations for $\overline{L}$ based on the traffic type). Although the throughput in the virtual network seemed to decrease slightly faster than the queuing model predicted, the shape remains the same and further analysis in a larger system (perhaps, with hardware SDN switches) may close that gap. The number of misreports sent during the attack was nearly exactly what was predicted by the model with the approximated $\overline{L}$, to within less than 7% error for either algorithm. This is important for either the adversary or the defender, as the accuracy of the model makes it an effective tool for both planning attacks and defensive analysis (without having to test the attack in a real network).

The fact that misreporting attacks from just a single switch are effective to within a 12% bound of the target load raises important concerns regarding multiple colluding switches. The attack model would still apply in the case of multiple switches in the load balancing pool misreporting. In fact, this scenario would be even more favorable, as any single switch would not have to misreport as much to draw the same target load (in aggregate). Importantly, we note here that the attack is independent of the controller platform and depends only on the load balancing algorithm and network traffic patterns. Although here we distinguish between two distinct traffic patterns found in real networks, in future work we will investigate mixed traffic models (where a pool may serve broad/heterogeneous services) and the limits to both attacks under such conditions.

The other facet of this problem is how effective an over-reporting attack could be in denying service to endpoints connected by the adversary, i.e., the endpoint would never be selected by the load balancer. Similarly, if over-reporting combined with other attacks like spoofing can be used to deny service to arbitrary portions of the network. Future work could also investigate over-reporting to prevent the adversary from receiving new flows at epochs before and after batches or during the misreporting period. As the adversary may naturally receive new flows without misreporting (if the reported load is already low), these flows may skew the estimate for $\overline{L}$, although this did not prove to be problematic in our evaluation.

There are some limitations of the work which warrant some additional experimentation. Certainly, there are any number of ways an adversary could approach a misreporting attack, and here we just explored two variants which can deny service to other pool members and exploit a steady-state analysis to guide the load rate toward some target. The main purpose of this work was to demonstrate the feasibility of mounting such an attack under real network conditions and quantifying how accurate and effective it could be. Specialized load balancing variants of the evaluated algorithms may introduce difficulty for the attack in some cases, for example, by keeping counters on the number of flows scheduled through a switch for least-connections balancing. Additionally, the granularity of the misreporting period and batch size is limited to discrete epochs (e.g., the

attacker cannot misreport every 1.3 seconds). Therefore, close by target loads may have T_m or B rounded to similar values.

As with the under-reporting attack, identifying to what extent an adversary can control network services—like load balancers—by manipulating control information is a large concern that has not been thoroughly investigated. Prior work has discussed methods for an adversary to compromise an SDN switch [15, 6, 41, 12]. Mounting a misreporting attack can be done by simply swapping out the OpenFlow-agent binary with a modified one on the switch. Thus, the need for a general security framework becomes obvious. Recent advances in SDN-based anomaly detection have tried to address this problem, however, we discover that these solutions are not robust enough to detect this attack in practice [31, 28, 17, 46]. Most defenses are designed to thwart specific attacks, namely, DDoS, link-flooding, or topology poisoning attacks. Additionally, these systems have design constraints (e.g., only able to detect *host-based* adversaries) that make them not applicable to the proposed misreporting attacks.

Parameter sensitivity. With respect to the parameters introduced in Table 2 (see Appendix A), the attack may be negatively affected in a few ways for either long or short flows. In terms of the stealthy attack against short flows, a lower P may cause certain lower load values to be considered load spikes, skewing the average load spike amplitude estimate downward and therefore increasing the number of misreports unnecessarily. For long flows, computing the average load before the system reaches a steady state may also increase or decrease the misreporting batch size to a sub-optimal choice that leads to larger error rates with respect to the target load. Lastly, for either traffic type, higher choices of U will cause the adversary to misreport to within a large range of values. The success rate decreases with a power-law relationship to the threshold U (i.e., the probability of all other switches reporting higher than the adversary is $\approx (1-U)^{(N-1)}$). Hence, misreporting within a larger range requires a significantly higher misreporting rate for the same target load, and therefore for an effective attack the flexibility in parameter choice is limited. Experiments with different pool sizes and longer collection periods yielded qualitatively similar results.

With respect to the traffic distribution, here we considered two distinct types of traffic that reflects what is observed in real networks. The assumption is that switches connected to similar downstream services will observe similar patterns. In a mixed-traffic model, although misreporting is still effective, the accuracy of M is affected by the ratio of short and long network flows, which means that the drawn amount of load may have a larger margin from the target (e.g., a 20% error). However, as the ratio of traffic shifts toward more short flows or more long flows (the more realistic case [10, 40]), then the drawn load will more closely approach the goal.

Acknowledgements

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. The netfilter.org project (1998), https://www.netfilter.org/
2. Project floodlight. http://www.projectfloodlight.org/floodlight/ (2011), [Online; accessed 19-October-2018]
3. Opendaylight project. https://www.opendaylight.org/ (2013), [Online; accessed 19-October-2018]
4. Openflow switch specification. https://www.opennetworking.org/software-defined-standards/specifications/ (2015), [Online; accessed 19-October-2018]
5. Araújo, J.T., Saino, L., Buytenhek, L., Landa, R.: Balancing on the edge: Transport affinity without network state. In: 15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18). pp. 111–124 (2018)
6. Arbettu, R.K., Khondoker, R., Bayarou, K., Weber, F.: Security analysis of opendaylight, onos, rosemary and ryu sdn controllers. In: 2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks). pp. 37–44 (Sept 2016). https://doi.org/10.1109/NETWKS.2016.7751150
7. Aslam, S., Shah, M.A.: Load balancing algorithms in cloud computing: A survey of modern techniques. In: 2015 National Software Engineering Conference (NSEC). pp. 30–35. IEEE (2015)
8. Aslan, M., Matrawy, A.: On the impact of network state collection on the performance of sdn applications. IEEE Communications Letters **20**(1), 5–8 (2016)
9. Aweya, J., Ouellette, M., Montuno, D.Y., Doray, B., Felske, K.: An adaptive load balancing scheme for web servers. International Journal of Network Management **12**(1), 3–39 (2002)
10. Benson, T., Akella, A., Maltz, D.A.: Network traffic characteristics of data centers in the wild. In: Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. pp. 267–280. ACM (2010)
11. Benson, T., Anand, A., Akella, A., Zhang, M.: Understanding data center traffic characteristics. In: Proceedings of the 1st ACM workshop on Research on enterprise networking. pp. 65–72. ACM (2009)
12. Benzekki, K., El Fergougui, A., Elbelrhiti Elalaoui, A.: Software-defined networking (sdn): a survey. Security and communication networks **9**(18), 5803–5833 (2016)
13. Chandrasekaran, S.S.: Understanding traffic characteristics in a server to server data center network (2017)
14. Curtis, A.R., Mogul, J.C., Tourrilhes, J., Yalagandula, P., Sharma, P., Banerjee, S.: Devoflow: Scaling flow management for high-performance networks. In: ACM SIGCOMM Computer Communication Review. vol. 41, pp. 254–265. ACM (2011)
15. Dargahi, T., Caponi, A., Ambrosin, M., Bianchi, G., Conti, M.: A survey on the security of stateful sdn data planes. IEEE Communications Surveys & Tutorials **19**(3), 1701–1725 (2017)

16. De Oliveira, R.L.S., Schweitzer, C.M., Shinoda, A.A., Prete, L.R.: Using mininet for emulation and prototyping software-defined networks. In: 2014 IEEE Colombian Conference on Communications and Computing (COLCOM). pp. 1–6. IEEE (2014)
17. Dhawan, M., Poddar, R., Mahajan, K., Mann, V.: Sphinx: Detecting security attacks in software-defined networks. (2015)
18. Dridi, L., Zhani, M.F.: Sdn-guard: Dos attacks mitigation in sdn networks. In: 2016 5th IEEE International Conference on Cloud Networking (Cloudnet). pp. 212–217. IEEE (2016)
19. Eisenbud, D.E., Yi, C., Contavalli, C., Smith, C., Kononov, R., Mann-Hielscher, E., Cilingiroglu, A., Cheyney, B., Shang, W., Hosein, J.D.: Maglev: A fast and reliable software network load balancer. In: 13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16). pp. 523–535 (2016)
20. Feghhi, S., Leith, D.J.: A web traffic analysis attack using only timing information. IEEE Transactions on Information Forensics and Security **11**(8), 1747–1759 (2016)
21. Fowler, S., Sarfraz, J., Abbas, M.M., Bergfeldt, E., Angelakis, V.: Evaluation and prospects from a measurement campaign on real multimedia traffic in lte vs. umts. In: 2014 4th International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace Electronic Systems (VITAE). pp. 1–5 (May 2014). https://doi.org/10.1109/VITAE.2014.6934475
22. Greenberg, A., Hamilton, J.R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D.A., Patel, P., Sengupta, S.: Vl2: a scalable and flexible data center network. In: ACM SIGCOMM computer communication review. vol. 39, pp. 51–62. ACM (2009)
23. Guirguis, M., Bestavros, A., Matta, I., Zhang, Y.: Reduction of quality (roq) attacks on dynamic load balancers: Vulnerability assessment and design tradeoffs. In: IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications. pp. 857–865. IEEE (2007)
24. Guo, Z., Su, M., Xu, Y., Duan, Z., Wang, L., Hui, S., Chao, H.J.: Improving the performance of load balancing in software-defined networks through load variance-based synchronization. Computer Networks **68**, 95–109 (2014)
25. Handigol, N., Seetharaman, S.: Plug-n-serve: Load-balancing web traffic using openflow (2009)
26. Hong, S., Xu, L., Wang, H., Gu, G.: Poisoning network visibility in software-defined networks: New attacks and countermeasures. (2015)
27. Kandula, S., Sengupta, S., Greenberg, A., Patel, P., Chaiken, R.: The nature of data center traffic: measurements & analysis. In: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement. pp. 202–208. ACM (2009)
28. Kang, M.S., Gligor, V.D., Sekar, V., et al.: Spiffy: Inducing cost-detectability tradeoffs for persistent link-flooding attacks. (2016)
29. Kang, N., Ghobadi, M., Reumann, J., Shraer, A., Rexford, J.: Niagara: Scalable load balancing on commodity switches. Tech. rep., Technical Report (TR-973-14), Princeton (2014)
30. Khan, S., Gani, A., Wahab, A.W.A., Guizani, M., Khan, M.K.: Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art. IEEE Communications Surveys & Tutorials **19**(1), 303–324 (2016)
31. Lee, S., Kim, J., Shin, S., Porras, P., Yegneswaran, V.: Athena: A framework for scalable anomaly detection in software-defined networks. In: 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 249–260. IEEE (2017)
32. Li, J., Chang, X., Ren, Y., Zhang, Z., Wang, G.: An effective path load balancing mechanism based on sdn. In: 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications. pp. 527–533. IEEE (2014)

33. Mahmood, A., Rashid, I.: Comparison of load balancing algorithms for clustered web servers. In: ICIMU 2011: Proceedings of the 5th international Conference on Information Technology & Multimedia. pp. 1–6. IEEE (2011)
34. Mesbahi, M., Rahmani, A.M.: Load balancing in cloud computing: a state of the art survey (2016)
35. Neghabi, A.A., Jafari Navimipour, N., Hosseinzadeh, M., Rezaee, A.: Load balancing mechanisms in the software defined networks: A systematic and comprehensive review of the literature. IEEE Access **6**, 14159–14178 (2018). https://doi.org/10.1109/ACCESS.2018.2805842
36. Patel, P., Bansal, D., Yuan, L., Murthy, A., Greenberg, A., Maltz, D.A., Kern, R., Kumar, H., Zikos, M., Wu, H., et al.: Ananta: Cloud scale load balancing. In: ACM SIGCOMM Computer Communication Review. vol. 43, pp. 207–218. ACM (2013)
37. Qian, H., Medhi, D.: Server operational cost optimization for cloud computing service providers over a time horizon. In: Hot-ICE (2011)
38. Qilin, M., Weikang, S.: A load balancing method based on sdn. In: 2015 Seventh International Conference on Measuring Technology and Mechatronics Automation. pp. 18–21. IEEE (2015)
39. Raghavan, B., Vishwanath, K., Ramabhadran, S., Yocum, K., Snoeren, A.C.: Cloud control with distributed rate limiting. In: ACM SIGCOMM Computer Communication Review. vol. 37, pp. 337–348. ACM (2007)
40. Rao, A., Legout, A., Lim, Y.s., Towsley, D., Barakat, C., Dabbous, W.: Network characteristics of video streaming traffic. In: Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies. pp. 1–12 (2011)
41. Scott-Hayward, S., O'Callaghan, G., Sezer, S.: Sdn security: A survey. In: 2013 IEEE SDN For Future Networks and Services (SDN4FNS). pp. 1–7. IEEE (2013)
42. Wang, R., Butnariu, D., Rexford, J., et al.: Openflow-based server load balancing gone wild. (2011)
43. Williams, J.: Introducing the github load balancer (Sep 2016), https://githubengineering.com/introducing-glb/
44. Yoon, C., Lee, S., Kang, H., Park, T., Shin, S., Yegneswaran, V., Porras, P., Gu, G.: Flow wars: Systemizing the attack surface and defenses in software-defined networks. IEEE/ACM Transactions on Networking (TON) **25**(6), 3514–3530 (2017)
45. Zhang, J., Yu, F.R., Wang, S., Huang, T., Liu, Z., Liu, Y.: Load balancing in data center networks: A survey. IEEE Communications Surveys & Tutorials **20**(3), 2324–2352
46. Zhang, Y.: An adaptive flow counting method for anomaly detection in sdn. In: Proceedings of the ninth ACM conference on Emerging networking experiments and technologies. pp. 25–30. ACM (2013)
47. Zhou, Y., Zhu, M., Xiao, L., Ruan, L., Duan, W., Li, D., Liu, R., Zhu, M.: 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. pp. 851–856. IEEE (2014)