Stealthy Misreporting Attacks Against Load Balancing

Mingli Yu*, *Student Member, IEEE*, Quinn K Burke†, *Student Member, IEEE*, Tom La Porta*, *Fellow, IEEE*, and Patrick McDaniel†, *Fellow, IEEE, ACM*

***Abstract*—Load balancing in software-defined networks (SDNs) is commonly realized with a centralized architecture. Dynamic load balancing relies on the SDN controller to periodically collect traffic statistics from network switches and make decisions in a timely manner. In this paper, we examine the extent to which an adversary that has compromised a switch can influence the load balancing algorithm by misreporting its own traffic statistics. We design an attack that allows an adversary to perform preliminary reconnaissance, which means learning network traffic distributions and setting attack parameters, and then accurately model and estimate the reward from misreporting while evading detection. Our evaluation offers three insights: (1) network traffic exhibits discernible patterns by reconnaissance; (2) the reconnaissance can be used to design misreporting attacks that can effectively draw unfair proportions of network traffic to the adversary under the guise of honest behavior; and (3) reconnaissance itself can be accelerated by misreporting to launch more targeted attacks.**

***Index Terms*—load balancing, misreport, Software Defined Networking, network security.**

I. Introduction

Software Defined Networking (SDN) has fundamentally changed the way networks are built and maintained. In SDN, the control functions are integrated into a logically centralized controller to separate the data plane and the control plane. This separation enables flexible load balancing [1], routing, service composition and network management. The deployment of SDNs has also given rise to security concerns. Although SDN eases the defense against traditional IP-network attacks such as port scanning and firewall probing [2] by using agile structures and policies [3], it also introduces new vulnerabilities that may be leveraged to attack the network.

We focus on the vulnerabilities of the *load balancer*, which is a key network function in the SDN controller. The load balancer in SDN is typically 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. In this paper we study the least-loaded load balancing algorithm that require the network switches to report traffic statistics to the controller application in discrete time epochs. The controller application then decides how to route incoming flows that arrive during that epoch. If a switch is assigned the flows in an epoch, we say it "wins" the flows or it "wins" the time epoch. Hence, dynamic load balancing requires distributed trust among the switches in reporting accurate traffic loads, which creates an opportunity for an adversary at a compromised switch to misreport traffic loads to influence load balancing.

This vulnerability has been exploited [4] and several defense mechanisms have been put forward to deal with these type of attacks [5], [6]. However, existing studies have not addressed the misreport attacks in the ingress/egress switches, which make up a large proportion of the network. Therefore, we argue that the adversary that takes advantage of this vulnerability and misreports traffic loads can control the amount of the traffic flowing through the compromised edge switch for eavesdropping and traffic analysis.

In this paper we define an attack in which an adversary that has compromised a switch measures the traffic distribution at the switch, and based on this reconnaissance sets parameters to capture a targeted load of traffic.

Using real network traces we show the difficulty in precisely learning the traffic distributions and specifically targeting a percentage of load, but also show such attacks can still be effective. We bound the probability of capturing ("winning") a flow and its traffic depending on misreported traffic values. A tight bound is difficult to achieve because a successful attack biases the load on all switches, but nevertheless show that the bound is useful in helping attackers launch an effective attack.

We also show that reconnaissance allows an adversary to set their attack parameters to allow for a stealthy attack that is still effective in capturing traffic. We then characterize the effectiveness of the attack by showing how an attacker can learn traffic characteristics in a network, such as the existence of a specific flow or the types of flows in a network and the stealthiness of the attack by evaluating the attack against several defense methods.

A. Related Work

SDN vulnerabilities: The design of existing SDN architectures and protocols is performance-driven, leaving many security vulnerabilities. One of the weakest aspects is the inter-dependency between the controller and the switches. This creates a vulnerability such that the attacker may gain control of one or more switches [7] to inject malicious control messages [8] or subvert control applications by sending false reports [4]. Other works [9]–[14] demonstrate that an adversary can compromise the control plane by eavesdropping on control messages and client traffic or executing a man-in-the-middle attack.

The *authors are with the Pennsylvania State University, University Park, PA, USA. Email: {mxy309, tfl12}@psu.edu. The †authors are with the University of Wisconsin-Madison, Madison, WI, USA. Email: {qkb, mcdaniel}@cs.wisc.edu.

We exploit this dependency to perform network reconnaissance, for example, exploiting the network flow distributions, and execute a misreport attack against the load balancer. We build on prior work [15] which models an intelligent adversary that also performs reconnaissance and misreporting. However, unlike [15] which relied on uniform synthetic traffic, we characterize flow distributions with real network traces, develop realistics bounds of captured flows based on attack parameters, and show how attackers can overcome imbalances in flow characteristics and dynamics that emerge in real networks.

SDN defenses: Although the centralized control in SDN eases the defense against traditional network attacks through agile structures and policies [3], it remains open how to design robust defenses against a large threat surface and unknown attacks. There are some state-of-the-art detection systems that have partially addressed the dependency issues between the control plane and data plane. Sphinx [5] builds an auxiliary network topology and data plane forwarding state graph to maintain a global view of the network state. It tackles the misreporting attack by examining the consistency of traffic statistic reports of the switches that share the same flows. However, it fails to detect the misreports from ingress/egress switches for lack of another source of information for verification. Related systems such as Spiffy [16] and OpenWatch [17] specifically target short-term volume-based attacks by malicious hosts but are not appropriate for monitoring noisy datacenter or backbone traffic.

Load balancing security: Existing load balancing solutions can be classified into two categories: static and dynamic. Common static load balancing algorithms include Round-Robin and hash-based solutions such equal-cost multipath (ECMP) [18], Maglev hashing [19] and Beamer [20]. Static solutions implement fixed mappings and thus cannot exploit run-time knowledge of bandwidth utilization, often resulting in under-utilization and increased latency.

Dynamic solutions implement various reactive techniques for connection assignment by maintaining a per-connection state. They allow more flexible decision making by exploiting knowledge of resource utilization. Widely used dynamic load balancing algorithms include least-response-time, least-loaded, and least-connections, along with their weighted counterparts [1], [21], [22]; all of which require the switches to report statistics to the controller. In our work, we focus on the least load selection because it is robust when switches have different processing capabilities which is common with current switches.

B. Summary of Contributions

Compared to prior work [15] with a similar methodology we make the following contributions in this paper:

1) We formally define a misreport attack model with respect to the number of time epochs the malicious switch wins.
2) We develop a lower bound and an upper bound on the probability that the malicious switch wins a time epoch.
3) We verify the similarity of load report distribution among the pool and over time which enables us to evaluate the proposed attack using trace-driven Mininet experiments. We show that a malicious switch can substantially increase the amount of network traffic it captures and characteristics it learns.
4) We mathematically define stealthiness and evaluate the attack against several defense algorithms, which provides empirical evidence that the attack is difficult to detect.

Roadmap. Section II introduces our models and problem formulation. Section III presents our attack model. Section IV evaluates the attack on network traces. Section V discusses defenses. Section VI concludes the paper.

II. Problem Formulation

A. Load Balancer Model

SDN-based load balancers typically follow a centralized model, where the load balancer uses a global view of the real-time traffic to facilitate fine-grained traffic engineering. A simple example of network topology with a centralized load balancing module is shown in Fig. 1[1]; the edge switches or ToR switches in datacenter networks may be connected to server racks or another LAN. This centralized approach to load balancing therefore critically relies on the network switches reporting accurate traffic information to the controller application.

We define a *pool* as a set of switch ports among which the network load is balanced. To simplify analysis, we only consider a static set of pool members.

The load balancer monitors the status of switch ports by periodically collecting traffic load reports from each switch port in the pool at a specified time *epoch* using features supported by the OpenFlow protocol [23]. Assume each time epoch lasts t seconds. The load reports will come in the form of switch statistics, which record all the incoming and outgoing bytes at the switch ports since the last report. We note that load reports are typically collected at port-level (coarse-grained) rather than, for example, flow-level (fine-grained) because port-level statistics capture a better measurement on the activity at the connected end-hosts.

Once the load balancer gathers the statistics via the controller, it determines which pool member is assigned the new incoming flows in the next time epoch. Widely used algorithms like least-loaded-server typically follow a "winner takes all" approach, where the least-loaded-server temporarily receives all inbound traffic until the next epoch. When a winner is selected, the load balancer issues new flow rules to the appropriate switches to establish a path from the source to the selected pool member.

There are several reasons why load reports are generated from the switches and not the aggregator. First, there may be multiple gateways that act as aggregators into a network, so none know the full state of the load on the edge switches. Further, load balancing decisions must be made at service-level granularity, and therefore measurements are often collected at the edge (rather than at aggregation-layer switches) to more easily be able to discern flows targeted toward the specific service. Finally, we note that end hosts do not run OpenFlow agents in general and therefore, the edge switch load is used as a proxy for server load instead [24].

[1]The switch-to-controller connections are hidden in the figure.

The formal definition of the load balancing scheme follows. Consider a network consisting of N pool members. The load balancer polls the switches for their load statistics R_i^k every time epoch, where i refers to the switch index, and k refers to the time epoch index. New flows are directed to the switch that reports the minimum load—in terms of the number of bytes forwarded since the last time epoch (with ties broken arbitrarily). Formally,

$$I = \operatorname*{argmin}_i R_i^k$$

B. Threat Model

We assume a threat model similar to prior works on SDN security [7]. An adversary may therefore compromise a switch through hardware backdoors, weak admin interfaces, or through vulnerabilities in potentially open-source switch softwares [9], [25]–[29]. We assume for simplicity that only a single edge switch in the load balancing pool is compromised; our attacks extend naturally to multiple compromised (and potentially colluding) switches and will only worsen the damage that an attacker can do. We also assume for simplicity that a single egress port on each switch is present in the pool; an attacker who can misreport for multiple ports in a pool (or across different service pools) will similarly be able to cause more damage to a single service (or multiple services). The following primitives are therefore available to the adversary:

- **Primitive 1:** The attacker can fully control the compromised switch and can report any value whenever the load statistics report is requested by the load balancer.
- **Primitive 2:** The attacker can ascertain whether a least load-like algorithm or a least flow-like algorithm is employed based on what information the controller is requesting.
- **Primitive 3:** The attacker can record its own load statistics reports and keep track of the flows routed through it. Apart from that, it has access to adequate memory space where the reports and the flow information may be stored, which does not affect the detectability significantly since it only requires less than 1GB in normal 10Gbps link.
- **Primitive 4:** The attacker can measure the pool size. The attacker may probe and scan the subnet MAC, IP addresses, ports and protocols so that it can accurately reconstruct all the flow rules in flow table and determine the load balancing pool. These capabilities has been shown in prior works [30], [31].

The significance of a successful misreporting attack is threefold. First, it may directly violate a core tenet of security, availability, because an adversary can arbitrarily control how much traffic they receive and thus potentially drop. Second, it also directly allows an adversary to obtain unfair proportions of client traffic sent to the target service endpoint. Prior works have demonstrated that access to large volumes of network traffic has been key means to fingerprinting services, users, and obtaining other private information about clients and who they are communicating with [32], [33]. If an adversary is able to identify certain client behavior patterns, e.g., when they login to a web service, they may execute targeted misreporting

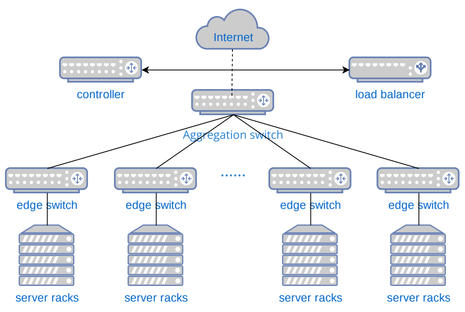


Figure 1. An example local-area network topology representative of, for example, cloud datacenter networks.

attacks at specific times to capture certain client flows and perform further traffic analysis. Last, misreporting can be used by advanced persistent threats (APTs) as a stepping stone in a larger reconnaissance campaign to stage and carry out more sophisticated attacks such as mapping out subnets, clients or other patterns [27], [31], [34], [35].

Since the pool members in our model are typically edge switches which each contain a substantial number of ingress/egress ports, verifying the integrity of load reports is a challenging problem. Although efforts [5], [36] have been made to eliminate this misreporting issue, the overhead for these solutions is not negligible and, most fail to detect misreported ingress/egress port statistics. Traditional cross validation detection fails to detect the misreport because the controller only has access to switch reports and inter-switch link information while some links that connect some hosts to edge switches are invisible to the controller. Our central goal in this work is therefore to quantify the bounds of an attacker's capabilities when misreporting and provide recourse for future defense designs.

The adversary leverages the primitives above to learn the flow and load statistics and under-report to induce the load balancer into sending a target volume of traffic through the compromised switch. This type of attack are independent of the actual internal network topology since the load balancing is based solely on the load reports from edge switches which are vulnerable to modification.

III. Misreport attack model

The misreport attack occurs in two phases: reconnaissance and misreporting. The reconnaissance phase is used by the attacker to learn network statistics so that it can set attack parameters. In the misreporting phase the malicious switch reports its traffic incorrectly according to the parameters set based on reconnaissance. The reconnaissance and parameter setting is important for two reasons. First, if malicious switches are not careful with the parameters and frequency with which they misreport, they may be easily detected by the network administrator [15]. Second, the parameter settings heavily influence the amount of traffic and flows an adversary can capture. Thus the parameters must be set to evade detection and meet the goals of the adversary.

In the following, we first show that network reconnaissance can accurately characterize network flows. We then present the attack model and establish bounds on how successful an attacker can be with specific parameter values.

A. *Preparations: Network Reconnaissance*

This section shows why network reconnaissance is necessary and feasible, and how to do it.

The simplest strategy for misreporting traffic is for an adversary to always report a load value of 0. This trivial attack will capture a large amount of traffic, but is easily detected. A better approach is to have the adversary report a random load value between zero and the true load of the switch. However, our studies of real network flows shown below indicate that real switches almost never experience a load of 0, and so that type of attack may also be easily detected. In addition, because the range of misreport is wide, the effectiveness of the attack may not be high.

Therefore, we put forward a heuristic that determines misreport values based on reconnaissance. A malicious switch collects load reports for a time period, and using these reports approximates the distributions of the future load reports of all the switches. It then launches the misreport attack with the estimated load distribution. By sampling the misreport value from the underlying true load distributions we significantly reduce the risk of being detected. We discuss how to set the parameter values in the next subsection; here we show that network reconnaissance is effective for this purpose.

There are two fundamental characteristics of the load distributions that make the approximation possible: similarity of the load distributions among the pool, and similarity of the load distributions over time.

1) Dataset introduction: In this paper, we leverage the network backbone traffic traces from the samplepoint-F of MAWI [37] to validate the two characteristics. The traces are derived from a 1Gbps transit link of the WIDE network to an upstream ISP. They are collected every day from 2pm and 2:15pm since 2006. Only the 2020 traces are used in this work. Each trace lasts 15mins and contains around 100 million packets. The IP addresses in the traces are anonymized using a prefix-preserving method and the mapping is consistent only within a single trace[2]. We extract the IPv4 TCP and UDP packets for analysis.

2) Similarity of load distributions among the pool: In the context of the MAWI traces, the pool members could be the edge switches that further dispatch the WIDE network traffic among the upstream provider. Since the pool members are generally homogeneous, they may observe similar traffic characteristics(observe means the traffic is routed through the switch) of load distribution with the least load load balancing algorithm [4]. We examine the claim on 100 randomly picked traces from the MAWI dataset. We set the load report interval to be $t = 1$ second and pool size to be $N = 10, 50, 100$, which are common edge switch pool sizes [38]. The load report cumulative probability functions of two different traces are shown in Fig. 2 and Fig. 3. Surprisingly, there are roughly two dominant patterns. In some traces like Fig. 2, it is hard to distinguish the load distributions of pool members. Nevertheless, in some traces like Fig. 3, the pattern deviates from the norm, where one or two members receive a higher load than the others while others have nearly identical load distributions. These two dominant patterns persist when the pool size is set to any number from 10 to 1000. We observe similar results in a Chicago to New York backbone link [39].

These results show that for the most part, the load distribution within a pool is nearly identical with only a few outliers. The reason for the outliers is that there are some flows that are heavy and long lasting, and if a switch receives this traffic, it will tend to have a higher load than all other switches for the duration of the flow. In summary, the uneven distribution of load across flows can cause the outliers.

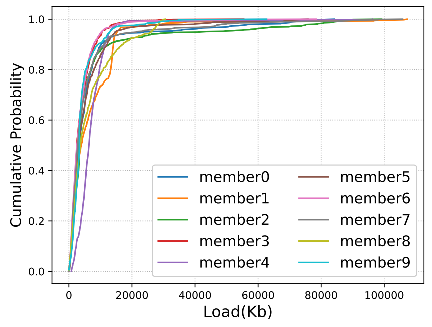


Figure 2. Pattern 1: distribution of load reports collected every second from a 10-member load balancing pool.

img/n50loadreport5-eps-converted-to.pdf

Figure 3. Pattern 2: distribution of load reports collected every second from a 50-member load balancing pool.

3) Similarity of load distributions over time: In addition to similarity of flows within a pool, there is also similarity of load distributions over time—specifically, over a sufficiently short period of time. MAWI dataset offers several day-long traces of samplepoint-F. Among them we randomly pick 30 30-min-long traces from 2020 and draw the load distributions of the pool members of the first 15 mins and the second 15 mins separately. The pool size and the load report interval are the same as those in the previous section. The result in Fig. 4

[2]We don't have any further information about the traffic locality since the prefix preservation length is unknown.

confirms the similarity exists between the first and second time intervals.

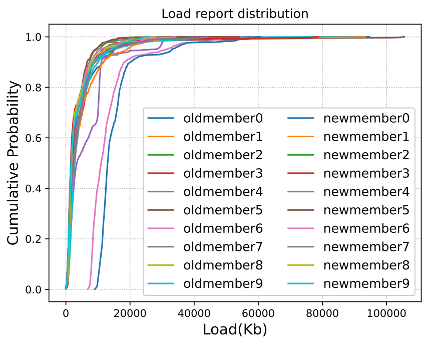


Figure 4. Distribution of load reports collected per second from switches in a 10-member load balancing pool over two consecutive periods (old refers to the first 15 mins, while new refers to the second 15 mins).

Takeaway: The load distributions of the pool members are similar. So are those of two consecutive time periods. This enables a single switch to perform reconnaissance and load distribution approximation, which in turn will allow the adversary to misreport while evading detection.

B. Attack Model in Terms of Win Rate

In the second phase the malicious switch misreports a load value at each time epoch, leveraging the load report and other statistics collected in the first phase. Various choices of misreport parameters like the misreported load amount lead to different consequences. In this section we build the misreport attack model by determining the probability that a malicious switch receives (wins) the new traffic arriving in a time epoch under the least load balancing algorithm.

Suppose the target is to win $X\%$ of the time epochs. Specifically, we build a probabilistic model of the misreport success by analyzing how the misreport frequency M, and the misreported amount L (unit:Kb) at each time epoch affect X. These two parameters define when to misreport and what to misreport. Here we assume the attack has the same effects when misreporting every $1/M$ second or misreporting every second with the probability M; we discuss the differences in these assumptions in Section IV-C.

There are two cases where the malicious switch may win a time epoch: misreport and honest report, which are exclusive. When reporting honestly in a pool size N, we assume every switch will win, on average, $1/N$ time epochs. The challenge is to determine the probability of winning when misreporting. This probability solely depends on L. So we classify the two attack methods by L.

- **Trivial Attack:** In this case $L = 0$ at each time epoch of the misreport. The goal of the adversary is to maximize the winning probability under a given target. While this will result in a very large winning percentage, it is easily detectable. We include this as a point of comparison.
- **Stealthy Attack:** Recall that we approximate the future load distribution of all the pool members by the load distribution of the adversary in the first phase. In this attack, the adversary samples the load L from the prior distributions and determines the minimum load experienced by a switch during the reconnaissance period. Here we introduce another pair of parameters of sampling: p, a cumulative probability bar of the load distribution ranging from 0 to 1 and L_p, the load value at the corresponding p. The bottom p fraction of load reports falls into the interval between the minimum report load and L_p.
For example, Fig. 5 shows how the adversary determines L by the learned load distribution and load minimum shown in Fig. 3. If the cumulative probability bar $p = 0.6$ (y-axis), we find that the corresponding L_p (x-axis) is around 3500Kb for most pool members. So $L_p \approx 3500Kb$. With p and L_p set, the adversary will randomly select a misreported load value L such that $L \in$ [history load min, min(true load amount, L_p)], shown as the horizontal bar in Fig. 5. We call this a stealthy attack because it reuses the history load report and is harder to detect. Generally speaking, the stealthy attack does not report a zero load. The minimum load values are $50 \sim 200\ Kb/s$ in the traces. The smaller p is, the more aggressive the stealthy attack is.

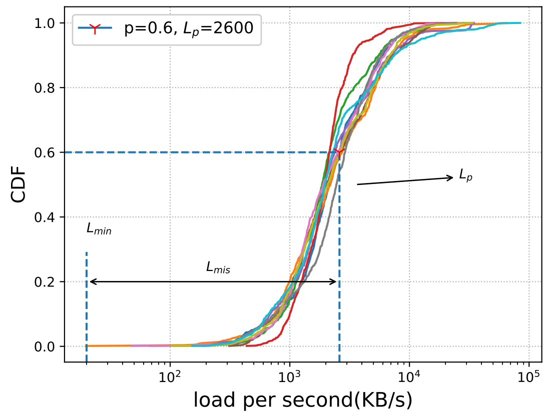


Figure 5. Determine misreported amount from the learned load distribution and load minimum

C. Win Rate Bounds

With the parameters defined, we give a bound on the target win rate X in this section. This allows an attacker to pick parameter values based on their target win rate.

Theorem 1. *Let the adversary adopt the stealthy attack method with parameters p and L_p. Denote P as the probability that the malicious switch wins the time epoch. Assume the misreport attack lasts for a sufficiently short time so that the load distributions between pool members remain similar*[3].

[3]This matters because long attacks will reshape the load distributions of the malicious switch and other honest switches, enlarging the KL divergence between them and thus degrading the accuracy of the characteristics learned during reconnaissance.

Then

$$M\sum_{i=0}^{N-1}\binom{N-1}{i}\frac{1}{i+1}(1-p)^{N-1-i}p^i \le \mathbb{E}(P) \le M+\frac{1-M}{N}. \tag{1}$$

Proof. The winning probability is composed of two exclusive cases: honest report and misreport. Denote the probability of winning by misreport as P_1 and that of winning by honest report as P_2. Formally, $P = P_1 + P_2$. Then

$$P_1 \le P = P_1 + P_2 \tag{2}$$

There are two cases in which the malicious switch can win by misreporting. In the first case, the malicious switch is the only switch that misreports the minimum load value so it wins outright. In the second case at least one other switch reports the same minimum load value and the malicious switch is randomly selected. Denote the probability of the two cases as P_3 and P_4 by sequence. Then $P_1 = P_3 + P_4$.

$$P_3 = \Pr[\bigcap_{i\neq N}(R_i^t > R_{S_1}^t)] \ge M(1-p)^{N-1} \tag{3}$$

$$P_4 = M\sum_{i=1}^{N-1}\binom{N-1}{i}\frac{1}{i+1}(1-p)^{N-1-i}p^i \tag{4}$$

Note that $1-p$ is the probability that the the load report of any other switch is larger than L_p. On the other hand, since $P(A\cap B)\le P(A)$ we have

$$P_1 = \Pr\{\textit{misreport and win}\} \le M \tag{5}$$

Assume the malicious switch receives its fair share of the traffic when it reports honestly, that is, $\mathbb{E}(P_2) = \frac{1-M}{N}$. However, the higher average load brought by the misreport diminishes the chances of winning by honest report because the load in the honest switches is suppressed when the malicious node receives extra traffic. Therefore

$$\mathbb{E}(P_2) \le \frac{1-M}{N} \tag{6}$$

By combining (2), (3), (4), (5), (6) we can obtain the result. □

It seems that Thm. 1 may only give a loose upper bound under ideal circumstances since the upper bound is unrelated to p. However, it's impossible to get a tighter bound in fact.

Theorem 2. *$M + \frac{1-M}{N}$ is the least upper bound of $\mathbb{E}(P)$.*

Proof. Prove by a counter example. Assume there exists a fixed constant $\epsilon > 0$ such that $\mathbb{E}(P) \le M + \frac{1-M}{N} - \epsilon$. Suppose $t = 1$s and the duration of all the flows except N flows is less than 1s. These N flows are identical and last for a long time. Each of the N flows is assigned to an individual pool member ahead of time and each has a non-zero load every second. Then the bonus load or flows acquired by the misreport attack would vanish in 1s. This shows the misreport attack has no effect on the honest report stage. So $\mathbb{E}(P_2) = \frac{1-M}{N}$. Since every pool member has non-zero load due to long-lived flows, the malicious switch may win every time it misreports if it misreports 0. Then $\mathbb{E}(P_1) = M$ and $\mathbb{E}(P) = M + \frac{1-M}{N}$, giving a contradiction. □

As a matter of fact, a tighter bound does not exist unless the load distributions between pool members are no longer similar after some time and the byte distribution over time follows some pattern. As the misreport attack evolves, an unfair load share would gradually move to the malicious switch, breaking the initial assumption that the load distributions between pool members are nearly the same. In that case p would be a biased estimation of the actual percentile. Furthermore, the byte distribution over time needs to follow some pattern so that we can estimate the bias of p and measure the influence the misreport has on the honest report stage, that is, P_2. Unfortunately, the traffic byte distribution does not follow any short-range distribution or common distribution such as uniform distribution [40]. Therefore, it's hard to give a tighter bound.

In practice, the honest win term in the upper bound (P_2) may be close to 0 if the malicious switch is highly successful when it misreports because when reporting honestly its load will be high. In this case we can approximate P with P_1 as shown in Sec IV-B3.

Although Thm. 1 has nothing to do with the reconnaissance intuitively, it is the reconnaissance phase that allows the attacker to sample from the load report history and thus determine p. Now the adversary can set its target X, decide the appropriate attack approach, compute the required misreport frequency from Eq. (1) if necessary, and begin misreporting. We can even obtain the actual number of misreports: the product of $M \times W$, the time window of the attack.

Instead of targeting the win rate, a previous study [15] targets the load directly. It bridges the gap between the win rate and load by the assumption that an adversary will obtain $X\%$ of the total flows and load in the system at steady state if the adversary wins $X\%$ of the time epochs. They calculate the expected load as:

$$\mathbb{E}(load) = \frac{1-M}{N} + M(1-p)^{N-1}$$

However, this claim doesn't hold as shown in Sec IV-B. Consequently, we formalize the misreport attack in terms of a target fraction of time epochs where the compromised switch wins in our model.

IV. Evaluation

In this section we evaluate the proposed misreport attack strategies via trace-driven simulations using backbone traffic traces from the samplepoint-F of MAWI.

A. Simulation Setting and Evaluation Metrics

Experiments are carried out on a virtual SDN created in Mininet [41], running OpenFlow 1.5, Open vSwitch 2.14.0, and a Ryu controller [42]. The load balancer resides in the Ryu controller. We run the Mininet SDN emulator and the controller in a network with the topology shown in Fig. 1[4]. The connections from every switch to the controller are hidden.

In these experiments, we treat a flow which appeared before as a new flow if it has been idle for more than 10s as is common in SDNs. As we mention in Section III-A, we generate the traffic according to traces from MAWI. The load report collection interval $t = 1$ second and the pool size $N = 10$ according to [43], [44]. Different flows are created by varying the source/destination port numbers. New flows originate from the aggregation switch which represents a common gateway from which flows split paths in the network. The attacks are carried out by designating one edge switch as the adversary. Note that our experiments with larger pool sizes yield qualitatively similar results where the traffic load is scaled proportionately.

To evaluate the misreport attack against load balancers we use several metrics. First, we measure the direct impact of misreport, i.e. how much load or how many flows are routed to the malicious switch compared with the baseline where the switch reports honestly. Then we evaluate the bound formulation Eq. (1) under both the trivial and stealthy attacks to understand to what extent an adversary can cause imbalance in the load balancing pool and compare the accuracy of our model with others' [15]. Next, we measure how fast a malicious switch can learn about network traffic patterns while misreporting.

By analyzing the bonus load or flows brought by misreporting at a fine-grained level, the adversary is able to launch more effective attacks against specific targets. We measure the learning speed of the malicious switch in terms of two aspects:

- The increase in observations of specific target flows defined by an IP source and destination pair by the malicious switch compared to honest switches
- The increase in the number of individual IP source and destination pairs the malicious switch observes compared to honest switches

The first metric quantifies how misreporting helps capture more target communications and the second characterizes the capability of a malicious switch to learn more about the communication patterns in the network [45].

B. Ability to Capture Flows and Load

In this section we evaluate the capability of the adversary to capture flows and load, the accuracy of the misreport attack model and the efficiency of the misreport attack.

1) Parameters setting: We randomly pick 16 daily traces from MAWI in this group of experiments. We set $p = 0.01$ for the stealthy attack, which boosts the win rate lower bound in Thm 1 to $0.92M$. If $p = 0.1$, the lower bound is $0.41M$, indicating that more than half of the misreports would fail in the worst case[5]. We set the attack window $W = 300$s, which means during the first 600 seconds the malicious switch reports honestly and records its own load report, and then determines L_p. The misreport attack is launched in the last 300 seconds. In this example, we set the target to be 30% win rate. The trivial misreport frequency is set to $M_1 = 22\%$ and the stealthy misreport frequency $M_2 = 25\%$. These two frequency values are derived from Thm 1 where both methods are expected to achieve 30% win rate at most[6].

2) Capturing flows and load: Fig. 6 and Fig. 7 show the percentage of the flows and load captured by the malicious switch in the last 5mins of each trace, respectively. The traces are sorted by the flow number or load percentage of honest reporting. As we can see, the compromised switch receives 23.3% of flows and 23.9% of load with the trivial attack and 17.8% of flows and 20.5% of load with the stealthy attack while it only receives 8.2% of flows and 16.0% of load when reporting honestly. In fact, the trivial attack effectively draws more flows or load in every trace. The stealthy attack is less effective. In 5 out of the 16 traces the stealthy attack does not give rise to any increase in flow or load compared to the honest case, however it never performs worse than honest reporting. Neither attack achieves the given target. This is because we implement the misreport attack by generating a random number $r \in (0, 1)$ each time and misreport if $r < M_1$ or $r < M_2$. Therefore, the misreport rate does not always equal the fixed M_1 or M_2 and the attacks do not reach the upper bound of the target.

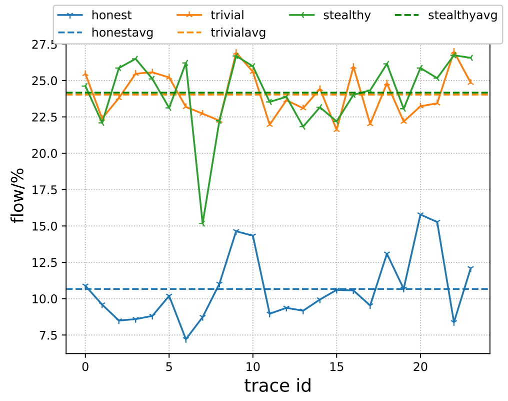


Figure 6. Flows captured by misreport in daily traces

3) Misreport attack model accuracy: We evaluate our model accuracy in terms of win rate and compare it with the model of [15] in this section. Fig. 8 and Fig. 9 give us insight into Fig. 6 and Fig. 7 by showing the win rates (i.e. the number of winning time epochs of a single switch compared to the number of total time epochs), which allows comparison with [15]. The average win rate of an honest switch is close to 10%, which agrees with the fairness assumption. However, the win rate percentage doesn't agree with the load percentage if we look at the trivial/stealthy win rate and trivial/stealthy win load, which contradicts the assumption in [15]. As we show later, this happens because of the uneven load distribution over time and the irregular distribution of winning time epochs of a single switch.

[4]The topology is representative. For example, the aggregation switch in the figure can be any aggregation switch in a fat-tree topology and our goal is to balance the traffic load among the edge switches in the pod.

[5]This numerical analysis implies again that for an effective and efficient attack, the flexibility in parameter choice is limited.

[6]The trivial attack can be viewed as the special stealthy attack where $p = 0$.

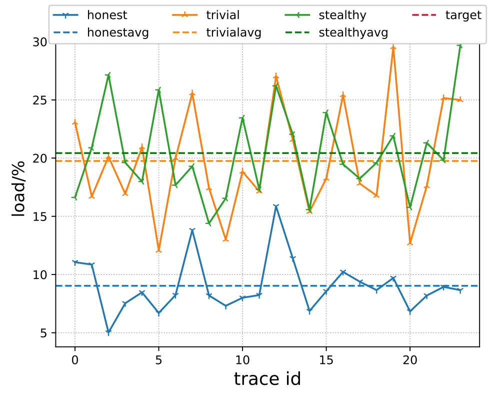


Figure 7. Load captured by misreport in daily traces

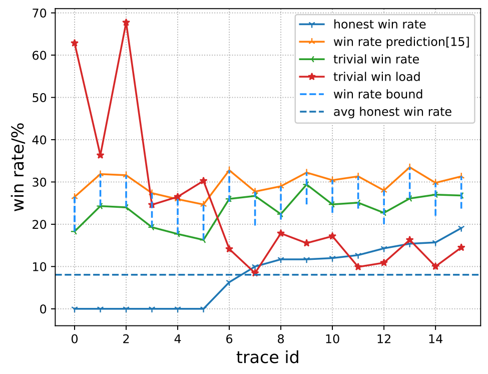


Figure 8. Win rate with trivial misreport attack

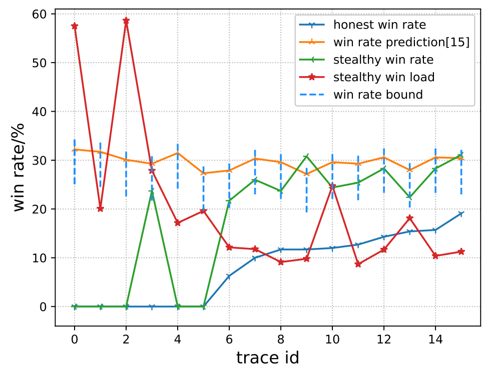


Figure 9. Win rate with stealthy misreport attack

We also notice that the win rate of an honest switch is exceedingly low (e.g., 0.0%) in some special cases such as in traces 0, 1 and 2. We reveal why this happens in Sec IV-B4. As for the bound theorem, all the trivial attack win rates fall within our bounds of prediction while [15] always overestimates. On the other hand, the stealthy attack win rates deviate from our bounds in $1/3$ of the traces but our model is still more accurate than [15]. Table I shows more details of three representative traces; the two anomalies are marked in dark, which we will explain in Sec IV-B5.

Specifically, we can approximate the win rate expectation with the lower bound when p is sufficiently small as in the trivial attack ($p = 0$) or when $p = 0.01$ as we use in the stealthy attack. The malicious switch almost always wins the epochs in which it misreports and almost never wins when reporting honestly because of the high average load brought by misreporting. As shown in Figs. 8 and 9, the trivial win rate tracks the lower bound and the stealthy win rate is sometimes below the lower bound for this reason.

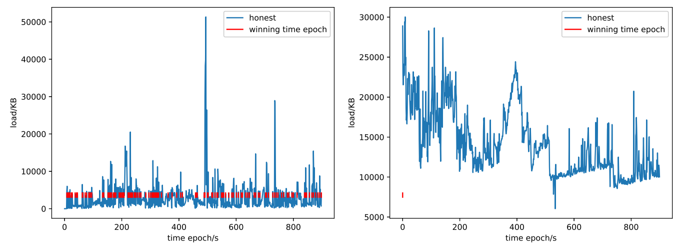
Trace id	0	9	6
pool average load(KB/s)	2842	2720	5251
honest report flow no.	0.3%	11.7%	6.3%
honest report load	57.5%	7.2%	5.2%
trivial attack load	62.9%	15.5%	14.2%
stealthy attack load	57.5%	9.8%	12.1%
honest report win rate	0.0%	11.7%	6.3%
trivial misreport rate	18.3%	24.7%	25.3%
trivial win rate	18.3%	29.4%	26.0%
trivial win rate bound	18.3% ∼ 26.5%	24.7% ∼ 32.3%	25.3% ∼ 32.8%
stealthy misreport rate	27.3%	21.1%	22.0%
stealthy win rate	0.0%	30.8%	21.7%
stealthy win rate bound	25.1% ∼ 34.6%	19.3% ∼ 29.0%	20.2% ∼ 29.8%

Table I
WIN RATE PREDICTION(PARTLY)

4) *Distribution of winning epochs over time:* In this section we explain why the assumption that winning $X\%$ of the time guarantees winning $X\%$ of load does not hold and why it is difficult to obtain an accurate estimation of win rate or load.

First, we look at the distribution of winning epochs over time when the malicious switch reports honestly or misreports. This sheds light on the change of load distribution over time with/out the misreport attack from the time perspective. Fig. 10 show the load distribution over time of an honest switch and the time epochs when the switch wins on two traces. The switch wins short and bursty flows most of the time in Fig. 10 (a) while the switch wins a single elephant flow at some early epoch in Fig. 10 (b). In the latter case the switch does not win any more epochs because the elephant flow persists and keeps the switch load at a high level. The fluctuation of load stems from the dynamic elephant flow. In the former case, most spikes result from winning one time epoch, while a few result from the dynamic flows themselves.

Fig. 11 and 12 illustrate the load distribution and winning time epoch distribution of a malicious switch which launches trivial and stealthy misreport attacks, respectively, on the same trace as shown in Fig. 10 (a). An obvious difference from the honest case is that there are few time epochs when the switch wins without misreporting. The probability of winning by reporting honestly during the second phase drops because the average load becomes higher due to the previous successful misreports. The malicious switch wins every epoch in which it misreports in the trivial attack while it does not in the stealthy attack. For example, the stealthy misreport attack at around 150s does not lead to a win in Fig. 12.



(a) winning short and bursty flows (b) winning some elephant flow

Figure 10. Load report of a honest member winning different flows

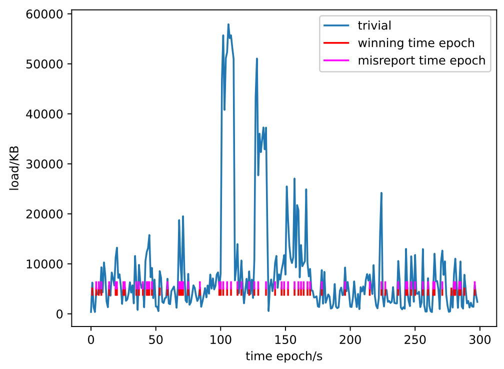


Figure 11. Load reports with the trivial attack.

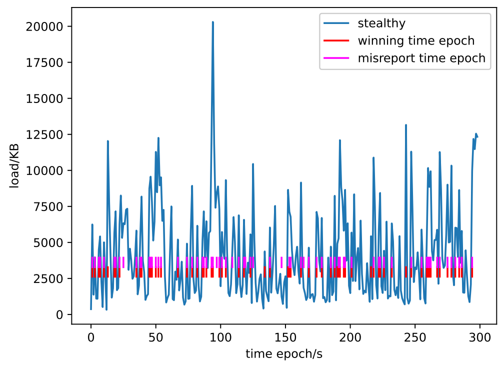


Figure 12. Load reports with the stealthy attack.

We find that the winning time epochs are not uniformly distributed when the switch reports honestly or misreports. Also, the load distribution over time is irregular as shown in Fig. 13. These two facts indicate that the switch which wins at a lucky time epoch during which flows with higher load arrive may obtain a heavier load compared with the other switches. In fact, for more than half of traces in the dataset, there are one or two members that receive $17\% \sim 60\%$ of the total load, leaving other members with less than 10% of the total load as shown in row 4 of Table I and Fig. 3. From another perspective, the load distributions across the pool balanced by the least load algorithm take a long time to converge to the equilibrium where all loads are identical or similar. Fortunately, this chaos opens up a time window for the misreport attack because low loads, whether faithful or not, may exist for some time. In summary, the uneven winning time epoch distribution and irregular load distribution over time leads to the fact that winning $X\%$ of the time does not guarantee winning $X\%$ of load.

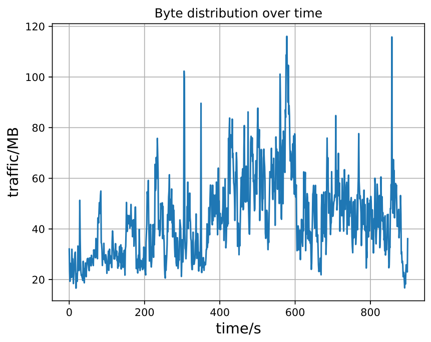


Figure 13. Uneven load distribution over time

5) Joint analysis of load and win rate: Here we analyze the relationship of the load and win rates further and address the anomalies in Table I. Looking deep into Fig. 8 and 9 we find that the load value may be dramatically different while the win rates are close. For example, in trace 10, the trivial attack captures 15% of the total load while the stealthy attack captures 25% though the stealthy attack win rate resembles the former. This further verifies the existence of lucky and unlucky time epochs, that is, the uneven load distribution over time.

Now we turn to anomalies in Table I. The extremely low honest win rates and zero stealthy attack win rates for some traces such as trace 0 can be attributed to the uneven distribution of flows and loads across the switches shown in row 4 of Table I and Fig. 3. Take trace 0 in column 2 for instance; the contrast between the few flows and heavy load suggests the existence of elephant flows, considering the fact that the flow size distribution follows a pareto distribution. As shown in Fig. 10 (b), elephant flows won early result in consistently high load, which prohibits the switch from winning again. In this edge case, a malicious switch cannot accurately approximate the load of other switches with its own for a stealthy attack (even with an extremely small p), and thus the attack fails. Similarly, if the malicious switch happens to be the outlier in Fig. 3, it will win additional load with a trivial attack but nothing with stealthy attack.

To understand the anomaly in Column 3 of Table I, we need the following lemma. The slight deviation in trace 9 can happen because Thm 1 only bounds the expectation of the win rate. $\Pr\{win\ rate \geq \frac{1-M}{N} + M\} > 0$ may hold by Lemma 3 when the expectation equals the upper bound.

Lemma 3. *Suppose we have a probability space S and a random variable X defined on S such that $\mathbb{E}(X) = \mu$. Then $\Pr\{X \geq \mu\} > 0$ and $\Pr\{X \leq \mu\} > 0$.*

Proof.

$$\mu = \mathbb{E}(X) = \sum_x x \Pr\{X = x\}$$

where the summation ranges over all the values in the range of X. If $\Pr\{X \geq \mu\} = 0$, then

$$\mu = \sum_x x \Pr\{X = x\} = \sum_{x<\mu} x \Pr\{X = x\}$$
$$< \sum_{x<\mu} \mu \Pr\{X = x\} = \mu$$

giving a contradiction. The other case is similar. $\square$

Essentially, it is the flows of various loads and diverse arrival pattern of the flows that cause the uneven load distribution over time. Eventually, switches observe a slight load imbalance due to the uneven load distribution over time and the uneven distribution of their winning time epochs.

6) Misreport efficiency: In this section we explain why the upper bounds seems loose in Fig. 8 and 9 and prove that the misreport attack is highly efficient.

For the win rates that fall into the estimated interval, the upper bound seems loose in most cases such as traces 6 and 10. This is because the estimation of the win rate by honest reports is not accurate. To have a better idea of the bias, we explore the three misreport cases below and the effects that the misreport has on the win rate of honest reports in the last 5mins. There are three cases when the malicious switch misreports at one time epoch:

- *Case 1:* either a misreport and honest report would win
- *Case 2:* neither a misreport nor honest report would win
- *Case 3:* a misreport changes the result and wins

When $p = 0.01$, $M_1 = 22.2\%$, $M_2 = 24.6\%$ and $N = 10$, the average probability of case 1 is $4.7\%(4.3\%)$ for the trivial (stealthy) attack, and that of case 2 is $0\%(3.0\%)$, which demonstrates the high efficacy of the misreport attack. On the other hand, the average win rate of the honest report drops to $4.2\%(2.7\%)$ when trivial (stealthy) attack exists, while the expectation of the honest report win rate is 7.7%. Generally speaking, the high average load brought by misreporting makes it hard to win by honest reporting.

Takeaway: Misreport attacks can control the traffic to a high degree by drawing a significant fraction of the traffic to the malicious switch. A stealthy attack has less efficacy than the trivial attack, which can bring more load or flows even in the edge cases with the price of higher detection risk. Thm. 1 bounds the attack win rate well but does not hold on some cases. The attack is highly efficient except for edge cases.

C. Ability to Learn the Traffic Faster

In this section we show the extent to which misreporting can accelerate learning about the network traffic and how the parameters affect this capability.

1) Parameters setting: We pick four 30-min traces of the same day from MAWI, starting at 9am, 12pm, 3pm, and 6pm since the link of MAWI samplepoint F observes significantly more load and flows during the day than the night[7]. We set the attack window $W = 900$s. The malicious switch reports honestly in the first 15 mins and misreports in the last 15 mins.

[7]We cannot carry out any experiment on a longer timescale such as weekly due to the anonymization of the dataset. We cannot repeat the experiments on other datasets because they do not have a two-day or three-day long backbone traffic trace.

2) Learning specific network communications: We study how much the malicious switch learns about specific network communications in this subsection. Set $p = 0.01$. Fig. 14 shows that given an IP source and destination pair, how many flows that belong to this IP pair are observed by the malicious switch under different circumstances. Specifically, we misreport every 4 seconds (25% of time) and start from two different initial time epochs to differentiate the misreport timing. The IP pairs are the ten with the highest frequency. We can tell that the misreport attack enables the malicious switch to observe many more flows per IP pair. The stealthy misreport attack has performance similar to the trivial attack when p is small, e.g., 0.01. The misreport timing makes little difference when the misreport frequency is the same and the attack is carried out uniformly over time. Here the trivial attack can capture nearly 100% of the flows of the ten IP pairs because the other pool members do not get starved and continue to report a non-zero load for a long time due to long-lived elephant flows.

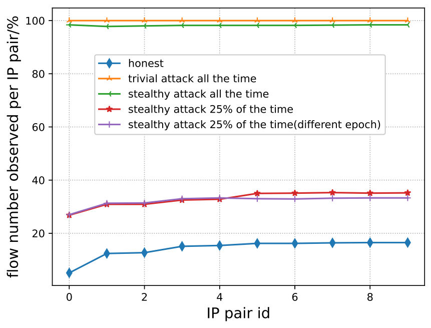


Figure 14. Flow number observed per IP pair

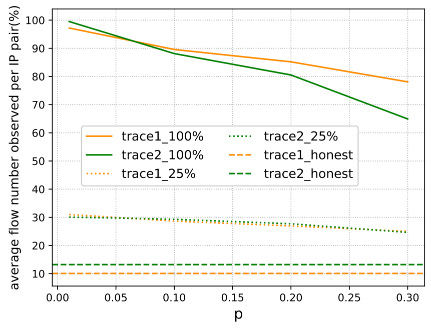


Figure 15. Ability to observe more traffic per IP pair: average over pairs.

Fig. 15 shows how the two parameters: misreport rate M and the percentile p affect the switch's ability to observe more traffic. In the figure, we investigate three cases: stealthy attack 100% of the time, stealthy attack 25% of the time and honest report on two traces. Recall that a smaller p results in the misreported load being closer to the minimum observed load and this is expected to result in a higher captured load. As we can see, p plays a less important role when the misreport frequency is low, which implies the adversary can set p higher

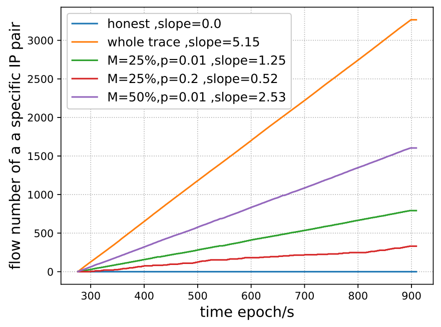


Figure 16. Ability to observe more flows of a specific IP pair

to be more stealthy with little cost when the misreport rate is low. Note that the choices of M and p are dependent. Given a fixed target, the choice of a smaller p will result in a higher misreport success probability and requires a smaller misreport frequency. But reporting a low load consistently is dangerous in terms of being detected. The inverse is also true. That is to say, there is a natural tradeoff between misreport success and detection risk.

Fig. 16 shows that the number of flows of a specific IP pair the malicious switch observes grows linearly with respect to time with different settings. The line labeled "whole trace" shows the ground truth when the switch wins every time epoch, so it represents the ideal case with 100% observability of the network and perfect learning. When p is sufficiently small like 0.01, the learning rate, i.e., the slope, correlates with M linearly and scales with perfect learning by a factor of M because the switch always wins when misreporting which dominates the win cases. So when $M = 50\%$, it observes 50% of the traffic and the learning rate is 50%. When p is larger like 0.2, the malicious switch will not always win when it misreports and may win on occasion when reporting honestly. As shown, even if the malicious switch does not capture more flows than average, it has access to the flows of the IP pair when misreporting while it does not when it reports honestly.

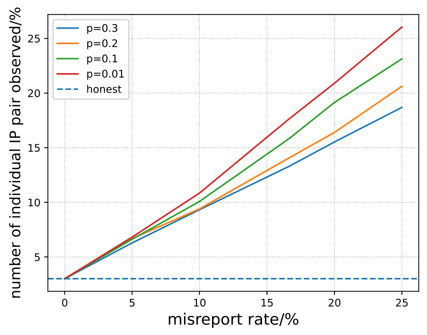


Figure 17. Ability to observe more individual IP pairs.

3) Learning communication patterns: While Section IV-C2 provides insights on how misreporting helps capture particular network communications, this part offers a unique perspective of the role misreporting plays in learning the network communication patterns by looking at the number of unique IP pairs observed. Fig. 17 reveals that the number of individual IP source and destination pairs observed by the malicious switch increases linearly with respect to the misreport rate of the stealthy attack. The impact of p is amplified at higher misreport rates in this case.

Takeaway: Misreporting enables a malicious switch to observing more flows per IP pair and more individual IP pairs. The individual IP pairs seen by the switch increases linearly with respect to the misreport rate M and the significance of p becomes more obvious with greater M.

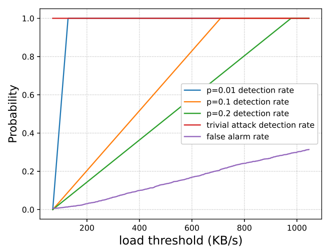
V. Stealthiness vs. Defenses

In this section we show how our proposed attack is stealthy against three basic approaches: a threshold detection algorithm [5], an algorithm based on Kolmogorov–Smirnov (KS) tests and one based on change point detection. When evaluating stealthiness in this section we set $N = 10, M = 25\%$ and use the 15-min daily traces.

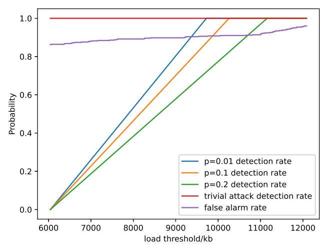
We define an attack as being stealthy if follows: 1) the probability that the attack is detected is much smaller than that of trivial attacks, and 2) the malicious switch behavior is not distinguishable from a percentage of honest switches where the percentage is an acceptable false alarm rate. We define the detection rate as the percentage of misreports that are detected, and the false alarm rate as the percentage of honest reports that are wrongly detected as misreports.

Prior work [15] gives a comprehensive introduction to the-state-of-art defense mechanisms as mentioned in I-A and explains why most of the approaches are not effective against the specific misreporting attack against ingress/egress switches.

One approach reviewed in [15], based on threshold detection [5] is effective against the trivial attack but not the stealthy attack. This approach provides two layers of defense: 1) it periodically checks whether the throughput of any port is below a threshold indicating it is likely reporting a low load value, and 2) it examines the consistency of the load reports of switches along a flow routing path. The latter test cannot identify the malicious ingress/egress switch in a flow path that adds/drops packets to influence the load report similarity since it relies on STATS-REPLY from untrusted switches along the flow path to calculate the inconsistencies.



(a) malicious switch



(b) honest switch

Figure 18. Threshold misreport detection evaluation for different switches

To use the threshold test, care must be taken in setting the threshold. If the threshold is set too low, then malicious switches that generate stealthy misreports can report values

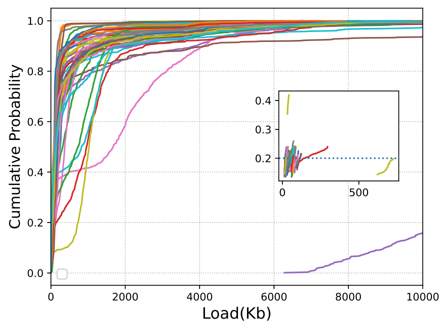


Figure 19. Distribution of load reports collected every second from a 50-member load balancing pool.

that are over the threshold and still gain large amounts of traffic. If the threshold is set too high to detect these stealthy misreports, the honest switches with naturally occurring low load will be erroneously detected as misreporting.

Fig. 18 shows how the threshold detection [5] works against misreporting attacks under different parameters. A threshold can be set that always detects the trivial attack with a false positive rate of 0 because honest switches rarely report no load, so the threshold can be set at a very low value and still detect the trival attack that reports 0 load. Since the trivial attack is easy to detect, we do not consider it further.

When using the threshold test to defeat a stealthy attacker, from the defender perspective, it is rational to pick a load threshold according the cumulative probability distribution of traffic through honest switches as shown in Fig. 3 so that only a small fraction of honest switches will be classified as malicious under normal conditions. For example, if we pick 600 Kb as the threshold in Fig. 18 (a), if the malicious switch sets $p = 0.2$, 50% of the misreports can be detected while the average false positive rate is 10%. Because the misreport rate, $M = 25\%$, the malicious switch is detected about 20% of the time. While this may seem effective, this threshold is harmful to many honest switches.

Recall that while some honest switches carry elephant flows, others are assigned small “mice” flows due to the uneven distribution of traffic. In Fig. 18 (b) we show the false positive rate for an honest switch that carries mice flows and as can be seen, this honest switch is classified as malicious more than 80% of the time which is four times the rate as the malicious switch.

Now we evaluate our stealthy attack against a KS test to compare the load report distribution of two switches in the same pool. In [15] the authors used a KS two sample test and median test to compare the load distribution of the potential malicious switch and the honest one. In this paper we have shown that the median distribution of traffic on flows, while similar across many switches, is not a good measure because of the existence of elephant flows. To see why please refer to Fig. 19. In this Fig it is clear that 3 out of the 50 flows deviate by more than 100% from the 50-percentile carried load of the remaining 47 switches. A malicious switch, with $p = 0.25$ and $M = 0.25$ can capture twice the normal load it would if it reports honestly, but its distribution is still be consistent with the distributions of these honest switches because it bounds its misreports based on its reconnaissance.

We also perform the KS two sample test between the load report cumulative probability distribution of a single edge switch $f(x)$ and the aggregation switch $g(x)$ when all the switches are honest. We conclude that with 99% confidence for only around 10% of the honest switches $f(x) = g(x)$, while $f(x) \geq g(x)$ for 96% of the honest switches. This happens because of the long-living elephant flows which result in heavy load on 5% of the switches. This implies that this comparison will have high false positive rates even for the honest case and is not an effective defense.

The last defense mechanism we evaluate is online change point detection. For example, we can track the load reports of a single switch and tell if there is a sudden change that may be a misreport. Fig. 20 shows the load report time series anomaly analysis by an online change point detection algorithm [46]. The load reports are collected from a malicious switch which records the load report in the first 10 mins and misreports in the last 5 mins. Only 5 spikes of anomaly scores are presented which means the probability that the stealthy attacks get detected is 6.7%.

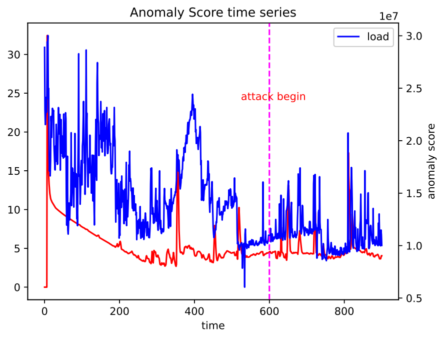


Figure 20. Ability to evade change point detection

Takeaway: The advantage of the stealthy attack is that the attacker can control when it receives the extra load but still appear to be similar to a percentage of honest switches where this percentage is an acceptable false positive rate.

VI. Conclusion

In this paper we defined and analyzed an attack on SDN load balancers. The attack includes a reconnaissance phase during which an attacker that has compromised a switch learns traffic distributions in a network and sets parameters, and an attack phase during which the attacker misreports its traffic load. We develop bounds on the win rates of misreporting switches based on which adversaries can set their attack parameters. We show that despite Internet traffic having uneven traffic distributions, these attacks are effective and allow adversaries to accelerate learning network traffic characteristics.

References

[1] J. Zhang, F. R. Yu, S. Wang, T. Huang, Z. Liu, and Y. Liu, "Load balancing in data center networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2324–2352, 2018.

[2] M. de Vivo, E. Carrasco, G. Isern, and G. O. de Vivo, "A review of port scanning techniques," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 2, pp. 41–48, April 1999.

[3] H. Kim and N. Feamster, "Improving network management with Software Defined Networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[4] Q. Burke, P. McDaniel, T. La Porta, M. Yu, and T. He, "Misreporting attacks in software-defined networking," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2020, pp. 276–296.

[5] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: detecting security attacks in software-defined networks." in *Ndss*, vol. 15, 2015, pp. 8–11.

[6] A. Kamisiński and C. Fung, "Flowmon: Detecting malicious switches in software-defined networks," in *Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense*, 2015, pp. 39–45.

[7] A. Akhunzada, E. Ahmed, A. Gani, M. K. Khan, M. Imran, and S. Guizani, "Securing software defined networks: taxonomy, requirements, and open issues," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 36–44, 2015.

[8] S. Hong *et al.*, "Poisoning network visibility in software-defined networks: New attacks and countermeasures." in *NDSS*, 2015.

[9] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful sdn data planes," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1701–1725, 2017.

[10] J. Cao, Q. Li, R. Xie, K. Sun, G. Gu, M. Xu, and Y. Yang, "The CrossPath attack: Disrupting the SDN control channel via shared links," in *USENIX Security*, 2019.

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

[12] M. Yu, T. He, P. McDaniel, and Q. K. Burke, "Flow table security in sdn: Adversarial reconnaissance and intelligent attacks," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1519–1528.

[13] S. M. Mousavi and M. St-Hilaire, "Early detection of ddos attacks against sdn controllers," in *Computing, Networking and Communications (ICNC), 2015 International Conference on*. IEEE, 2015, pp. 77–81.

[14] M. Yu, T. Xie, T. He, P. McDaniel, and Q. K. Burke, "Flow table security in sdn: Adversarial reconnaissance and intelligent attacks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 6, pp. 2793–2806, 2021.

[15] Q. Burke, P. McDaniel, T. La Porta, M. Yu, and T. He, "Misreporting attacks against load balancers in software-defined networking," *Springer Mobile Networks and Applications*, 2021.

[16] M. S. Kang, V. D. Gligor, V. Sekar *et al.*, "Spiffy: Inducing cost-detectability tradeoffs for persistent link-flooding attacks." in *NDSS*, vol. 1, 2016, pp. 53–55.

[17] Y. Zhang, "An adaptive flow counting method for anomaly detection in sdn," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, 2013, pp. 25–30.

[18] D. Thaler and C. Hopps, "Rfc2991: Multipath issues in unicast and multicast next-hop selection," 2000.

[19] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "Maglev: A fast and reliable software network load balancer," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 523–535.

[20] V. Olteanu, A. Agache, A. Voinescu, and C. Raiciu, "Stateless datacenter load-balancing with beamer," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 125–139.

[21] M. Hamdan, E. Hassan, A. Abdelaziz, A. Elhigazi, B. Mohammed, S. Khan, A. V. Vasilakos, and M. N. Marsono, "A comprehensive survey of load balancing techniques in software-defined network," *Journal of Network and Computer Applications*, vol. 174, p. 102856, 2021.

[22] J. Byers, J. Considine, and M. Mitzenmacher, "Simple load balancing for distributed hash tables," in *International Workshop on Peer-to-Peer Systems*. Springer, 2003, pp. 80–87.

[23] "Openflow switch specification," https://opennetworking.org/software-defined-standards/specifications/, 2015, [Online; accessed 5-January-2022].

[24] J. Li, X. Chang, Y. Ren, Z. Zhang, and G. Wang, "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, 2014, pp. 527–533.

[25] R. K. Arbettu, R. Khondoker, K. Bayarou, and F. Weber, "Security analysis of opendaylight, onos, rosemary and ryu sdn controllers," in *2016 17th International telecommunications network strategy and planning symposium (Networks)*. IEEE, 2016, pp. 37–44.

[26] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (sdn): a survey," *Security and communication networks*, vol. 9, no. 18, pp. 5803–5833, 2016.

[27] K. Thimmaraju, L. Schiff, and S. Schmid, "Outsmarting network security with sdn teleportation," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2017, pp. 563–578.

[28] K. Thimmaraju, B. Shastry, T. Fiebig, F. Hetzelt, J.-P. Seifert, A. Feldmann, and S. Schmid, "Taking control of sdn-based cloud systems via the data plane," in *Proceedings of the Symposium on SDN Research*, 2018, pp. 1–15.

[29] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open {vSwitch}," in *12th USENIX symposium on networked systems design and implementation (NSDI 15)*, 2015, pp. 117–130.

[30] P. Kampanakis, H. Perros, and T. Beyene, "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, 2014, pp. 1–6.

[31] S. Achleitner, T. La Porta, T. Jaeger, and P. McDaniel, "Adversarial network forensics in software defined networking," in *Proceedings of the Symposium on SDN Research*, 2017, pp. 8–20.

[32] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012.

[33] S. Feghhi and D. J. Leith, "A web traffic analysis attack using only timing information," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 8, pp. 1747–1759, 2016.

[34] J. C. C. Chica, J. C. Imbachi, and J. F. B. Vega, "Security in sdn: A comprehensive survey," *Journal of Network and Computer Applications*, vol. 159, p. 102595, 2020.

[35] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Flow wars: Systemizing the attack surface and defenses in software-defined networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3514–3530, 2017.

[36] H. Wang, L. Xu, and G. Gu, "Floodguard: A dos attack prevention extension in software-defined networks," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2015, pp. 239–250.

[37] C. Sony and K. Cho, "Traffic data repository at the wide project," in *Proceedings of USENIX 2000 Annual Technical Conference: FREENIX Track*, 2000, pp. 263–270.

[38] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM computer communication review*, vol. 38, no. 4, pp. 63–74, 2008.

[39] "Caida ucsd anonymized internet traces, howpublished = "http://www.caida.org/data/passive/passive_dataset.xml", note = "[online; accessed 5-january-2022]", year=2019."

[40] R. Fontugne, P. Abry, K. Fukuda, D. Veitch, K. Cho, P. Borgnat, and H. Wendt, "Scaling in internet traffic: a 14 year and 3 day longitudinal study, with multiscale analyses and random projections," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2152–2165, 2017.

[41] "Mininet," http://mininet.org/, 2010, [Online; accessed 5-January-2022].

[42] "Ryu controller," https://ryu-sdn.org/, 2014, [Online; accessed 5-January-2022].

[43] H. Qian and D. Medhi, "Server operational cost optimization for cloud computing service providers over a time horizon." in *Hot-ICE*, 2011.

[44] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011, pp. 254–265.

[45] P. D. Mcdaniel, S. Sen, O. Spatscheck, J. E. van der Merwe, W. Aiello, and C. R. Kalmanek, "Enterprise security: A community of interest based approach." in *NDSS*, vol. 6, 2006, pp. 1–3.

[46] K. Yamanishi and J.-i. Takeuchi, "A unifying framework for detecting outliers and change points from non-stationary time series data," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 676–681.