# LineSwitch: Efficiently Managing Switch Flow in Software-Defined Networking while Effectively Tackling DoS Attacks

Moreno Ambrosin, Mauro Conti*, Fabio De Gaspari,
University of Padua, Italy
{surname}@math.unipd.it
fabio.degaspari@studenti.unipd.it

Radha Poovendran
University of Washington, USA
rp3@uw.edu

## ABSTRACT

Software Defined Networking (SDN) is a new networking architecture which aims to provide better decoupling between network control (control plane) and data forwarding functionalities (data plane). This separation introduces several benefits, such as a directly programmable and (virtually) centralized network control. However, researchers showed that the required communication channel between the control and data plane of SDN creates a potential bottleneck in the system, introducing new vulnerabilities. Indeed, this behavior could be exploited to mount powerful attacks, such as the *control plane saturation attack*, that can severely hinder the performance of the whole network.

In this paper we present LineSwitch, an efficient and effective solution against control plane saturation attack. LineSwitch combines SYN proxy techniques and probabilistic blacklisting of network traffic. We implemented LineSwitch as an extension of OpenFlow, the current reference implementation of SDN, and evaluate our solution considering different traffic scenarios (with and without attack). The results of our preliminary experiments confirm that, compared to the state-of-the-art, LineSwitch reduces the time overhead up to 30%, while ensuring the same level of protection.

## Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Security and protection

## Keywords

Software-Defined Networking (SDN); Denial-of-Service (DoS); SYN Flooding Attack

*Mauro Conti is supported by a Marie Curie Fellowship funded by the European Commission under the agreement No. PCIG11-GA-2012-321980. This work is also partially supported by the TENACE PRIN Project 20103P34XC funded by the Italian MIUR, and by the Project "Tackling Mobile Malware with Innovative Machine Learning Techniques" funded by the University of Padua.


*ASIA CCS'15,* April 14–17, 2015, Singapore.

http://dx.doi.org/10.1145/2714576.2714612.

## 1. INTRODUCTION

The great increase in demand for flexibility and automation in key Information Technology sectors is leading to the rise of new paradigms that greatly simplify the management of network infrastructures. One of such paradigms is Software Defined Networking (SDN). Unlike the current network infrastructure, SDN decouples the network layer (i.e., the *control plane*) and the data layer (i.e., the *data plane*) functionalities in separate entities. This separation allows a (virtually) centralized and directly programmable network control, while at the same time reduces and abstracts the complexity of network devices.

The most widely adopted instantiation of the SDN paradigm is OpenFlow (OF) [11, 2]. OF provides a standard communication interface between the data plane, and the control plane, and introduces the concept of *flows* to identify the network traffic [2]. Each OpenFlow entity, namely *OF switch*, maintains a set of *flow tables*. They specify the rules that the OF switch uses to perform the routing of packets. Moreover, flow tables are organized in a pipeline, which is traversed by the switch every time a packet is received. The control plane can program the flow tables of the OF switches either statically or dynamically. In the latter case, if an OF switch does not have a matching rule in its pipeline for a new incoming packet, it must contact the control plane to retrieve a new rule [1]. Unfortunately, while enabling a flexible network management, the required extensive communication between control and data plane might result in poor scalability. Moreover, it introduces a serious vulnerability that can be exploited to overload the control plane with flow requests: the resulting attack is called *control plane saturation* [17, 8], and can be easily performed, for example, through SYN flooding [15]. By overloading the control plane, this attack incapacitates the target OF switch, which will not be able to retrieve rules for new network flows. Furthermore, if the controller manages more than a single switch, the attacker might hinder an even larger part of the network [17, 9, 8, 6].

Recently, Shin et al. [17] proposed AVANT-GUARD, a countermeasure against the control plane saturation attack. AVANT-GUARD introduces a new module into the OF switch, called *connection migration* module, which protects the switch and the controller from saturation attacks performed by SYN flooding, while at the same time being transparent to the end hosts. With this module, each OF switch acts as a SYN proxy during the TCP handshake stage of a connection, effectively shielding the controller from possible floods. Unfortunately, while being beneficial in the general

case, this technique introduces new subtle vulnerabilities and heavy limitations. Indeed, we will show that when running AVANT-GUARD, the state the OF switch needs to maintain in order to proxy each connection can lead to a new Denial of Service attack, that we refer to as *buffer saturation attack*. Furthermore, the transparency required with respect to the end hosts limits the number of connections that the switch can proxy to the number of available TCP port numbers.

*Our Contribution.* In this paper, we make the following contributions:

- We identify and discuss some unintended vulnerabilities of one of the recently proposed schemes against the control plane saturation attack that, for the best of our knowledge, represents the state-of-the-art solution against this threat.
- We propose a novel attack, which we name *buffer saturation attack*. Our attack exploits some of the identified vulnerabilities introduced by the state-of-the-art solution for the control plane saturation attack. As confirmed by our analysis, *buffer saturation attack* is both realistic and simple to run, and leads to significant network performance degradation.
- We propose LineSwitch, a new efficient and effective solution to mitigate the control plane saturation attack. LineSwitch greatly reduces the effects of this attack, while at same time protects the network from the *buffer saturation attack*.
- We did a preliminary evaluation of our solution, which confirms the effectiveness of LineSwitch against the *control plane saturation attack*. Moreover, our experiments show a significant reduction of the time overhead (up to 30%) when compared to the state-of-the-art.

*Organization.* The remaining of this paper is organized as follows. In Section 2 we provide some background knowledge on SDN, and on the SYN flooding attack and its possible countermeasures. In Section 3 we revise some related work in the area of DoS attacks and defense in Software Defined Networking. Moreover, we provide a brief introduction of AVANT-GUARD, which represents the state-of-the-art solution against the control plane saturation attack. In Section 4 we analyze the limitations of the current state-of-the-art, and introduce a new possible attack, i.e., the buffer saturation attack. Section 5 describes LineSwitch, our countermeasure against the control plane saturation attack in SDN, while in Section 6 we provide a preliminary evaluation of its effectiveness. Finally, in Section 7 we draw our conclusions.

# 2. PRELIMINARIES

In this section we introduce some concepts that will be used in the remaining of the paper: Software Defined Networking (Section 2.1) and the SYN flooding attack (Section 2.2).

## 2.1 Software Defined Networking (SDN)

Software Defined Networking (SDN) has emerged as a new network paradigm aimed at providing higher flexibility in network research, development and operation. The core concept behind SDN is the separation of two distinct aspects of networking that in today's architecture are blended together: the *network control* and the *forwarding functions*. The SDN architecture postulates that these two logically separated aspects of networking are decoupled in two corresponding layers, respectively the *control plane* and the *data plane*. Figure 1 provides a high-level representation of the SDN architecture.

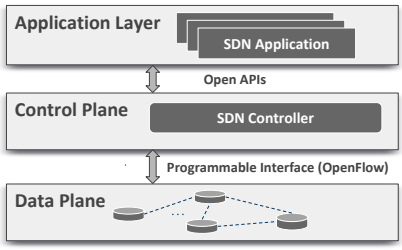



Figure 1: SDN Architecture.

The control plane provides a directly programmable layer that acts as an interface to the data plane for applications, abstracting both the complexity of the underlying network infrastructure and the communication between the two planes. Using the control plane as a middleware, it is possible to easily deploy a great variety of network management applications, that act independently from the physical network devices. Moreover, the control plane offers a logical centralized system that controls and accesses data from all network devices, effectively offering a global view of the network infrastructure. OpenFlow (OF) [2] is the reference implementation of the SDN paradigm. It defines a standard communication interface between the control plane and the data plane. With OpenFlow, the routing is performed based on traffic *flows*; each OF network entity (i.e., *OF switch*) maintains one or more *flow tables*, that are used to route incoming packets, and an OF channel to an external controller. The control plane can program the physical devices through a series of *flow rules* [1], that are installed inside the flow tables. Such rules specify which actions a switch will perform on a specific network flow. For each unique network flow, or group of flows, there will be a corresponding flow table entry (i.e., a flow rule). Once an OF switch receives a packet, it matches the packet header against the pipeline of its local flow tables, which will dictate what actions will be applied to that specific flow. The control plane populates the flow tables of each OF switch either statically, by pre-installing a set of flow rules into the flow tables, or dynamically, by allowing a switch-driven run-time installation of new flow rules [2]. In the latter case, if an OF switch does not have any matching rule for a new incoming flow, it forwards the corresponding packet header to the controller, which at this point can install a new rule for that flow into the switch [1]. Through the combination of different flow rules, a controller can define a broad range of actions, from the standard routing of a packet to a more complex analysis, involving forwarding the packet to the controller [1].

## 2.2 SYN Flooding Attack

SYN flooding attacks are one of the most widespread and effective DoS attacks [15]. The effectiveness of a SYN flooding attack is based on the state that TCP stacks allocate when a connection request (i.e., a SYN packet) is received. Such state, called *Transmission Control Block* (TCB) [3], is retained in memory for a certain amount of time, even if the TCP handshake is not completed by the client. This is

needed to handle the case in which the ACK packet, needed to complete the connection, is lost due to network congestion. Once a critical amount of half-open connections is reached, the available memory reserved to the TCBs is saturated, and no new connections can be established. In general, in order to make it difficult for the victim to detect the attack and its source, an attacker performs a SYN flooding attack using spoofed IP addresses. In this way, the victim will not be able to identify the source of the attack, and therefore, it will not be able to automatically stop the SYN flooding. Indeed, new incoming SYN packets could belong to legitimate connections, which the victim can not distinguish from the malicious traffic. Researchers and industry proposed several countermeasures against the SYN flooding attack [4, 15]. However, none of them offers a definitive solution to this important problem.

When applied to SDN, the SYN flooding attack does not aim at saturating internal data structures. Rather, the goal is to exploit data to control plane communication to saturate the controller by generating a huge number of new network flows. For each flow, the OF switch will have to contact the controller, which will need to analyze the flow information, prepare and then send an OF response to the switch. If the rate of the flood is high enough, the controller will not be able to keep up with the attack and will be incapacitated from serving legitimate network flows.

# 3. RELATED WORK

In this section, we provide a brief overview of the main research studies related to Denial of Service (DoS) attacks in SDN. Due to space limitations, in this section we focus only on DoS attacks against the control plane of an SDN network. In [15], Peng et al. provided a first feasibility study for Denial of Service attacks on SDN control plane. In [10], Kreutz et al. analyzed the SDN architecture, identifying critical aspects and possible new attack vectors, while Kloti et al. [9] assessed some vulnerabilities that affects OpenFlow [2], i.e., the possibility for an attacker to mount DoS attacks on the control plane, and to disclose potentially sensitive information by using timing analysis techniques. In [17], Shin et al. propose AVANT-GUARD, a solution that addresses architectural flows of the original OpenFlow protocol by altering the flow management at the data plane level [8, 17]. In particular, the authors focused on solving OpenFlow's control plane saturation attack vulnerability. To the best of our knowledge, AVANT-GUARD [17] represents the state-of-the-art for tackling the control plane saturation attack in SDN. For this reason, in the remaining of this section we briefly describe this solution. We will further compare AVANT-GUARD against our solution when evaluating our proposal (see Section 6).

AVANT-GUARD [17] adds two extensions to the standard OpenFlow protocol: (1) a *Connection Migration* module, which limits the effect of the control plane saturation attack based on SYN flooding by proxying the incoming TCP requests, and (2) the *Actuating Trigger* module, which allows the controller to limit the number of control messages required to collect network statistics. Since the focus of this paper is on solving the control plane saturation attack, in the remaining of this paper we will focus only on the connection migration module of AVANT-GUARD, that we briefly describe in this section. Moreover, in Section 4 we will provide a security analysis of the connection migration module.

As introduced in Section 2.1, whenever an OF switch receives an inbound network flow for which it has no flow rule, it will forward the packet to the control plane. This behavior holds even for SYN packets: indeed, for each received SYN packet not matching a flow rule, an OF switch will contact the controller to obtain a corresponding rule. The proposed connection migration module of AVANT-GUARD addresses this problem at the data plane level, by having the OF switch act as a SYN proxy.

This process is articulated in four phases (see Figure 2):

*Classification phase.* When an OF switch receives a SYN packet belonging to an unknown flow (action (1) in Figure 2), instead of forwarding it to the control plane, the switch acts as a proxy, engaging the client in a stateless TCP handshake through SYN Cookies (actions (2) and (3) in Figure 2).

*Report phase.* If the client completes the TCP handshake, the switch then forwards the new flow to the controller (action (4) in Figure 2) and waits for a new rule that defines how it should be handled (action (5) in Figure 2).

*Migration phase.* If the controller allows the migration, the switch initiates a TCP handshake with the destination host (actions (6), (7) and (8) in Figure 2). The OF switch further reports the result of the handshake to the control plane (actions (9) and (10) in Figure 2).

*Relay phase.* If the handshake is successful, the switch forwards all the subsequent messages between the client and the destination host.

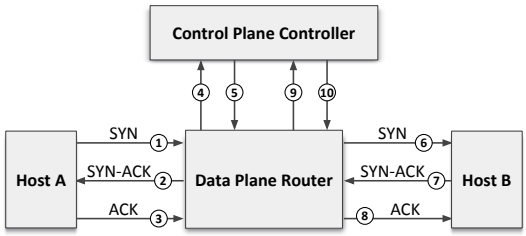



Figure 2: AVANT-GUARD [17] – Connection Migration.

The foremost advantage of connection migration, is the classification mechanism. Only complete TCP flows will be reported to the control plane, effectively shielding it from SYN flooding attacks performed with spoofed IP addresses, and greatly mitigating the threat of link saturation. For non-spoofed TCP flows, the result is that any $\{IP, port\}$ combination will appear to be valid, effectively converting the network to a whitehole network and preventing an attacker from mapping possible targets. Moreover, the consequent SYN flooding vulnerability at data plane level is addressed by the use of SYN Cookies [7]. Since the SYN Cookie algorithm does not need to maintain state for connection requests, there is no need for storing information in the OF switch for failed TCP connections.

Generally, it is possible to infer the use of connection migration by analyzing the round trip time of a SYN packet. An adversary might use this information to flood the data plane with complete TCP handshakes, forcing the switch to forward each of them to the control plane. To solve this issue, AVANT-GUARD provides a modification to the basic connection migration module, namely *delayed connection migration* [17], which requires the initiator of the communication to send the first valid packet, before forwarding the packer header to the control plane.

# 4. LIMITATIONS OF THE CONNECTION MIGRATION MODULE

The connection migration module of AVANT-GUARD [17] is indeed a valid solution against the control plane saturation attack. Moreover it also shields end hosts from SYN flooding attacks and, by replying unconditionally to every received SYN packet, it prevents port scanning attacks. Unfortunately, along with the above desirable properties, the connection migration module of AVANT-GUARD introduces new vulnerabilities too. In particular, we identified two distinct vulnerabilities that can lead to a shutdown of the OF switch. First, the state that the switch must keep in order to implement the connection migration module of AVANT-GUARD can be exploited by an attacker, which can try to fill the allotted buffers and incapacitate the OF switch (see Section 4.1). Second, the use of SYN proxy limits the number of connections that can be forwarded. In particular, the maximum number of connections forwarded to a specific $\{IP, port\}$ pair is 64513 (see Section 4.2).

In what follows we provide an in-depth analysis, as well as the scheme for possible attacks, for each of the above points.

## 4.1 Proxying Requires State

A switch implementing the connection migration module of AVANT-GUARD needs to maintain some state, for the whole duration of the TCP connection. In particular, acting as a proxy between two communicating hosts, $A$ and $B$, a switch $R$ should execute the following three operations:

(1) Once received a SYN packet from host $A$, with Initial Sequence Number [3] $ISN_A$, $R$ will respond with a SYN-ACK packet with a spoofed address, i.e., using host $B$ address. The ACK sequence number will be $ISN_A + 1$ and the sequence number will be a random number $ISN_R$. Note that, according to TCP protocol specifications [5, 13] each ISN is computed in a non-predictable way. Therefore, it is impossible for $R$ to predict the ISN that $B$ would generate and, as a consequence $ISN_R$ will be different from the sequence number $B$ will use.

(2) Upon receiving the permission to migrate the connection, switch $R$ will start a handshake with host $B$ by sending a SYN packet with sequence number $ISN_A$. Note that at this stage, $R$ must use its IP address to establish the TCP session since it has no guarantees that the reply from host $B$ will follow the same path through switch $R$ on the way back.

(3) Once the SYN-ACK packet from host B (with an ACK number $ISN_A + 1$ and a sequence number $ISN_B$) is received, switch R finalizes the connection sending an ACK with number $ISN_B + 1$ to B.

To maintain the connection migration transparent to the end hosts, $R$ must perform a *sequence number translation* (and analogously an ACK translation), for the packets exchanged by $A$ and B. Moreover, for each host connecting to the same $\{IP, port\}$ pair, the switch needs to use a distinct port number to migrate the connection in order to later match the response packets to the correct host on the way back. Consequently, for each connection the switch needs to store the following information:

$$\{IP_{src}, port_{src}, port_R, \delta_{seq}\},$$

where $IP_{src}$ and $port_{src}$ are respectively source address and port of the initiator of the connection, $port_R$ is the port number used by the router in the migration and $\delta_{seq}$ is the difference between the ISN used by the router and the ISN used by the destination host.

The translation table required by the switch to act as a proxy gives an attacker an easy mean for mounting a *buffer saturation attack*. Indeed, the attacker simply needs to open several complete TCP connections through the target OF switch to a given host. Note that each of these connections will need state to be stored on the switch for translation. Therefore, if the number of connections is large enough, the portion of memory dedicated to that data structure will be saturated, incapacitating the switch from serving any further valid connection.

## 4.2 Limit on the Number of Connections

As we already stated in Section 4.1, when a connection from hosts $A$ to $B$ is proxied by switch $R$, all the packets translated by the switch to the destination $B$ will have the IP address of $R$ and a port number which will be different from the original one used by A. This behavior introduces yet another important problem: if there are several clients attempting to connect to a given destination on the same port (e.g., $\{IP_B, port_B\}$) through switch $R$, the latter will need to use a different port for each outgoing connection. However, since TCP port numbers are 16 bit fields, the maximum number of connections the switch will be able to migrate to a given IP-port pair is at most $2^{16} - 1024 = 64512$ (the first 1024 ports are reserved for well known services). This number can be quickly reached if we consider extremely popular HTTP services (e.g., Google or Facebook). Therefore, each switch is bound to a maximum number of connections it can migrate for each service, after which all new incoming connection requests can not be satisfied.

The limited number of available ports can easily be exploited by an attacker to mount DoS attacks. In fact, it would be easy to target a given service by opening a series of long-lasting connections through the OF switch. Once enough connections are opened and all the possible ports have been used by the switch to migrate the connections, any other client trying to connect to said service will be rejected. This bound on the maximum number of connections that a switch can migrate, provides a simple and efficient way of mounting a DoS attack to a given host $B$, for all the clients whose path to $B$ passes through the same OF switch. There are no definitive solutions to this problem if proxying is used: each connection to the same service must be assigned a unique port number by the switch. The most promising way to somewhat mitigate this restriction is to purchase several IP addresses for the switch to use. In this way, when the OF switch consumes all the available ports with a given address, it will switch to a new address, this way being able to migrate other 64512 connections. While address purchasing can be employed as a partial solution, it is worth noting that for each additional address, the space of possible combinations increases by just 64512. If we consider a complex network, where several switches employ the connection migration technique, we will quickly hit a point where the cost and complexity of management increase extremely rapidly, reducing the appeal of this workaround.

# 5. OUR SOLUTION: LINESWITCH

Breaking TCP end-to-end semantics introduces the need to store state, which in turn opens the system to attacks exploiting buffer saturation. Therefore, there is a strong need to reduce its use as much as possible while retaining its beneficial effects against control plane saturation attacks.

To reach this goal, we propose LineSwitch, an OpenFlow extension intended for edge OF switches of a network. The idea is the following: LineSwitch proxies all incoming TCP connections from a given IP until one is completed, while for all subsequent SYN packets the proxy is used with a small probability $P_p$.

Our solution effectively protects the switch even in presence of an attacker $E$ with knowledge about the proxy mechanism in use. Indeed, it would be possible for $E$ to perform an attack by correctly completing the handshake associated with the first SYN packet sent, and then initiating the SYN flooding. Although this is true, $E$ would be forced to use its real IP address in all the packets, to ensure they will be forwarded to the OF pipeline. Otherwise, they will be proxied by the switch, and therefore discarded. Moreover, since the effectiveness of the SYN flooding attack is based on a high throughput, once $E$ is forced to use its real IP address for the flooding, the OF switch will quickly proxy one of the packets, thus detecting the attack. Then, the OF switch can blacklist the IP address of host $E$, $IP_E$, for $T \times 2^{count_{IP_E}}$ seconds, where $count_{IP_E}$ indicates the number of times $IP_E$ did not complete a connection, and $T$ represents a default time value.

With our approach, all packets with spoofed IP addresses will be blocked at the data plane, and malicious clients that initiate SYN flooding with non-spoofed IPs (after first establishing a complete TCP connection) will be penalized with rapidly increasing blacklist periods of time.

Our approach provides two major improvements with respect to the problems identified in the state-of-the-art (see Section 4). The first advantage of our approach is that it drastically reduces the memory usage needed to perform the address/port translation for each connection, thus offering protection against the described buffer saturation attack (see Section 4.1). Indeed, LineSwitch requires port translation only for the first SYN packet per each IP address, and only for a very small number of packets after that based on the chosen probability $P_p$. Therefore, the memory usage increases roughly linearly with the number of clients with a TCP connection through the switch R. In contrast, the memory overhead introduced by AVANT-GUARD grows linearly *with the number of connections* that passe through the switch. An attacker can easily generate a huge number of connections from the same source IP using different port numbers (up to $2^{16} = 65535$ per $< IP_{dst}, port_{dst} >$ pair; the theoretical limit would then be $2^{16+32+16}$) and, with AVANT-GUARD, the switch would need to store state for each of these connections. As a reference, in our experiments with a link of 1 Mbps we were able to open approximately 780 connections per second, while with a higher bandwidth of 5 Mbps, it was possible to complete more than 4000 connections per second. Instead, with LineSwitch the number of entries the switch needs to store under attack, is roughly proportional to the number of distinct real IP addresses (machines) the attacker possesses. As a consequence, the effect of buffer saturation attacks is greatly reduced, while at the same time retaining full protection against SYN flooding attacks.

A second major advantage with respect to the state-of-the-art is that LineSwitch proxies a minimum number of connections. Indeed, as discussed in Section 4, while being an effective mechanism to protect against SYN flooding attacks, proxying introduces several problems which derive from breaking the end-to-end paradigm. Therefore, its use should be limited as much as possible. LineSwitch proxies only the first connection from a given host (i.e., an IP address), and subsequent connections from the same host with probability $P_p$. Since, in general, the migration probability value $P_p$ is small (see Section 6), LineSwitch will assure in most cases the normal network flow, mitigating intrinsic problems of proxying such as limited maximum number of migrated connections (see Section 4.2). Moreover, our experimental results show that a small value for $P_p$ does not reduce the level of protection that LineSwitch provides against the control plane saturation attack (see Section 6).

# 6. PRELIMINARY EVALUATION

In order to assess the effectiveness of LineSwitch against the control plane saturation attack, and its resiliency to buffer saturation attacks, we designed preliminary experiments on the setting shown in Figure 3. Our system model includes three hosts connected to an OF switch running the reference OpenFlow software switch [14], and a local controller (running the POX controller, `l3_learning` module [16]). We ran all our experiments using the Mininet network simulator [12] in a virtual machine. We compared LineSwitch to the state-of-the-art solution to tackle SYN flooding-based control plane saturation attack, i.e., AVANT-GUARD [17]. The computer used for the simulation is equipped with a quad core Intel i5-4670 @3.40GHz, all of which were available to the virtual machine.

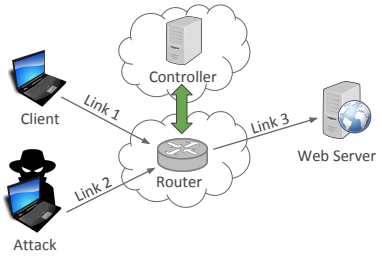
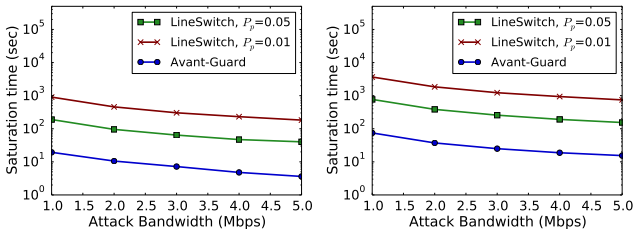



Figure 3: Experimental Setup.

We simulated the behavior of both AVANT-GUARD and LineSwitch under the buffer saturation attack introduced in Section 5. To this aim, we configured the system with different buffer sizes and run the attack at different rates. As a result, we demonstrate that: (1) the attack rate required to successfully incapacitate an OpenFlow switch running AVANT-GUARD grows *linearly with the size of the buffer*; (2) when using AVANT-GUARD, the throughput needed to successfully complete the attack in a reasonable amount of time *is easily achievable, even with larger buffers*; (3) LineSwitch offers an extremely high resiliency to the buffer saturation attack, and can be further configured through the $P_p$ parameter to address the specific needs of the network.

In order to directly assess the relation between attack bandwidth and time required to saturate the buffer, we set the RTT is set to 0ms, for each link in Figure 3. In any case, since the attacker continuously floods the switch with new connections, the RTT would have been relevant just until the first connection were completed. Figure 4 presents the results of our simulation. It shows the average time needed to successfully overload a switch with a buffer saturation attack, running both AVANT-GUARD and LineSwitch, with the latter executed with parameter $P_p$ set to 0.01 and 0.05. The results

are presented for varying size of the buffer (expressed in Bytes) and for different rates of attack (expressed in Mbps).




(a) buffer size $2^{20}$Bytes (b) buffer size $2^{22}$Bytes

Figure 4: Average buffer saturation time, varying buffer size and attack bandwidth, under a buffer saturation attack. Time on y-axis is in logarithmic scale.

As Figure 4 shows, even with a modest rate of attack it is possible to quickly overflow the buffer of a switch running AVANT-GUARD: with an attack rate of 1 Mbps, a buffer of $2^{22}$ Bytes is saturated in 74.718 sec, preventing the switch from migrating any new connection. By contrast when using LineSwitch, even when setup with a highly conservative migration probability $P_p = 0.05$, the time needed to perform a successful buffer saturation attack is one order of magnitude greater when compared to AVANT-GUARD. As an example, with a 1 Mbps attack rate, a buffer size of $2^{22}$ Bytes and $P_p = 0.05$, LineSwitch requires 769.487 sec to be saturated against only 74.718 sec required when running AVANT-GUARD. When using lower (and more realistic) migration probability values, the time difference increases even more, as shown in Figure 4.

For completeness, we compared the average overhead introduced by AVANT-GUARD with the overhead introduced by LineSwitch, evaluating both a scenario without attack, and a scenario under SYN flooding based control plane saturation attack. All overhead data is expressed with respect to the standard OpenFlow protocol, under normal network conditions (e.g., no attacks performed). We sampled the time required by the legitimate client (see Figure 3) to download a web page of size 1 KB from the web server. In the regular traffic scenario, AVANT-GUARD introduces an average time overhead of 41.83%, while LineSwitch incurs only a 7.67% overhead. Moreover, under control plane saturation attack with an attack rate of 6.5 Mbps, AVANT-GUARD introduces an overhead of 36.92%, while LineSwitch introduces only a 5.45% overhead. Finally, under control plane saturation attack, both AVANT-GUARD and LineSwitch guarantee a 100% page retrieval success rate.

# 7. CONCLUSION

In this paper we analyzed the effects of the control plane saturation attack based on SYN flooding, one of the most widespread types of denial of service attack, when applied to Software Defined Networks (SDN) architecture, and in particular to its reference implementation, OpenFlow. We showed that the extensive communication needed by the control plane and the data plane in SDN amplifies the effect of typical denial of service attacks, resulting in an overload of the control plane and in the possible impairment of large parts of the network. Furthermore we considered AVANT-GUARD [17], which is, to the best of our knowledge, the only currently proposed solution against control plane saturation attack. We showed that in its original design, subtle points were not taken into consideration, opening critical system vulnerabilities. To this aim we proposed LineSwitch, a solution based on probability and blacklisting which offers both resiliency against SYN flooding-based control plane saturation attacks and protection from buffer saturation vulnerabilities. Our preliminary evaluation demonstrates that LineSwitch imposes a negligible overhead, which can be dynamically adjusted to fit the network needs, while successfully defending the OpenFlow switch and controller from attacks that can potentially disrupt the functionality of the network.

# 8. REFERENCES


[1] OpenFlow Switch specificatio, v.1.3.4. `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.4.pdf`.

[2] OpenFlow whitepaper. `https://www.opennetworking.org/sdn-resources/sdn-library/whitepapers`.

[3] Transmission Control Protocol. RFC 793, IETF, September 1981.

[4] TCP SYN Flooding Attacks and Common Mitigations. RFC 4987, IETF, August 2007.

[5] Defending against Sequence Number Attacks. RFC 6528, IETF, February 2012.

[6] K. Benton, L. J. Camp, and C. Small. OpenFlow Vulnerability Assessment. HotSDN '13, pages 151–152, 2013.

[7] D. J. Bernstein. SYN Cookies. `http://cr.yp.to/syncookies.html`.

[8] W. Haopei, X. Lei, and G. Guofei. OF-GUARD: A DoS Attack Prevention Extension in Software-Defined Networks. In *USENIX Open Network Summit*, 2014.

[9] R. Kloti, V. Kotronis, and P. Smith. Openflow: A security analysis. ICNP '13, pages 1–6, 2013.

[10] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards Secure and Dependable Software-defined Networks. HotSDN '13, pages 55–60, 2013.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Computer Communications Review*, 38(2):69–74, Mar. 2008.

[12] Mininet. `http://mininet.org/`.

[13] R. T. Morris. A Weakness in the 4.2BSD Unix TCP/IP Software, 1985.

[14] OpenFlow Software Switch. `http://yuba.stanford.edu/git/gitweb.cgi?p=openflow.git;a=summary`.

[15] T. Peng, C. Leckie, and K. Ramamohanarao. Survey of Network-based Defense Mechanisms Countering the DoS and DDoS Problems. *ACM Computing Surveys*, 39(1), 2007.

[16] POX. `http://www.noxrepo.org/pox/about-pox/`.

[17] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-defined Networks. CCS '13, pages 413–424, 2013.