# Slow TCAM Exhaustion DDoS Attack*

Túlio A. Pascoal[1], Yuri G. Dantas[2], Iguatemi E. Fonseca[1], and Vivek Nigam[1,3]

[1] Federal University of Paraíba, João Pessoa, Brazil
tuliopascoal@gmail.com, {vivek,iguatemi}@ci.ufpb.br
[2] TU Darmstadt, Darmstadt, Germany
dantas@mais.informatik.tu-darmstadt.de
[3] fortiss, Munich, Germany

**Abstract.** Software Defined Networks (SDN) facilitate network management by decoupling the data plane which forwards packets using efficient switches from the control plane by leaving the decisions on how packets should be forwarded to a (centralized) controller. However, due to limitations on the number of forwarding rules a switch can store in its TCAM memory, SDN networks have been subject to saturation and TCAM exhaustion attacks where the attacker is able to deny service by forcing a target switch to install a great number of rules. An underlying assumption is that these attacks are carried out by sending a high rate of unique packets. This paper shows that this assumption is not necessarily true and that SDNs are vulnerable to Slow TCAM exhaustion attacks (Slow-TCAM). We analyse this attack arguing that existing defenses for saturation and TCAM exhaustion attacks are not able to mitigate Slow-TCAM due to its relatively low traffic rate. We then propose a novel defense called SIFT based on selective strategies demonstrating its effectiveness against the Slow-TCAM attack.

**Keywords:** DDoS Attacks, SDN, Low-Rate Attacks, Selective Defenses

## 1 Introduction

In Software Defined Networks (SDN), a powerful controller is responsible for taking the decision of where packets should be forward, *i.e.*, defining the network flows (control plane), while the task of forwarding packets is left to powerful switches (data plane). Whenever a packet arrives a switch, it searches whether there is a matching installed rule. This search is efficient because of dedicated memories called Ternary Content-Addressable Memory (TCAM) where forwarding rules are stored. If no rule is applicable, the switch informs the controller which takes a decision by, for example, installing a new rule.

However, TCAM are expensive and have high power comsumption [14]. Therefore, SDN switches have a limited TCAM space [14,15,10] and can store

* This work has been funded by the DFG as part of the project Secure Refinement of Cryptographic Algorithms (E3) within the CRC 1119 CROSSING, by RNP, by Capes and CNPq.

only a limited number of rules (typically 1500 to 3000 rules) [25,14,13,27,10]. This limitation has led to TCAM exhaustion [10,17,13,29,24] and saturation attacks [31,12,11,29,28,14,2]. In a saturation attack, the attacker forces the target switch to install a great number of new rules consuming the switch TCAM capacity and moreover causing the whole network controller to crash because of increased traffic between the switch and the controller.

We describe some approaches for mitigating saturation and TCAM attacks:

1. Setting rule timeouts which remove a rule whenever it is not used for some given duration. This timeout is called *idle timeout* in the OpenFlow protocol used in SDNs as the basic mechanism for removing obsolete rules. There have been proposals [25] for optimizing timeout values according to the network behavior and by flow aggregation.
2. Monitoring the number of unpaired rules, *i.e.*, rules for which there is an incoming flow, but no outgoing flow. The purpose is to detect DDoS attacks in general, as it can be used to detect when a packet has a spoofed IP. Similarly, there have been defenses that evaluate TCP SYN cookies in order to validate TCP Handshake of the packets in order to detect IP spoofing [26].
3. Monitoring the rate that rules are installed. If the rate of rule installation is too high, then it is likely that the network is suffering an attack and defense mechanisms may be triggered [10];
4. CPU and memory of SDN Switches and Controllers also provide indications that a system is suffering a DDoS attack and trigger countermeasures [28].

The main underlying assumption of these measures, however, is that attackers will send unique packets in a very high rate by, for example, spoofing IPs. This causes some of these parameters to change abruptly triggering counter-measures.

### 1.1 Slow TCAM Exhaustion Attacks

The assumption that attackers only generate high traffic is not necessarily true. Indeed as witnessed by the class of Low-Rate Application Layer DDoS attacks, such as Slowloris, attackers can deny service of a web-server or a VoIP server by sending a very low rate of requests to the target server [9,18,8]. Attackers can also carry out Low-Rate attacks on not powerful devices using SlowDroid [6,3] and exploit new vulnerabilities on application layer protocols in order to evade detection mechanisms, *e.g.*, SlowNext [5].

Inspired by Low-Rate Application DDoS Attacks, our first contribution is the identification of the vulnerability of SDN to Slow TCAM attacks. We propose a novel attack called *Slow TCAM Exhaustion attack* (Slow-TCAM) which is carried out as follows:

1. Recruit a large enough number of bots, typically a number a bit greater than a half the rule capacity of the target switch. A number between 1500 – 3000 is enough. Notice that the attacker is not spoofing IPs.
2. Each bot sends a unique packet to the target switch. Whenever the switch receives the first packet, a new rule is installed. Moreover, since there is no IP spoofing, two flows, an incoming and outgoing flow, are eventually installed.

3. The unique packet generation rate is controlled so that the rate that new rules are installed is not too high. In our experiments, the attacker generates a traffic of up to 40 packets per second, while typical flooding and saturation attacks generate a traffic greater than 1000 packets per second [13,10,26,28].
4. Finally, each bot keeps sending at a low rate a packet to the switch within its rule idle timeout. The idle timeout can be inferred by the attacker by try and error using SDN SCANNER [25]. Therefore, no rule is uninstalled leaving the TCAM always full and not allowing new rules to be installed.

After the Slow-TCAM attack is carried out, the controller and the switch operate normally, but they are forced to serve only flow rules installed by the attacker thus denying service to legitimate clients.

Our second contribution is the proposal of SIFT, *SelectIve DeFense for TCAM*, a selective defense for Slow-TCAM attack. Our previous work used selective strategies to mitigate Low-Rate Application-Layer DDoS attacks on web and VoIP servers [8,9,18,19]. This paper shows that selective strategies can mitigate Slow-TCAM attack by randomly selecting rules to be dropped whenever the system is overloaded. We built SIFT over the Openflow protocol, *i.e.*, no additional SDN machinery is necessary nor hardware, and it runs in conjuction with the controller. Whenever a switch has its rule capacity full, *i.e.*, the controller receives a TABLE-FULL message, SIFT is activated and decides using a probability distribution whether a new rule is going to be installed or not. We demonstrate that SIFT is a lightweight defense for Slow-TCAM attacks with low impacting on the controller's CPU and memory consumption. Moreover, when under attack, SIFT mitigates the attack leading to high levels of availability.

Similarly to our previous work on selective strategies for Low-Rate Application-Layer DDoS attacks [9], we have also formalized the Slow-TCAM and SIFT in Maude and used Statistical Model Checking techniques to validate our results. The formalization can be found at [1], but due to space limitations is left out of the scope of this paper which focuses on the experimental results obtained.

The rest of the paper is structured as follows: Section 2 details the Slow-TCAM attack arguing why it is a fatal attack on SDN. Section 3 details our experimental results demonstrating the efficiency of the attack. Section 4 discusses means to mitigate Slow-TCAM attack and introduces the defense SIFT based on selective strategies showing that it can mitigate Slow-TCAM attacks. Finally, in Section 5, we conclude by discussing related and future work.

## 2 Slow TCAM Exhaustion Attack (Slow-TCAM)

While we assume that the reader is familiar with the OpenFlow protocol used in SDN, we review some of the messages exchanged between a SDN switch and the controller. Whenever a packet is received by a SDN switch, it checks whether there is a matching forwarding rule. If so, it applies the rule to this packet. However, if no rule is applicable, *i.e.*, it is a *new unique packet*, the switch

exchanges the following messages with the controller:

$$\begin{aligned} \text{Switch} &\rightarrow \text{Controller} : \text{PACKET-IN} \\ \text{Controller} &\rightarrow \text{Switch} : \text{FLOW-MOD} \end{aligned}$$

The message PACKET-IN contains the incoming packet information, *e.g.*, header, buffer_id, in_port, payload, etc. It will contain simply the header if the switch's incoming buffer[4] is not full, and it will contain the whole packet if its buffer is full. FLOW-MOD contains the rule that should be installed by the switch. Once the message FLOW-MOD is received by the switch, it checks whether there is enough space in the switch's TCAM memory for installing a new rule. If this is the case, the rule is installed and the packet is forwarded using it. Otherwise, the switch drops the packet and informs the controller that its TCAM memory is full by sending the following message:

$$\text{Switch} \rightarrow \text{Controller} : \text{TABLE-FULL}$$

The controller can specify a rule idle timeout. (OpenFlow comes with a hard timeout which is deprecated.) Given a rule timeout of $TO$, a rule is uninstalled by the switch if it is not triggered for $TO$ time units. The use of timeouts is a mechanism to remove less used rules freeing TCAM memory for other rules to be installed. Typically, the timeout $TO$ is a value between 9-11 seconds [31].

Finally, we point out that the communication between a SDN switch and the controller is expensive as it builds a secure channel for their communication. Therefore, a defense should avoid switcher-controller communication overhead.

### 2.1 Attacking SDN

As TCAM are expensive and consume a great amount of energy, SDN switches have limited TCAM space, consequently are not able to store many rules, typically a number between 1500 and 3000 rules [25,14,13,27,10]. There have been attacks on SDN which attempt to (1) consume the TCAM memory of switches (TCAM exhaustion attack) and (2) overload the controller (saturation attack).

These attacks are carried out by sending a great number of unique packets, normally by spoofing IPs. Once the TCAM is exhausted, the switch starts to drop packets leading to the TCAM exhaustion attack. Moreover, the saturation attack goes even further by sending unique packets at a even greater rate consuming not only the switch's TCAM memory, but also the switch's incoming buffer. The switch, then, starts sending to the controller the whole packet instead of only the packet header. This overloads the controller leading it to crash thus affecting the whole SDN.

Defenses for the TCAM exhaustion attack and the saturation attack assume that the attacker necessarily sends a great number of unique packets, *i.e.*, flood the switch, to deny its service. Existing defenses monitor parameters that could

[4] Not to confuse the incoming packet buffer which stores packets with the TCAM which stores rules.

be affected when receiving a large number of unique packets, *e.g.*, rule installation rate, CPU and Memory, number of unpaired rules.

However, this assumption is not necessarily true. We identify that SDNs are vulnerable to Slow TCAM exhaustion attacks, where the attacker exhausts a switch's TCAM memory without sending unique packets at a high rate.

### 2.2 Slow-TCAM

Inspired by Low-Rate Application DDoS attacks [9,18,8,4], such as Slowloris, we propose a variant of the TCAM exhaustion attack, called Slow TCAM Exhaustion Attack (Slow-TCAM), which does not need to send a great number of unique packets, *i.e.*, flood the system, but rather is able to slowly occupy all a switch's TCAM resources and deny service to legitimate.

In order to carry out a Slow-TCAM attack, we assume that the attacker has a botnet with more than the rule capacity of the target switch, *i.e.*, typically 1500 – 3000 bots.[5] This is feasible as he can recruit a botnet using standard methods, *e.g.*, phishing or purchasing such botnet[6]. We also assume that the attacker knows the rule timeout $TO$. This can be easily inferred by using existing tools such as an SDN Scanner [25][7] which uses a try and error approach applying statistical testing methods, *e.g.*, t-test analysis.

The Slow-TCAM attack then proceeds as follows:

- **Rule installation:** Coordinates its botnet to send a unique packet to the target switch directed to some service in the SDN, for example a web-server, at a low rate. Once a unique packet is received, the target switch follows the OpenFlow protocol which causes it to install a rule. As the rate of unique packets that are arriving is low, the rate of rule installation is also low.
- **Rule Activation:** Once a bot has send its first unique packet causing the target switch to install a rule, it sends packets in intervals of less than the timeout $TO$. This causes the corresponding rule to be fired and therefore to not be removed by the rule timeout mechanism.

As we demonstrated by our experiments in Section 3, Slow-TCAM can be quite effective:

- **Low Attacker Effort:** The main effort from the attacker is to recruit a large enough botnet. Once he possesses such botnet, the traffic generated by the botnet is very low compared to usual traffic. Bots have to send a single packet in intervals of less than $TO$, which is typically every 10s;
- **Disguised Attack:** As the traffic generated is low, it is hard to detect the Slow-TCAM attack. Indeed, differently from the saturation attack which causes the controller to crash, a Slow-TCAM attack does not stress the controller's memory and CPU resources. This renders defense that monitor these parameters ineffectively. Moreover, the rule installation rate is low thus not indicating

[5] One can reduce this number by half as a flow has an incoming and outgoing flows.

[6] https://tinyurl.com/zf27emp

[7] It is possible to carry out a Slow-TCAM attack by IP spoofing. However, this attack could be easily mitigated by checking for unpaired rules.

| Average Attack Rate | Success Rate | TTS | Time to DoS | CPU | Memory Usage |
|---|---|---|---|---|---|
| No Attack | 100% | 12,6ms | – | – | – |
| 3.2 unique packets/s | 0.0% | $\infty$ | 478s | 2.5% | 42.3 MB |
| 4.6 unique packets/s | 0.0% | $\infty$ | 324s | 3.83% | 43.0 MB |
| 5.8 unique packets/s | 0.0% | $\infty$ | 258s | 4.74% | 42.3 MB |
| 9.2 unique packets/s | 0.0% | $\infty$ | 162s | 4.98% | 42.5 MB |
| 13.6 unique packets/s | 0.0% | $\infty$ | 110s | 6.39% | 42.2 MB |
| 15.6 unique packets/s | 0.0% | $\infty$ | 96s | 7.17% | 41.9 MB |
| 23.6 unique packets/s | 0.0% | $\infty$ | 63s | 10.43% | 41.8 MB |
| 39.5 unique packets/s | 0.0% | $\infty$ | 38s | 10.97% | 42.3 MB |

**Table 1.** Slow-TCAM: Time to Service and Availability. The value on Success Rate corresponds to the number of clients that are able to obtain a response after the attacker has carried out the attack and occupied all the TCAM memory.

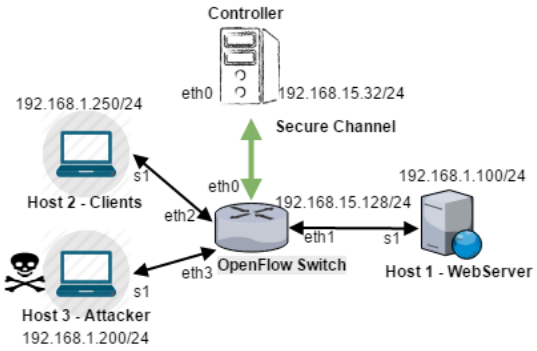


**Fig. 1.** Experimental Set-Up.

a malicious over use of the network. In fact, the attack can be in principle made to be as slow as desired as bots can install rules in a slower rate bypassing defenses based on traffic monitoring.

– **Effectiveness:** It effectively denies service to legitimate clients. As the attacker occupies the target switch's TCAM, legitimate clients packets are no longer forwarded being dropped and therefore they cannot access the services provided by the SDN.

## 3 Slow-TCAM Experimental Analysis

We implemented the Slow-TCAM attack and carried out a number of experiments. Figure 1 shows the set-up of our experiments. We used two virtual machines, one executing Mininet [20] along with Open vSwitch 2.5.0 [22], which are a well-known network emulator and open-source virtual switch, respectively. Another virtual machine executed the SDN controller Ryu [23] using OpenFlow 1.3 [21]. The Mininet machine was a Ubuntu 14.04 LTS, Intel i7-5500U

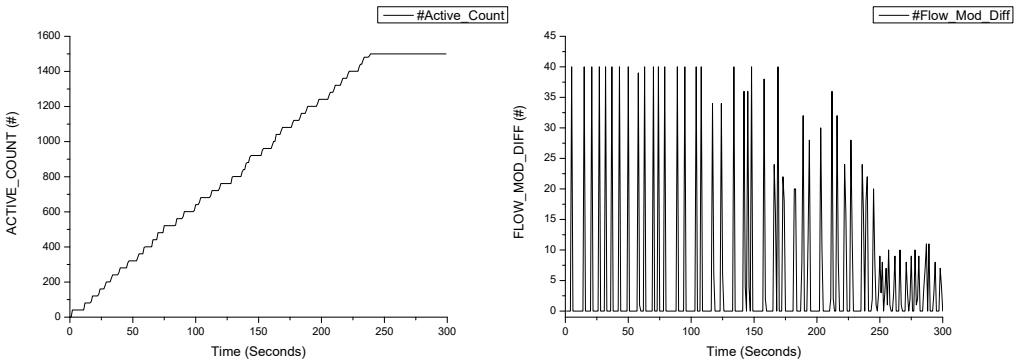


**Fig. 2.** Number of installed rules in the Target SDN Switch and the number of FLOW-MOD messages sent by the controller for a Slow-TCAM attack with intensity of 5.8 unique packets per second.

CPU@2,40GHz with 3GB of RAM memory, while the Ryu machine was a Ubuntu 16.04.1 LTS, Intel i7-5500U CPU@2,40GHz with 1GB of RAM memory. The host machine was a Windows 10 64 bit, Intel i7-5500U CPU@2,40GHz with 8GB of RAM memory.

We set the SDN switch rule capacity to 1500 rules with rule timeout $TO$ of 10 seconds as recommended in the literature [31].

Legitimate client traffic (Host 2) consisted of 375 unique connections, which means the installation of 750 rules (incoming and outgoing rules) in a switch, *i.e.*, half the switch rule capacity. We implemented the Slow-TCAM attack where the attacker (Host 2) possesses a botnet with more than 760 bots and no more than 800 bots. Both legitimate and attacker's bots accessed the web-server (Host 1).

Table 1 summarizes our experimental results. It shows that Slow-TCAM attack can be effective in denying service to legitimate clients accessing the network using a SDN switch. We carried out a number of experiments with different attack intensities from 3.2 unique packets per second to 39.5 unique packets per second. In comparison typical flooding attacks has a rule installation rate of 1000 unique packets per second [13,10,26,28]. Once the attacker successfully occupied all the TCAM memory, every one of its bots sends with periodicity of 3s a packet to keep its corresponding rule active in the SDN switch.

We measured the legitimate client availability after the attacker has occupied all the TCAM memory, time to service (TTS), the time for the attacker to deny service, the controller's average CPU and memory usage. We observed that the attacker can carry out the attack very slowly occupying all the TCAM memory in around 8 minutes or more quickly in only 38 seconds. There is little impact on the controller CPU usage and memory.

Figure 2 illustrates the TCAM usage by the Slow-Attack with intensity of 5.8 rules per second. It takes a bit more than 4 minutes to occupy all the rule capacity by installing 1500 rules. The remaining scenarios with different attack intensities had the same behavior. For our slowest attack with intensity of 3.2 unique packets per second, the attacker can deny service even more silently in

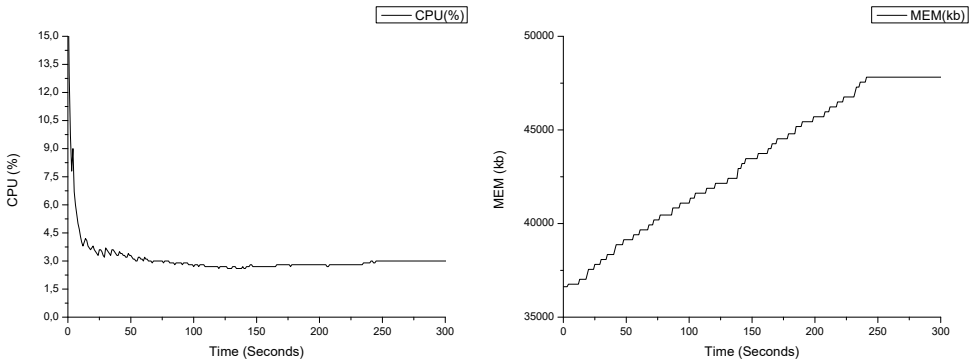


**Fig. 3.** Slow-TCAM: CPU and Memory Usage during a Slow-TCAM attack with intensity of 5.8 unique packets per second.

around 8 minutes with practically no impact on the controller's CPU usage. On the other hand, the attacker can also deny service more quickly in 38 seconds by carrying out a Slow-TCAM attack with intensity of 39.5 unique packets per second with still a very low impact on the controller's CPU usage. Notice that the attacker is able to keep the rules installed in the switch by avoiding their timeout to be fired. This can be observed by the fact that no rules are uninstalled. Once all 1500 rules are installed, there is no more room for new rules thus denying service to legitimate clients.

We also measured the number of FLOW-MOD messages sent by the controller (also illustrated in Figure 2). As the attack is slow, it causes the controller to send a low amount of FLOW-MOD messages (less than 40) and once the TCAM is occupied the number of FLOW-MOD messages reduces even further. Notice as well that this number can also be reduced if the attacker is willing to carry out an attack with an even lower rate.

We measured the CPU and memory effort of the controller during the Slow-TCAM attack depicted in Figure 3. The Slow-TCAM attack causes a low overhead on the CPU usage of less than 5% and little impact to the switch's memory usage from 34MB to less than 43MB which is due to the installation of new rules.

These results demonstrate that the Slow-TCAM attack is indeed an effective and silent attack as it denies service to legitimate clients without changing in abrupt ways the main parameters used by monitoring defenses (see Section 5 for futher details). The rule rate installation and of FLOW-MOD messages is kept low and there is little impact to the Switch's CPU and Memory. Moreover, as the attacker is not spoofing IPs, all rules installed in the switch to handle his packets are paired, *i.e.*, have an incoming and outgoing rules.

**Algorithm 1** SIFT Execution During a Round

```
1: procedure SIFT-ROUND
2:     if Received FLOW-MOD with ruleIns then
3:         ruleList.insert(ruleIns)
4:         lastFlowMod ← ruleIns
5:     if Received TABLE-FULL then
6:         pmod ← pmod + inc
7:         ruleList.remove(lastFlowMod)
8:         if random() < k/(k+pmod) then
9:             iRuleInd ← random(k)
10:            ruleDr ← ruleList.get(iRuleInd)
11:            ruleList.remove(ruleDr)
12:            send OFPFC_DELETE with ruleDr
13:            if hasPairRule(ruleDr,ruleList) then
14:                rulePair ← ruleList.getPair(ruleDr)
15:                ruleList.remove(rulePair)
16:                send OFPFC_DELETE with rulePair
17:    if Received OFPRR_DELETE or OFPRR_IDLE_TIMEOUT with ruleDr
   then
18:        ruleList.remove(ruleDr)
```

## 4 Mitigating Slow-TCAM

Before we introduce our new defense SIFT for mitigating Slow-TCAM attacks, we discuss some alternative defenses mechanisms. A detailed analysis of their applicability is left to future work:

- **Rule aggregation:** It seems possible to mitigate Slow-TCAM attack by aggregating different rules into broader rules. The controller can reduce the impact of Slow-TCAM as the attacker would not be able to to consume all the target switch's TCAM. The downside of using rule aggregation is that the controller has a coarser definition of unique packet and therefore, the system becomes more vulnerable to other attacks, such as volumetric attacks;
- **Dynamic Timeouts:** If the controller is able to distinguish a bot from a legitimate client, the controller can set different timeouts allowing rules created for possible clients to have longer timeouts. It is not yet clear how to set these timeouts with the Slow-TCAM attack as bots may behave very close to legitimate clients, *e.g.*, access a web-page with an expected behavior.
- **Improving TCAM usage:** The switch may improve its TCAM usage by storing less data for example. This mechanism may increase a switch's rule capacity and therefore, the attacker would need to hire a larger botnet to carry-out a Slow-TCAM attack.

### 4.1 SIFT

We propose a new defense called *SelectIve DeFense for TCAM* (SIFT) for defending against the Slow-TCAM attack. It is based on selective strategies [8,9,18] which have already been used to mitigate Low-Rate Application Layer DDoS attacks on web-servers and VoIP servers, such as Slowloris.

SIFT is executed together with the controller at the controller-layer. Assume that the switch rule capacity is $k$. SIFT maintains three variables:

`ruleList`, `lastFlowMod`, and `pmod`

where `ruleList` is a mirror list of the rules installed in the switch and `pmod` is a counter. Selective strategies including SIFT work in rounds with duration of $T_R$ time units. Our experiments demonstrate that $T_R = 0.1s$ is a suitable value for a round duration being able to mitigate attacks with very little overhead on the controller's CPU and memory usage. At the beginning of a round, SIFT sets a counter `pmod := 0`.

During a round, SIFT follows Algorithm 1. Whenever a new rule `ruleIns` is to be installed, *i.e.*, a FLOW-MOD is generated, then SIFT adds this rule to `ruleList` (lines 2-3). If a TABLE-FULL is received from the switch informing that a rule `ruleTB` was not able to be installed, then SIFT proceeds as follows: first it increments `pmod` by a value `inc` (line 6). Our experiments show that `inc = 100` is a good value for a switch with rule size 1500.

SIFT then generates a random number between 0 and 1 and checks whether this number is less than (line 8):

$$\frac{k}{k+\texttt{pmod}}$$

If this is not true, then SIFT simply rejects the rule *ruleTB* and leaves the currently installed rules as they are. Otherwise, SIFT drops a randomly chosen installed rule *ruleDr* so that new rules may be added. As `pmod` increases, the probability of installing new rules decreases with the rate of incoming traffic (for more formal justification for this rule see [16]).

If SIFT decides to install *ruleTB* (lines 9 -16), SIFT selects a number *iRuleDr* between 1 and $k$ and removes the rule *ruleDr* at the index *iRuleDr* from `ruleList` (lines 9-11). It then sends the OpenFlow message OFPFC_DELETE to the switch specifying that the rule *ruleDr* should be uninstalled (line 12). As the rule *ruleDr* has been uninstalled, we also search whether it has a pair rule and uninstall it as well (lines 14-16) as it would no longer have an incoming or outgoing flow.

Finally, whenever the switch uninstalls a rule sending an OFPRR_DELETE or an OFPRR_IDLE_TIMEOUT message, the corresponding rule is removed from `ruleList` (lines 17-18).

Notice that SIFT has a concrete effect on the switch only when its rule capacity is reached. If there is still space in the TCAM for new rules, the network behaves as if SIFT is not present.

**Rationale of Why SIFT Works:** The objective of the attacker is to keep its rules installed for long periods of time. Therefore, whenever a switch's rule capacity is reached, which is likely due to an attack, SIFT has a greater prob-

| Client Traffic | Without SIFT | | With SIFT | |
|---|---|---|---|---|
| | Success Rate | Median TTS | Success Rate | Median TTS |
| No Attack | 100% | 23.7ms | 100% | 20.2ms |
| 1 packet every 1-3s | 0% | $\infty$ | 97.3% | 97ms |
| 5 packets every 1-3s | 0% | $\infty$ | 96.9% | 1061ms |
| 10 packets every 1-3s | 0% | $\infty$ | 97.9% | 1082ms |
| 15 packets every 1-3s | 0% | $\infty$ | 98.9% | 1149ms |
| 100 packets every 10s | 0% | $\infty$ | 95.6% | 2454ms |

**Table 2.** SIFT: Time to Service and Availability when under an attack of intensity of 5.8 unique packets per second. The value on Success Rate corresponds to the number of clients that are able to obtain a response after the attacker has carried out the attack and occupied all the TCAM memory.

ability of selecting an attacker rule and enabling new rules for serving possibly legitimate clients to be installed.

**Variations of SIFT:** The results obtained in this paper assumes a uniform probability mechanism for choosing which rule to drop. Our experiments indicate that this strategy is effective for mitigating Slow-TCAM attacks. However, there are other selective strategies [8,18,19] that could be used, *e.g.*, taking into account the time a rule has been installed or the number of packets that fired a rule, etc. We leave this investigation to future work.

### 4.2 Experimental Results with SIFT

We carried out load tests with scenarios with SIFT and without SIFT when under an Slow-TCAM attack of intensity of 5.8 unique packets per second. These tests provide us with *lower bounds* on the performance of our defense. We varied the intensity of legitimate client traffic from 1 packet in intervals of 1-3 seconds (chosen randomly), to 15 packets every 1-3 seconds. We also tested SIFT when there is a burst of legitimate client traffic with 100 packets every 10s.

Table 2 summarizes the results with different scenarios. It first shows that SIFT does not have an impact when the system is not under attack. Then, it shows that the Slow-TCAM attack is effective in denying service resulting in 0% availability in all cases when not running SIFT. With SIFT, on the other hand, one is able to maintain high levels of availability with levels above 95% for each scenario. SIFT had, however, an impact on the time to service (TTS) specially when there are burst of client demand reaching 2.4 seconds. We are currently investigating how to improve TTS by using different selective strategies and incorporating other defense technique such as those described at the beginning of Section 4. This is left to future work.

Finally, we measured the impact of SIFT on the controller's memory and CPU. It is a lightweight defense not impacting the CPU and Memory consumption of the controller. This can be observed by the graphs depicted in Figure 4.

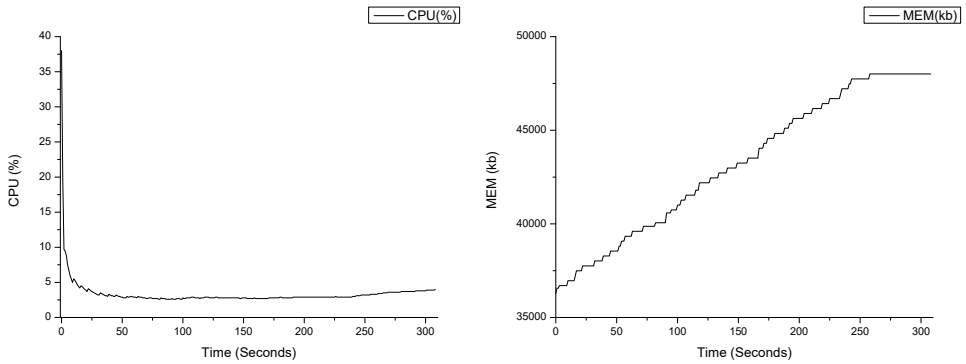


**Fig. 4.** SIFT: CPU and Memory Usage during a Slow-TCAM attack of intensity of 5.8 unique packets per second.

SIFT did not cause overhead on the CPU and memory usage when compared with the data in Figure 3.

## 5 Related and Future Work

The main objective of TCAM exhaustion attacks is to force the switch to install rules. In the literature, this is accomplished by sending a high rate of unique packets, *e.g.*, using spoofing and sending UDP packets [17,13,10]. Furthermore the saturation attack [31,12,11,29,28,14,2] has as main objective to crash the controller by sending a large amount of traffic to a SDN switch occupying its incoming buffer. This causes the switch send to the controller the whole packet instead of only sending the packet header.

Dhawan *et al.* [10] propose the detection of DoS attacks by monitoring the rate of rule creation by the SDN controller. If this rate passes a threshold, then mitigation actions are taken. Since the Slow-TCAM attack can be configured to set a particular rate of rule creation, this defense is not effective in mitigating the Slow-TCAM attack.

Shin and Gu [25] propose two mechanisms to detect TCAM exhaustion and saturation attacks. The first mechanism is compute an optimal rule timeout according to the complexity of the network. While this mechanism can mitigate flooding attacks which use IP Spoofing of UDP packets, for example, it is not efficient for mitigating Slow-TCAM attacks as the bots are continuously sending valid packets thus resetting timeouts. The second mechanism is the technique of Flow Aggregation which generates more general rules, defining macroflows, instead of using more specific rules, defining microflows. This strategy can mitigate the Slow-TCAM, but at the expense of leaving the network more vulnerable to other attacks, *e.g.*, Get-Flooding, allowing malicious traffic to use the network. Moreover, as pointed out by [28], Flow Aggregation is not capable of mitigating saturation attacks such as the ones proposed by [7].

The strategy AVANT GUARD [26] detects when a TCP-handshake is completed before creating rules in the network. It has been recently shown [2] that this defense is vulnerable to a modification of the saturation attack capable to consume all AVANT GUARD's resources. AVANT GUARD's strategy cannot detect Slow-TCAM attacks as the attacker's bots complete TCP-handshakes.

Wang *et al.* [28] propose to monitor switch buffer, controller's CPU and memory usage to mitigate saturation attacks. As the Slow-TCAM attack has little impact to these parameters, it seems that the defense proposed by Wang *et al.* is not effective in detecting Slow-TCAM attacks.

Shen [24] proposes a peer support strategy where SDN switches share their unused TCAM memory space among them when they are reaching its TCAM limit. This is done by installing flow rules in the attacked switch (they keep a reserved space in TCAM) in order to divert flows to other peer switches according to: not being full, nearest to the attacker switch, less busy, connects to more other switches. However they can only retard the attack and has the problem of when the majority or many switches are full they will divert traffic between them ending up in loop.

Some proposals [30,15] suggest modifications to the OpenFlow protocol used in SDN and in the structure of TCAM memory in order to improve memory management. Kandoi and Antikainen [13] comment the possibility of using Optimal Timeout technique and Flow Aggregation. However, their goal is to enhance SDN general performance, whereas we expose the TCAM limited space SDN vulnerability as a mean to deny its service.

We are currently investigating the use of alternative selective strategies for mitigating not only Slow-TCAM attack, but also saturation attacks. These selective strategies would use parameters such as CPU, Memory usage, number of times a rules has been fired. The probability of dropping a rule would then depend on such parameters. For example, rules that have not been frequently used should have a higher probability of being dropped. We believe that by using more parameters we can improve SIFT and mitigate other TCAM attacks.

## References


1. SIFT https://github.com/ygdantas/SIFT.git. 2016.
2. Moreno Ambrosin, Mauro Conti, Fabio De Gaspari, and Radha Poovendran. Lineswitch: Efficiently managing switch flow in software-defined networking while effectively tackling DoS attacks. In *ASIACCS*, pages 639–644. ACM, 2015.
3. Enrico Cambiaso, Gianluca Papaleo, and Maurizio Aiello. Slowdroid: Turning a smartphone into a mobile attack vector. In *FiCloud, 2014*, pages 405–410. 2014.
4. Enrico Cambiaso, Gianluca Papaleo, Giovanni Chiola, and Maurizio Aiello. Slow DoS attacks: definition and categorisation. *IJTMCC*, 1(3-4):300–319, 2013.
5. Enrico Cambiaso, Gianluca Papaleo, Giovanni Chiola, and Maurizio Aiello. Designing and modeling the slow next DoS attack. In *CISIS and ICEUTE*, 2015.
6. Enrico Cambiaso, Gianluca Papaleo, Giovanni Chiola, and Maurizio Aiello. Mobile executions of slow DoS attacks. *Logic Journal of IGPL*, 2015.
7. Andrew R Curtis, Wonho Kim, and Praveen Yalagandula. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *INFOCOM*, pages 1629–1637. IEEE, 2011.

8. Yuri Gil Dantas, Marcilio O. O. Lemos, Iguatemi Fonseca, and Vivek Nigam. Formal Specification and Verification of a Selective Defense for TDoS Attacks. In WRLA, 2016.
9. Yuri Gil Dantas, Vivek Nigam, and Iguatemi E. Fonseca. A Selective Defense for Application Layer DDoS Attacks. In *JISIC 2014*, pages 75–82, 2014.
10. Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. SPHINX: Detecting Security Attacks in Software-Defined Networks. In *NDSS*, 2015.
11. Xinshu Dong, Hui Lin, Rui Tan, Ravishankar K Iyer, and Zbigniew Kalbarczyk. Software-defined networking for smart grid resilience: Opportunities and challenges. In CPSS, 2015.
12. Sungmin Hong, Lei Xu, Haopei Wang, and Guofei Gu. Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures. In *NDSS*, 2015.
13. Rajat Kandoi and Markku Antikainen. Denial-of-service attacks in OpenFlow SDN networks. In *IM*, 2015.
14. Kalapriya Kannan and Subhasis Banerjee. Compact TCAM: Flow entry compaction in TCAM for power aware SDN. In *ICDCN* 2013.
15. Naga Katta, Omid Alipourfard, Jennifer Rexford, and David Walker. Infinite cacheflow in software-defined networks. In *HotSDN*, pages 175–180. ACM, 2014.
16. Sanjeev Khanna, Santosh S. Venkatesh, Omid Fatemieh, Fariba Khan, and Carl A. Gunter. Adaptive selective verification: An efficient adaptive countermeasure to thwart dos attacks. *IEEE/ACM Trans. Netw.*, 20(3):715–728, 2012.
17. Rowan Klöti, Vasileios Kotronis, and Paul Smith. Openflow: A security analysis. In *ICNP* 2013.
18. Marcilio O. O. Lemos, Yuri Gil Dantas, Iguatemi Fonseca, Vivek Nigam, and Gustavo Sampaio. A Selective Defense for Mitigating Coordinated Call Attacks. In SBRC, 2016.
19. Marcílio O. O. Lemos, Yuri Gil Dantas, Iguatemi E. Fonseca, and Vivek Nigam. On the Accuracy of Formal Verification of Selective Defenses for TDoS Attacks.
20. Mininet. 2016. http://www.mininet.org/. Accessed in: 02 November 2016.
21. OpenFlow. *Open Networking Foundation*. https://www.opennetworking.org/.
22. OpenVSwitch. 2016. http://openvswitch.org/. Accessed in: 14 November 2016.
23. Ryu. 2016. https://osrg.github.io/ryu/. Accessed in: 10 November 2016.
24. Jinan Shen. Defending against Flow Table Overloading Attack in Software-Defined Networks IEEE Transactions on Services Computing.
25. Seungwon Shin and Guofei Gu. Attacking software-defined networks: A first feasibility study. In HotSDN, pages 165–166. ACM, 2013.
26. Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: scalable and vigilant switch flow management in software-defined networks. In CCS 2013.
27. Anilkumar Vishnoi, Rishabh Poddar, Vijay Mann, and Suparna Bhattacharya. Effective switch memory management in openflow networks. In *DEBS* 2014.
28. Haopei Wang, Lei Xu, and Guofei Gu. Floodguard: a dos attack prevention extension in software-defined networks. In DSN, pages 239–250. IEEE, 2015.
29. Mingxin Wang, Huachun Zhou, Jia Chen, and Bo Tong. An Approach for Protecting the OpenFlow Switch from the Saturation Attack. 2016.
30. Minlan Yu, Jennifer Rexford, Michael J Freedman, and Jia Wang. Scalable flow-based networking with difane. *ACM Computer Communication Review*, 2010.
31. Adam Zarek, Y Ganjali, and D Lie. Openflow timeouts demystified. *Master Thesis, Univ. of Toronto, Canada*, 2012.