LineSwitch: Tackling Control Plane Saturation Attacks in Software-Defined Networking

M. Ambrosin, *Student Member, IEEE*, M. Conti, *Senior Member, IEEE*,
F. De Gaspari, R. Poovendran, *Fellow, IEEE*

***Abstract*—Software Defined Networking (SDN) is a new networking paradigm that in recent years has revolutionized network architectures. At its core, SDN separates the data plane, which provides data forwarding functionalities, and the control plane, which implements the network control logic. The separation of these two components provides a virtually centralized point of control in the network, and at the same time abstracts the complexity of the underlying physical infrastructure. Unfortunately, while promising, the SDN approach also introduces new attacks and vulnerabilities. Indeed, previous research shows that, under certain traffic conditions, the required communication between the control and data plane can result in a bottleneck. An attacker can exploit this limitation to mount a new, network-wide, type of Denial of Service attack, known as the *control plane saturation attack*. This paper presents LineSwitch, an efficient and effective data plane solution to tackle the control plane saturation attack. LineSwitch employs probabilistic proxying and blacklisting of network traffic to prevent the attack from reaching the control plane, and thus preserve network functionality. We implemented LineSwitch as an extension of the reference SDN implementation, OpenFlow, and run a thorough set of experiments under different traffic and attack scenarios. We compared LineSwitch to the state-of-the-art, and we show that it provides at the same time the same level of protection against the control plane saturation attack, and a reduced time overhead by up to 30%.**

I. Introduction

Software Defined Networking (SDN) is a recently proposed paradigm that aims at simplifying the management of networking infrastructures. SDN proposes a marked shift from the current network infrastructure by decoupling the network logic layer, called the *control plane*, and the data layer, called the *data plane*, into separate entities. Such separation provides several benefits: on one hand, it allows the development of virtually centralized and directly programmable network control systems; on the other hand, it reduces the complexity of network devices by providing to management applications an abstract representation of the network.

The work of M. 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.

M. Ambrosin is with the Department of Mathematics, University of Padua, Padua 35131, Italy, and also with the Department of Electrical Engineering, University of Washington, Seattle, WA 98105 USA (e-mail: ambrosin@math.unipd.it).

M. Conti is with the Department of Mathematics, University of Padua, Padua 35131, Italy (e-mail: conti@math.unipd.it).

F. De Gaspari is with the Department of Informatics, Sapienza University of Rome, Rome 00198, Italy (e-mail: degaspari@di.uniroma1.it).

R. Poovendran is with the Department of Electrical Engineering, University of Washington, Seattle, WA 98105 USA (e-mail: rp3@uw.edu).

Among the existing instantiations of the SDN paradigm, OpenFlow [1], [2] is the most widely adopted. OpenFlow defines the standard API for communication between the data plane and the control plane; moreover, it introduces the concept of *flows*, which identify the network traffic [2], and the idea of *flow tables*. Each OpenFlow-enabled network device, called *OpenFlow switch*, needs to maintain a set of flow tables, organized in a pipeline. These flow tables define which actions the OpenFlow switch will perform on a given network flow. The flow table pipeline is traversed every time a packet is received, allowing for multiple rules from different tables to be applied to a given network flow. The control plane can program the flow tables of the OpenFlow switches either proactively or reactively. When flow rules are installed reactively, each time an OpenFlow switch receives an inbound flow for which it has no matching rule, it will request the installation of a new rule to the control plane [3]. Unfortunately, while this behavior enables a flexible network management, it also introduces a potential bottleneck in the communication channel between the data and control plane.

The extensive communication between control and data plane results not only in poor scalability under high traffic load in normal circumstances, but also introduces a serious vulnerability which can be exploited to overload the controller with flow requests. This attack is called *control plane saturation* [4], [5], and can be easily performed, for example, through SYN flooding [6]. Since OpenFlow switches require the control plane to provide rules for new network flows, an attacker successfully incapacitating a controller is effectively incapacitating the switch connected to such controller. This vulnerability is further exacerbated by the fact that, in general, a controller manages more than a single switch [4], [7], [5], [8], potentially hindering a large part of the network.

Recently, Shin et al. [4] proposed AVANT-GUARD, a countermeasure against the control plane saturation attack. AVANT-GUARD introduces a new module into the OpenFlow switch, called *connection migration* module, which protects the switch from saturation attacks performed by SYN flooding, while at the same time being transparent to the end hosts. The connection migration module implements a SYN proxy mechanism in each OpenFlow switch; every time an inbound TCP connection is received, the switch acts as a proxy during the initial handshake, instead of immediately contacting the controller. Unfortunately, while this protocol effectively shields the controller from possible floods in the general case, it also introduces new unintended vulnerabilities and possible limitations. Indeed, we show that, in order to run AVANT-GUARD, an OpenFlow switch needs to maintain some state in memory for each connection, leading to a new type of Denial of Service attack we named *buffer saturation attack*. Furthermore, the transparency provided by AVANT-GUARD with respect to the end hosts comes with a cost in terms of maximum number of connections that a switch can proxy, which is limited to the number of available TCP port numbers.

Our Contribution: In this paper, which is an extended version of the work in [9], 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, to 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 ran a thorough set of experiments, and compared our solution against the current state-of-the art under three different traffic scenarios: under normal traffic, under a control plane saturation attack, and under a buffer saturation attack. The results of our experiments confirm that LineSwitch effectively protects against the control plane saturation attack, while reducing the introduced timing overhead by 30%, when compared to current state-of-the-art solutions.
- We further propose additional modifications to LineSwitch, which allow for a more refined use of proxying based on incoming packets rate at the controller level, and address possible attacks on legit clients.
- We discuss how the SDN network performs in the three different scenarios we considered in our evaluation, when either the official OpenFlow, the state-of-the-art solution, or LineSwitch is employed.

Organization: The rest of this paper is organized as follows. In Section II, we provide some background knowledge on SDN. In Section III, we revise some related work in the area of DoS attacks and defense in Software Defined Networking. Additionally, we provide an overview of AVANT-GUARD, the current state-of-the-art solution against the control plane saturation attack. In Section IV, we analyze the limitations of the current state-of-the-art, and propose a new potential attack we refer to as the buffer saturation attack. Section V describes LineSwitch, our countermeasure against the control plane saturation attack, while we experimentally evaluate its effectiveness in Section VI. In Section VII, we propose possible improved variants of LineSwitch through a small extension of the OpenFlow protocol. In Section VIII, we discuss our solution at a higher level, and provide a comparison with the state-of-the-art and the original OpenFlow. Finally, in Section IX, we draw our conclusions.

II. 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 cornerstone of SDN and its main difference compared to today's network architecture is the decoupling of 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.

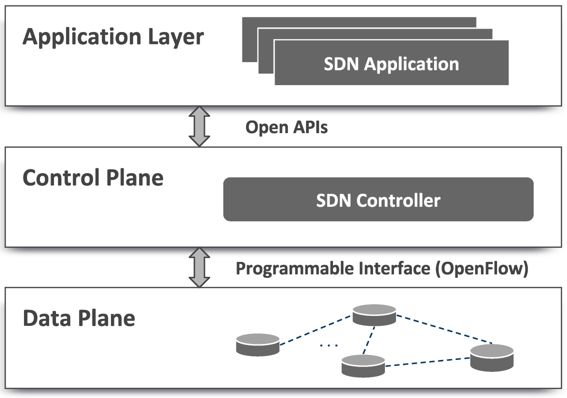


Fig. 1: SDN Architecture.

In the SDN model the control plane acts as a middleware, providing an interface to program the data plane behavior to third party applications. This model allows applications to interact with the data plane independently from the physical network devices employed, as well as abstracts all the complexity related to network infrastructure, and data-to-control plane communication. 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: OpenFlow [2] is the reference implementation of the SDN paradigm. It defines a standard communication interface between the control plane and the data plane. The OpenFlow specification defines that packet routing is performed on the basis of traffic *flows*. Each network device, which is called OpenFlow switch, needs to maintain a set of *flow tables*. Flow tables contain rules installed by the control plane, instructing the switch on how to handle incoming packets. Additionally, each OpenFlow switch also maintains a communication channel to an external controller in the control plane. Figure 2 presents the logical structure of an OpenFlow switch.

The control plane can program the physical devices through a series of *flow rules* [3], 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 OpenFlow 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. Flow rules can be either pre-installed (proactively), or installed on-demand (reactively) by the controller, allowing a switch-driven installation of new flow rules only when they are effectively required [2]. In the latter case, if an OpenFlow switch does not have any matching rule for a new incoming flow, it forwards the corresponding packet header to the controller (in a PacketIn packet), which at this point can install a new rule for that flow into the switch [3]. 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 [3].

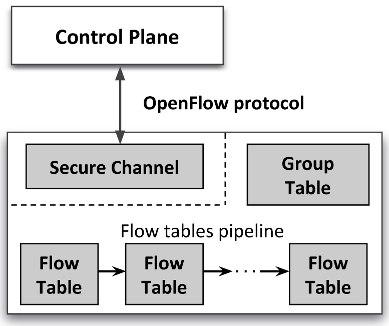


Fig. 2: OpenFlow Switch Architecture.

III. Related Work

In the recent years, SDN has become a quite common networking architecture, as well as a popular research subject. In this section, we provide a brief overview of the main research studies related to Denial of Service (DoS) attacks in SDN. In [6], Peng et al. provided a first feasibility study for Denial of Service attacks on SDN control plane. In [10], Kreutz et al. analyzed SDN architecture, identifying critical aspects and possible new attack vectors, while Kloti et al. [7] assessed some vulnerabilities that affects OpenFlow [2], using the STRIDE vulnerability modeling technique.The analysis in [7] highlighted two main vulnerabilities, 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 [11], Mehdi et al. provided a system for network traffic analysis at the control plane level. In their work, the authors applied anomaly detection techniques to expose several types of attacks, such as scanning worm infections. More recently, Li et al. proposed DrawBridge [12], a system based on SDN architecture that enables end hosts to use their knowledge about desired traffic, to improve ISPs traffic engineering for DDoS detection. In [13], Braga et al. proposed a lightweight DDoS detection system for OpenFlow/NOX [14] based on traffic flow analysis. The system classifies the traffic based on flow features, such as the average number of packets per flow. Statistics are collected by the control plane querying each OpenFlow switch at specific time intervals. In general, all of the above solutions act as high level applications deployed at the control plane level. Therefore, they do not address problems related to data plane and control plane interaction, nor offer protection from link saturation attacks.

In [15], Wang et al. propose Scotch, a solution that allows the network's control plane to scale-up when under high load. The authors identify the SDN bottleneck in the OpenFlow agent inside the switches, i.e., the control part of the OpenFlow switch responsible of interfacing with the control plane. Their proposed solution leverages the high control plane capacity given by a pool of Open vSwitch devices [16], as well as the high data plane capacity of commodity switches, to improve the scalability of the network under DoS attack. However, their solution addresses only the situation in which the network's bottleneck is represented by the OpenFlow Agent within the switches, and implicitly assume the remote controller to be able to handle the high control traffic load. Consequently, we believe that [15] could benefit from the use of LineSwitch on the physical switches.

In [17], Mekky et al. extend the SDN architecture by allowing packet forwarding based on application level information. Their solution allows to limit the communication between the data plane and the control plane, and to implement a DoS protection mechanics similar to the one proposed in [4].

Kotani et al. [18] propose a packet-level filtering approach to limit the data-to-control plane communication. In their solution, switches maintain two additional data structures for low-priority packets, i.e., Pending Rule Table (PRT) and Pending Flow Table (PFT). The PFT is populated a priori by the control plane, and contains entries indicating which attributes matches with low-priority packets; the PFT keeps track of the (low-priority) packets for which a PacketIn message has been already sent to the control plane. A packet is added to the PFT if and only if it matches with an entry in the Pending Rule Table, and if there is no matching entry in the PFT. However, their proposed defense mechanism is easily avoidable by an attacker by modifying the header of the flooding packets (something that would be done anyway in case of an attack, at least for the IP field, to avoid identification). In this way, all the packets will be forwarded to the controller without filtering, effectively circumventing the proposed defense mechanism. Moreover, selected fields of the header will be stored in the PFT for each packet, eventually resulting in the saturation of the table. As the authors suggest, at this point the switch can either: (1) arbitrarily evict entries from the PFT, which would not be very useful, as it would be saturated again shortly after; or (2) temporarily revert to the standard OpenFlow protocol, therefore exposing the controller to DoS attacks.

Finally, in [4], Shin et al. addresses the architectural flows of the original OpenFlow protocol by altering the flow management at the data plane level [5], [4]. In particular, they tried to solve the bottleneck introduced by the communication between the data plane and the control plane, that can be exploited to mount a control plane saturation attack. In their work, the authors focused on the control plane saturation attack based on SYN flooding. The authors applied SYN Cookie and middle-box concepts [19] to OpenFlow switches, changing significantly the way data flows are handled with respect to the standard OpenFlow. To the best of our knowledge, AVANT-GUARD [4] represents the state-of-the-art data plane level solution for tackling the control plane saturation attack in SDN. For this reason, we briefly describe this solution in Section III-A. We will further compare it against our solution when evaluating our proposal (see Section VI).

A. *Avant-Guard*

AVANT-GUARD [4] modifies the standard OpenFlow protocol and adds two extensions: (1) a *Connection Migration* module, which mitigates the efficacy of control plane saturation attacks based on SYN flooding by proxying the incoming TCP handshakes, 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 IV we will provide a security analysis of the connection migration module.

1) Basic Connection Migration: As introduced in Section II, whenever an OpenFlow switch receives an inbound network flow for which it has no flow rule, it will forward the packet to the control plane. This behavior applies to SYN packets too: indeed, for each received SYN packet not matching any flow rule in the pipeline, an OpenFlow switch will generate a flow request for the controller, asking for a new rule to be installed. The proposed connection migration module of AVANT-GUARD addresses this problem at the data plane level, by having the OpenFlow switch act as a SYN proxy. This process is composed of four phases (see Figure 3):

1) *Classification phase.* Whenever a SYN packet from a new network flow is received by an OpenFlow switch, (action (1) in Figure 3), the switch acts as a proxy and engages the client in a stateless TCP handshake by using SYN Cookies (actions (2) and (3) in Figure 3), instead of immediately contacting the control plane.
2) *Report phase.* If the client completes the TCP handshake, the switch then forwards the new flow to the controller (action (4) in Figure 3) and wait for a new rule that defines how it should be handled (action (5) in Figure 3).
3) *Migration phase.* Upon receiving permission for the migration, the switch initiates a TCP handshake with the destination host (actions (6), (7) and (8) in Figure 3). The OpenFlow switch further reports the result of the handshake to the control plane (actions (9) and (10) in Figure 3).
4) *Relay phase.* If the handshake is successful, the switch relays subsequent messages between the client and destination.

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 [20]. Since the SYN Cookie algorithm does not need to maintain state for connection requests, there is no need for storing information in the OpenFlow switch for failed TCP connections.

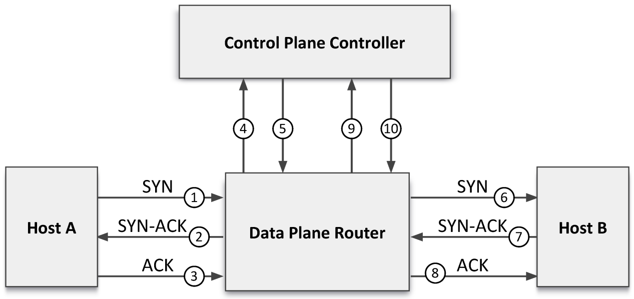


Fig. 3: Connection Migration technique of AVANT-GUARD [4].

2) Delayed Connection Migration: Generally, an adversary can potentially infer the use of the connection migration protocol through analysis of Round Trip Time (RTT) of SYN packets. This information can be used to alter the attack strategy, flooding the data plane with complete TCP handshakes instead of single SYN packets, hence forcing the switch to forward each of them to the control plane. To solve this issue, AVANT-GUARD introduces a modification to the connection migration module, intended for protocols in which the initiator of a connection is expected to send the first data packet: this process is named *delayed connection migration*. In particular, delayed connection migration extends the basic connection migration protocol adding a further requirement for the switch to forward a new flow to the control plane: after the initiator successfully completes the TCP handshake, the data plane will postpone the packet forwarding to the control plane, until the first valid data packet is received from the initiator of the connection. Figure 4 presents a high-level view of the modified connection migration module. This additional requirements aims at granting a better classification stage, where the reception of a valid data packet ensures that the connection is indeed valid and not part of a flooding attack.

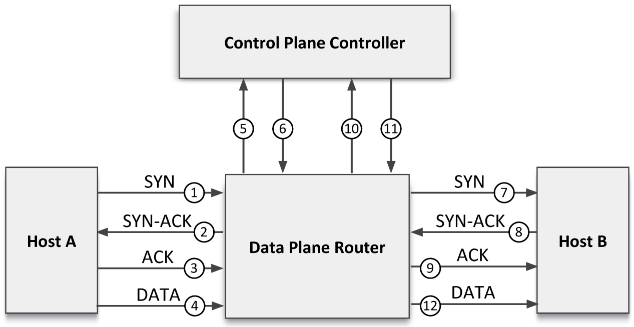


Fig. 4: Delayed Connection Migration technique of AVANT-GUARD [4].

IV. Limitations of the Connection Migration module

The connection migration module of Avant-Guard [4] 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 OpenFlow switch:

1) SYN proxying introduces the need to store timestamp and sequence number-related state throughout the duration of the connection, as well as source IP and port. This constraint can easily be exploited by an attacker to incapacitate an OpenFlow switch which implements the connection migration module (see Section IV-A).
2) Using SYN proxy techniques inherently limits the number of connections that can be forwarded. In particular, the maximum number of connection forwarded to a specific $\{IP, port\}$ pair is 64512 (see Section IV-B).

In this section we provide an in-depth analysis and the scheme of a possible attack for each of the above points, highlighting the vulnerabilities and limitations introduced by the connection migration module of Avant-Guard. Finally, in Section IV-C we discuss the consequences of the need for transparency during the migration, while in Section IV-D we discuss the hidden ramifications of using proxy in the inner nodes of a TCP connection.

A. Proxying Requires State

Let us consider a situation where a host A initiates a TCP connection with another host B. Let R be an intermediate OpenFlow switch. According to the TCP specifications [21], during the TCP handshake the Initial Sequence Numbers (ISNs) for the connection are set as follows:

- Host A sends a SYN packet to host B with a sequence number ISN_A.
- Host B replies with a SYN-ACK packet; the acknowledgment number will be set to $ISN_A + 1$ and the SYN sequence number will be set to an arbitrary value ISN_B.
- Host A replies with an ACK packet with the acknowledgment number $ISN_B + 1$.

After the completion of the TCP handshake, all subsequent packets from host A to host B will have sequence numbers computed starting from $ISN_A + 1$, and adding one for each byte in the payload. In the same way, all the packets from host B will have sequence numbers computed starting from $ISN_B + 1$.

When using the connection migration module of Avant-Guard, the OpenFlow switch R proxies the TCP connection between A and B, introducing a new intermediary handshake with host A that otherwise would not happen. As a result, after the first handshake is completed, host A will assume to have an established connection with host B. Therefore, host A will expect all the subsequent packets form B to have sequence numbers starting from $ISN_R{+}1$, where ISN_R is the sequence number of the SYN-ACK packet sent from the OpenFlow switch R. Moreover, given that for security reasons [22], [23] each ISN is computed in a non-predictable way, it is not possible for switch R to compute the exact sequence number that B will use in the TCP connection. As a consequence, the sequence number ISN_R used by the OpenFlow switch will differ from ISN_B. In particular, the actions computed by A, B and the OpenFlow switch R are the following:

- Host A sends a SYN packet to host B with a sequence number ISN_A.
- Switch R intercepts the incoming packet, and replies to host A with a SYN-ACK packet with a spoofed address, i.e., using host B address. The ACK number will be $ISN_A + 1$ and the sequence number will be a random number ISN_R.
- Host A replies with an ACK packet with the acknowledgment number $ISN_R + 1$. From now on, A will expect incoming packets from host B to have a sequence number $ISN'_R = ISN_R + payload\_bytes$, where $payload\_bytes$ is the number of payload bytes received up to a given moment.
- Upon receiving the permission to migrate the connection, switch R will start an handshake with host B by sending a SYN packet with sequence number ISN_A. It is important to note that at this step, switch R can not use the IP address of host A when sending the connection request. Indeed, there is no guarantee that the reply from host B will follow the same path through switch R on the way back. Therefore, during the whole TCP connection, the connection identifier stored by host B will be $\{IP_R, port_R, IP_B, port_B\}$, and consequently, host B will not have any knowledge about host A.
- Host B replies with a SYN-ACK packet containing an ACK number $ISN_A + 1$ and a sequence number ISN_B.
- Switch R finalizes the connection sending an ACK with number $ISN_B + 1$.

In order to maintain the existence of the proxy transparent for host A, a *sequence number translation* from host B to host A is necessary. Conversely, an acknowledgment translation is required in the opposite direction. As a result, switch R must change the sequence numbers of all the packets coming from host B, and the acknowledgment number of all the packets sent from host A, in order to match what the two end hosts expect. We underline that, in order for this conversion to take place, some state information is needed for each active connection. Moreover, let us consider a scenario where there are two separate clients, C and D, trying to connect to host B, both from source port $port_{C,D}$ to port $port_B$ through switch R. In this case, since R uses its IP address to migrate the connection, it can not use $port_{C,D}$ to proxy both connections in the migration process. Otherwise, it would not be possible for R to map each packet received from B, which would be addressed to R's IP on port $port_{C,D}$, to one of the two clients C or D. Therefore, for each host connecting to the same $\{IP, port\}$ pair, the OpenFlow switch needs to use a distinct port number to migrate the connection in order to be able to match the response packets to the correct host on the way back. Consequently, for each connection the OpenFlow switch needs to translate several fields of both the TCP and IP headers of each packet, and 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 δ_{seq} is the difference between the ISN used by the router and the ISN used by the destination host.

It is clear that, while proxying each TCP connection allows R to prevent malicious SYN packets from being forwarded to the controller, it also forces a connection-long storage of information, for each open connection. This behavior gives rise to the same type of vulnerability SYN flooding exploits, i.e., gives an adversary an easy mean to saturate the target OpenFlow switch buffer.

Buffer Saturation Attack: Given that the OpenFlow switch needs a translation table with one entry for each connection, it is a prime target for a buffer saturation attack. The attack itself is simple: the attacker just needs to open several complete TCP connections through the target OpenFlow 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.

B. TCP Ports and Limit on Connections Number

As we already stated in Section IV-A, 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 limitation: in the scenario where multiple clients connect to a given destination on the same port (e.g., $\{IP_B, port_B\}$) through switch R, the latter will be forced to use a different port to migrate each 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.

1) DoS through Port Saturation: This limitation is clearly a very important drawback of the system, and can easily be exploited by a malicious attacker. Indeed, it would be easy to target a given service by opening a series of long-lasting connections through the OpenFlow 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 an extremely 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 OpenFlow 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 all the ports have been used with a given address, it will switch to a new address and will be 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.

C. Connection Migration Transparency

The connection migration module of AVANT-GUARD [4] is transparent to both the source and destination host. Consider again a host A communicating with another host B, through an OpenFlow switch R. When host A completes the connection with R in the first step of connection migration, it has no knowledge of the fact that it is not connected to the destination host B. As a consequence, host A might start sending data as soon as the connection is completed. At this point, a switch executing the delayed connection migration module of AVANT-GUARD (see Section III-A) has only two possibilities: (1) discard all the incoming packets, until the connection has been migrated; or (2) buffer all the incoming packets, and further forward them as soon as the connection migration is completed. Both options have their pros and cons.

In the first case, discarding all incoming data packets will result in a higher overhead in the first stage of the connection, since the packets will be retransmitted by the sender only after TCP timeout expires. Moreover, if the duration of the connection migration spawns over more than one TCP retransmission (e.g., because of a slow or distant destination host), the overhead will increase exponentially due to TCP's binary exponential backoff mechanism [24]. If the connection is a long-lived one, this initial overhead might be negligible over its duration. However, if the connection is short-lived, and possibly part of a set of other short-lived connections, the cumulative overhead can indeed become considerable.

In the second case, buffering incoming data packets while the migration is in progress will certainly result in a lower overhead, since such packets would be forwarded to destination as soon as the migration is completed, but unfortunately, it would also give an attacker an easy way of flooding the switch and force it to buffer all the packets. However, once the buffer is saturated, the switch could simply revert back to discarding all data packets, so the impact of such an attack would be somewhat limited. Nonetheless, such buffering behavior could be used to consume computational and storage resources on the switch.

In our implementation of the connection migration module, we opted for discarding incoming data packets in order to minimize the surface for the attacker as much as possible. This design choice does not in any way influence the results of the experiments presented in Section VI, as the code used for the SYN proxy is the same for both our implementations of AVANT-GUARD and LineSwitch (see Section V).

D. Consequences of Breaking End-to-End Semantics

All the vulnerabilities we identified above should be results of breaking TCP end-to-end semantics. In fact, in this way we force intermediate nodes in the path to actively take part in the communication which was designed to be purely end-based. Beside what we already argued, this can have deeper and more subtle consequences on higher level protocols that rely on the standard TCP design, and even on some aspects of TCP itself.

1) Higher-Level Protocols and Applications: A problem arises when we consider higher-level protocols and applications relying on the standard TCP behavior. In particular, it is common for higher layers of the network stack to access lower layer information, e.g., information about IP addresses might be used for location based services. Unfortunately, when proxying mechanisms, such as the one used in AVANT-GUARD, are in place, this information will not be valid, since modified by the OpenFlow switch. As a consequence, applications and protocols relying on the correctness of lower layers information, will either not work or return unexpected results.

2) TCP Setup: The TCP handshake is designed to allow for the correct setup of a connection, which includes the exchange of the capabilities (in TCP terms, available options [25]) of the two end hosts. Since it is not possible for an OpenFlow switch to have any knowledge about the capabilities of the destination TCP stack, when it completes the connection with the source, it can not allow any option which is not required by the standard (namely, only the MSS option [21]). This precludes the setup of all the TCP extensions for high performance [26] as well as the SACK option [25], which are fundamentals in order to achieve efficient use of the available bandwidth in large-bandwidth connections. Moreover, for the MSS option the switch is forced to use a conservatively small value in order to have a reasonably high probability that the destination host will be able to manage such segment size. Indeed, if the MSS is greater than what the destination host can manage, the switch will be forced to break up packets that are too big in multiple fragments (each with a new set of headers), introducing additional delays.

3) Path Instability Issues: Since the migration is done with the IP address of the OpenFlow switch that is proxying the connection, the destination host will have no notion of the real initiator of the connection. Consider the case in which the path between the two end hosts A and B changes, and does not pass through the proxy OpenFlow switch anymore, e.g., because of normal congestion, or as a result of a targeted attack. In this scenario, as soon as the destination host B receives a packet from A, this time with IP address and port of A, it will not be able to match it with any existing flow. Therefore, B will respond with a RST packet, abruptly terminating the connection and loosing any data in transit.

The above behavior could theoretically be exploited by a malicious user to target known OpenFlow switches executing the connection migration protocol. Indeed, an attacker could overload and incapacitate such switches to force network flows to change path, this way effectively destroying all open connections passing through that node. Moreover, the same problem arises with fragmented packets: if a packet is too big to be sent as a single TCP/IP datagram, the source host, or the routers along the path, might separate it into multiple fragments. In general, there is no guarantee that all this fragments will be routed along the same path. Therefore, if one or more fragments do not pass through the proxy OpenFlow switch, the destination host will not recognize them, and possibly terminate the original connection at the source host (with a RST packet). Finally, while it is possible for the control plane to mitigate this limitation by keeping the routing of TCP flows static inside its network, this would highly reduce the effectiveness of existing load balancing techniques. Moreover, the control plane can not influence external subnetworks. Therefore, if the packets from a migrated flow are routed through a different subnetwork by external nodes, the connection will not pass through the proxy OpenFlow switch and will be consequently reset by the destination host.

V. 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, any defense mechanism against the control plane saturation attack needs to reduce the use of proxying as much as possible, while at the same time retaining the beneficial effects proxying offers. As noted before, one of the main strengths of SYN flooding is the possibility to launch the attack with spoofed source IP addresses. If we were able to remove this possibility, it would be possible to identify flooding attempts and to discard all incoming packets from the offending IP.

To this end we propose LineSwitch, an OpenFlow module deployed on edge OpenFlow switches. LineSwitch proxies all incoming TCP connections from a given IP until one is completed, while subsequent connections are proxied only with a very small probability P_p. Figure 5 presents a high-level representation of the logical flow of LineSwitch.

LineSwitch can protect the control plane and the switch, even in presence of an attacker Adv with knowledge about the proxy mechanism in use. Indeed, it would be possible for Adv 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, Adv would be forced to use its real IP address, IP_{Adv} in all the packets, to ensure they will be forwarded to the OpenFlow pipeline. This requirement stems from the fact that the first connection from any address is proxied by the switch, and then forwarded through the OpenFlow pipeline only if the handshake is completed. Since an attacker using a spoofed IP address is unable to complete the handshake, all his requests are immediately discarded by LineSwitch. Moreover, since the effectiveness of the SYN flooding attack is based on a high throughput, once Adv is forced to use its real IP address for the flooding, the OpenFlow switch will quickly proxy one of the packets, thus detecting the attack. Then, the OpenFlow switch can blacklist the IP address of host Adv, IP_{Adv}, for $T \times 2^{count_{IP_{Adv}}}$ seconds, where $count_{IP_{Adv}}$ indicates the number of times IP_{Adv} did not complete a connection, and T represents a default time value. The time required for the detection of the attack is related to the probability of proxying, which is defined by the network administrator and should be tailored to the specific needs and history of the network under consideration for maximum efficacy. We further discuss what realistic values can be used in Section VI-D.

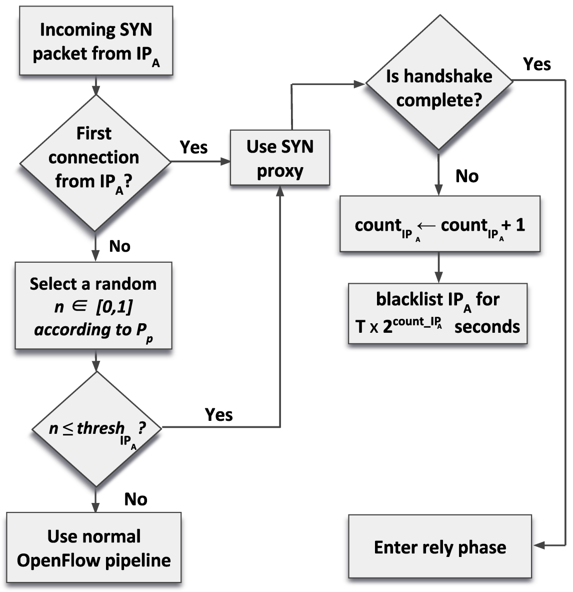

Incoming SYN packet from IP_A
First connection from IP_A?
Yes
No
Use SYN proxy
Is handshake complete?
Yes
No
count_IP_A ← count_IP_A + 1
blacklist IP_A for T x 2^count_IP_A seconds
Select a random n ∈ [0,1] according to P_p
n ≤ thresh_IP_A ?
Yes
No
Use normal OpenFlow pipeline
Enter rely phase

Fig. 5: LineSwitch logic flow.

LineSwitch effectively blocks all packets with spoofed IP addresses at the data plane, while at the same time penalizes clients performing SYN flooding with non-spoofed IPs (after they first established a complete TCP connection) with exponentially increasing blacklist periods of time.

A. Key Advantages

In this section we describe the various improvements that our approach grants with respect to the problems identified on the current state-of-the-art in Section IV.

1) Higher Resiliency to Buffer Saturation: The first clear advantage of our approach is that it will drastically reduce the memory usage due to the required translation for each connection, thus offering high resilience against the proposed buffer saturation attack. 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 almost 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.

While it would be prohibitive for any realistic attacker to keep track of such a huge amount of connections, this is not needed for a buffer saturation attack. Indeed, the attacker can simply send SYN packets as quickly as possible, and blindly reply to any SYN-ACK packet received from the OpenFlow switch, without checking if it is the answer to a previous sent SYN packet or not. Due to this peculiarity, there is no need for the attacker to store the state related to the opened connections, therefore allowing him to reach the theoretical maximum limit (albeit in a long time). Instead, with LineSwitch the number of entries the switch needs to store under attack, is almost 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. Moreover, even if an attacker were able to generate a huge amount of complete connections in a relatively small time, it would be sufficient to dynamically adjust the migration probability P_p for the offending hosts, as soon as a buffer saturation attack attempt is detected. Finally, when an attack is detected and the OpenFlow switch is under high distress (e.g., the buffer use crosses a given critical threshold, which will happen only after an extremely long time, as we will show in Section VI-C), it is fair to consider incomplete connections whose duration exceeds a given value to be malicious, and therefore they can be reset [27]. In the statistically unlikely scenario that the connection was a valid one, the originator will simply restart it with little harm done (and it will not be proxied again with high probability). What would constitute a good method to detect such attacks is left as future work.

2) Reduced Use of Proxy: As discussed in Section IV-B and Section IV-D, 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. To this extent, LineSwitch proxies only the first connection from a given host (i.e., an IP address), while subsequent incoming connections from the same IP address are proxied only with probability P_p. Since LineSwitch is effective even with small migration probability value P_p (see Section VI), in most cases the normal network flow is preserved, mitigating intrinsic problems of proxying such as routing path instability, limited maximum number of migrated connections, and allowing for all possible TCP options that proxying and SYN Cookies do not allow for (see Section IV-B and Section IV-D). 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 VI).

3) Reduced Overhead: While according to its specifications AVANT-GUARD introduces a negligible overhead retrieving a web page [4], we will show in Section VI-A that, after a more careful analysis of the AVANT-GUARD system (namely, that it cannot use the original IP and port, as discussed in Section IV-A) this claim of low overhead changes considerably. Indeed, the connection migration protocol introduces a remarkable amount of overhead. Moreover, as our experimental evaluation will show, LineSwitch fares considerably better under this point of view both under normal circumstances and under SYN flooding attack.

VI. EVALUATION

In order to assess the feasibility and effectiveness of our solution in different scenarios, we designed and performed a thorough set of experiments. In particular, we implemented our solution in the reference OpenFlow software switch [28], and evaluated it against the standard OpenFlow implementation. Moreover, we compared it to the state-of-the-art solution to tackle the control plane saturation attack generated by SYN flooding, i.e., AVANT-GUARD [4], adapted to include the necessary modifications pointed out in this paper (see Section IV). More in detail, AVANT-GUARD does not require any setup, since there are no configurable parameters. All the parameters related to the conducted experiments are kept constant when comparing the different solutions (standard OpenFlow, AVANT-GUARD and LineSwitch) and are described in detail in this section. The specific modifications to AVANT-GUARD, when compared with [4], are as follows:

1) For each completed connection, our AVANT-GUARD implementation stores $\delta_{timestamp}$ and δ_{seq} number, source IP IP_{src}, and source port $port_{src}$ (see Section IV).
2) For each packet in transit, our AVANT-GUARD implementation replaces all the above fields to maintain the connection state consistent (see Section IV).

The specific implementation of these steps is shared between AVANT-GUARD and LineSwitch (which requires them too when proxying is used). For this reason, our experimental results are not influenced by our specific implementation. We ran all our experiments using the Mininet network simulator [29] in a virtual machine.

Figure 6 presents the setup of our simulation, which includes two client hosts, and an HTTP server, connected to an OpenFlow switch running the reference OpenFlow software switch [28], and a local controller (running the POX controller, `l3_learning` module [30]).

The only meaningful parameters for a buffer saturation attack are the rate at which connections are completed, which is influenced only by the total bandwidth available to the attacker, and the size of the buffer the OpenFlow switch uses to store the required information. Therefore, we did not deem necessary using more complex network layouts in our experiments. 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.

In what follows, we analyze and compare the performance of OpenFlow, AVANT-GUARD and LineSwitch both in a regular use-case scenario (Section VI-A), under SYN flooding attack (Section VI-B) and under buffer saturation attack (Section VI-C).

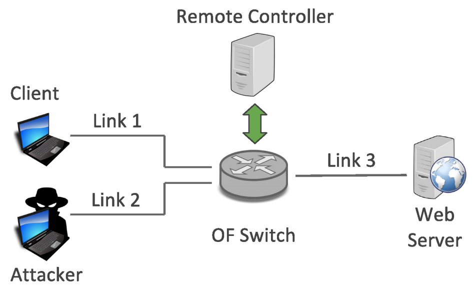


Fig. 6: Experimental Setup.

A. Regular Traffic Scenario

As a first test, we simulated the system in Figure 6 under a regular traffic scenario, i.e., a client performs an HTTP request to a web server through an OpenFlow switch. For this experiment, Link 1, Link 2 and Link 3 of Figure 6 are setup with a bandwidth of 10 Mbps and a Round Trip Time (RTT) between nodes of 80 ms. Moreover, the probability for LineSwitch to migrate a connection of an already known host is set to $P_p = 0.05$ and maintained static. We sampled the time required to retrieve a web page (of size 1 KByte) with both OpenFlow, AVANT-GUARD and LineSwitch, and computed the average over 500 separate runs. The results of our simulation are displayed in Table I. As we can see, our implementation of AVANT-GUARD, which takes into account the practical considerations we discussed in Section IV-A, requires considerably more time compared to what originally estimated in [4].

The main difference seems to arise from the fact that the analysis in [4] did not consider the need for an OpenFlow switch to modify all the packets of a connection passing through it.

Implementation	**Avg. Time**	**Std. Dev.**	**Overhead**
OpenFlow	0.404 s	0.001 s	0.00%
AVANT-GUARD	0.573 s	0.014 s	41.83%
LineSwitch	0.435 s	0.030 s	7.67%

TABLE I: Average web page retrieval time/success rate, in a regular traffic scenario.

Note that, given the unavailability of the original source code, we needed to provide our own prototype implementation of AVANT-GUARD, which is not optimized. However, in general the operations required by the connection migration, coupled with the lookups needed to retrieve the information pertaining to the specific TCP flow, result in a high total overhead for the connection. In contrast, on average LineSwitch introduces just a small overhead per connection. Moreover, it can be further tuned to the specific needs of the environment through adjustments the P_p parameter, as this will allow to filter a higher or smaller percentage of the incoming connection requests. Finally, it is worth noting that the connection migration code used in our proposal is the same as the one adopted for the implementation of AVANT-GUARD, therefore any improvement on the latter will consequently reflect on the results of our proposal too.

B. SYN Flooding Scenario

In this test we simulated the behavior of the system under the control plane saturation attack via SYN flooding, and assessed the state of the various components. In our experiment, we measured the average retrieval time of a web page, and the corresponding success rate. In particular, in order to have a better understanding of the impact of such attack on the system at different attack rates, we ran the experiments varying the bandwidth of Link 2, i.e., the attacker's link in Figure 6, while keeping Link 1 and Link 3 at 10 Mbps. This allows us to ascertain the behavior of the different solutions under increasing rates of attack. Table II and Table III report the results of our experiments, along with the corresponding bandwidth values for Link 2.

In all our tests, the flooding attack was started 20 s before the measurements were performed; this was done in order to obtain more meaningful results, simulating a background SYN flooding attack already in progress while trying to retrieve a web page.

Implementation	**Avg. Time**	**Success Rate**	**Overhead**
OpenFlow	2.41 s	100%	495.54%
AVANT-GUARD	0.56 s	100%	39.85%
LineSwitch	0.41 s	100%	1.73%

TABLE II: Average web page retrieval time/success rate, under SYN flooding attack. Link 2 is set to 3 Mbps.

We recall that the network setup we used is the one depicted in Figure 6. As we can see from Table II, even under a modest rate of attack of 3 Mbps, the average retrieval time for the standard OpenFlow implementation increases by almost 500%, while almost doubling the attack rate is enough to completely overload the controller (see Table III). By contrast, both AVANT-GUARD and our solution dispatch 100% of packets to destination, but with considerable different overheads: indeed, AVANT-GUARD introduces approximately a 40% overhead compared to the standard OpenFlow under normal network conditions, while LineSwitch introduces only a negligible overhead, i.e., roughly 2% (see Table III).

Implementation	**Avg. Time**	**Success Rate**	**Overhead**
OpenFlow	–	0%	–
AVANT-GUARD	0.568 s	100%	36.92%
LineSwitch	0.426 s	100%	5.45%

TABLE III: Average web page retrieval time/success rate, under SYN flooding attack. Link 2 is set to 6.5 Mbps.

As we can see from Table III, in these tests both LineSwitch and AVANT-GUARD perform better than in the regular traffic scenario, with results well within the standard deviation (see tables I, II and III). This is a strong indicator that both solutions are barely affected by background SYN flooding attacks.

Figure 7 gives a more complete view of the behavior of the system under SYN flooding attack at different attack rates, in all the three cases we considered.

As Figure 7a shows, even under a high attack rate of 20 Mbps, both AVANT-GUARD and LineSwitch guarantee a 0% packet loss. Moreover, the web page retrieval time remains at stable levels with both solutions (see Figure 7b). Finally when considering the original OpenFlow implementation, the controller reaches a critical point between an attack rate of 6 and 6.5 Mbps, where the percentage of successfully delivered packets decreases dramatically to approximately 0%, while the average retrieval time grows extremely rapidly. This behavior is consistent with the results presented in [4], and is exactly what is expected from an overloaded controller.

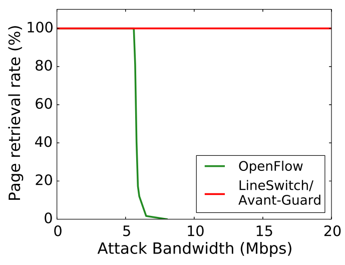
C. Buffer Saturation Scenario

In the tests presented in this section, we simulated the behavior of both AVANT-GUARD and LineSwitch under the buffer saturation attack introduced in Section IV-A. To this aim, we configured the system with different buffer sizes and run the attack at different rates. As a result, we show that:

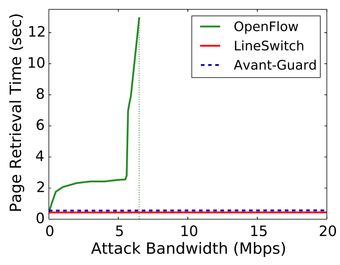
1) The attack rate required to successfully incapacitate an OpenFlow switch running AVANT-GUARD grows *linearly with the size of the buffer*.
2) When using AVANT-GUARD, the throughput needed to successfully complete the attack in a reasonable amount of time *is easily achievable, even with larger buffers*.
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.

The network setup used in the experiments is the same one as in Figure 6. Figure 8 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).

As Figure 8 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 74.72 s, 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} and $P_p = 0.05$, LineSwitch requires 769.49 s to be saturated against only 74.72 s required when running AVANT-GUARD. When using lower (and more realistic) migration probability values, the time difference increases even more, as shown in Figure 8. For completeness, we evaluated the average time required

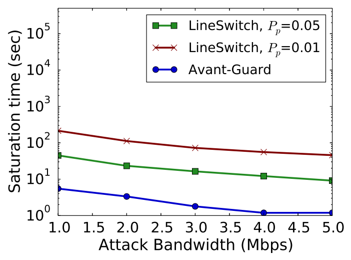


(a) Success rate

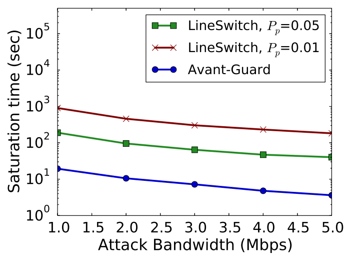


(b) Average retrieval time

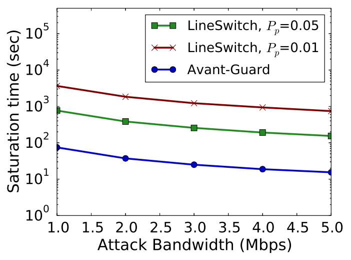
Fig. 7: Average web page retrieval time and success percentage, under SYN flooding attack at different rates.



(a) buffer size 2^{18} Bytes



(b) buffer size 2^{20} Bytes



(c) buffer size 2^{22} Bytes

Fig. 8: Average buffer saturation time, varying buffer size and attack bandwidth, under a buffer saturation attack. Time in logarithmic scale.

to complete a successful buffer saturation attack employing $P_p = 0.001$. Results are shown in Table IV.

Link Bandwidth	Buffer Size	Time
5 Mbps	2^{18}	439.84 s
	2^{20}	1814.00 s
	2^{22}	7375.98 s

TABLE IV: LineSwitch with $P_p = 0.001$: buffer saturation attack with different buffer sizes.

As we can see, when using a probability value $P_p = 0.001$, saturating the buffer of a switch running LineSwitch can take up to hours (7375.98 s $\approx$ 2.02 h) with a 5 Mbps attack rate and 2^{22} Bytes buffer, while under the same conditions AVANT-GUARD is saturated in a matter of seconds. This confirms the resiliency of LineSwitch against this type of attack, and shows how easily it can be adapted to different security needs. As argued in Section V-A1, given the long time needed to saturate the buffer when employing LineSwitch, it is a fair assumption that when an attack is detected most of the longer lasting connections are malicious and can safely be discarded from the buffer with minimal impact on legitimate ones. Moreover, since a high percentage of the incoming connections follows the normal OpenFlow pipeline, on average the attacker will have to complete $1/P_p$ times more connections than with AVANT-GUARD in order to fill the buffer. It would then be easy to detect such cases, since there would be an immediate and prolonged spike in the number of incoming connections from a given set of IPs. Therefore, it is possible to adopt the countermeasures we introduced in Section V-A1.

The size of the buffer we used in our simulations are based on the fact that we assume only a single attacking node with limited computational resources. Nonetheless, our experimental results show that, in order for the attack to be successful, the rate needs to grow just linearly (see Figure 8a, Figure 8b and Figure 8c) with the size of the buffers. Even taking into consideration top of the line OpenFlow switches with gigabytes of memory available (which clearly needs to be shared with many other data structures beside the target buffer), the attacker could make use of a botnet, which today are readily available for rent at accessible prices, to obtain a sufficiently high attack rate. As an example, if we consider an extremely small set of 200 bots with an average bandwidth per bot of 5 Mbps, using our experimental data as a reference (see Figure 8c), a buffer of 4 GBytes of memory (dedicated only to this particular data structure) would be saturated in approximately 74s. Since the size of botnets can range in the thousands [31], it is easy to see how, in general, the size of the buffer does not offer any protection against the buffer saturation attack. Indeed, the required attack rate is easily achievable regardless of buffer size.

D. Configuring the Proxy Probability

LineSwitch efficacy depends partially on the value assigned to the proxying parameter P_p. At the two extremes, i.e., $P_p = 1$ and $P_p = 0$, the behavior of LineSwitch is equivalent to that of AVANT-GUARD and the standard OpenFlow respectively. To better balance the protection level provided by our solution with respect to both control plane saturation and buffer saturation attacks, P_p needs to be configured appropriately. The most important deciding factors in selecting the appropriate value of P_p are the configuration and the history of the network under consideration. As an example, a network provided with multiple controllers with high computational power is less susceptible to control plane saturation attack, and will require a very high attack rate to be incapacitated. In this scenario LineSwitch can be configured with a really low proxying probability value. Indeed, intuitively, even if a single connection request has a low probability of being proxied, given the large number of packets required for a successful attack, at least one will be quickly proxied and the attack detected. On the other hand, a network with low resources will be incapacitated with lower attack rates, and therefore a higher proxying probability is desirable.

In general, we can estimate the average attack detection time d_t as:

$$d_t = \frac{1}{\frac{v}{s} \cdot P_p},$$

where v is the attack speed, s is the size of each packet, and P_p is the adopted proxy probability. Using the same parameters we chose in our experimental evaluation in Section VI-C, we can see that, on average, an extremely low value such as $P_p = 0.1 \times 10^{-3}$ is sufficient to detect an attack in less than one second. Let $v = 6.5$ Mbps be the minimum critical attack rate for a successful control plane saturation attack, as from our results in Figure 7b; moreover, let $s = 70$ Byte (560 bit). A successful control plane saturation attack requires (at least) approximately 12000 packets per second. With a proxying probability $P_p = 0.1 \times 10^{-3}$, on average LineSwitch will recognize an attack in ≈ 0.82 sec. Note that, in our experimental evaluation in Section VI-C, we considered $P_p = 0.05$, which is an extremely conservative proxy probability value: indeed, this choice results in a detection time of $\approx 0,016$ sec.

For a better understanding of the role of P_p in our solution, we can estimate the average attack detection time given a certain confidence probability P_d of detecting an attack. We can model the event of LineSwitch checking a packet sent by the attacker as a Bernoulli trial, with success probability P_p. Moreover, for a finite number n of tries, the probability of detecting the attack, i.e., of proxying at least one packet sent by the attacker, follows the binomial distribution. As a consequence we can write:

$$P_d = 1 - \binom{n}{0} P_p^0 (1 - P_p)^{n-0} = 1 - (1 - P_p)^n.$$

With a confidence of $P_d = 0.9$ (i.e., in the 90% of the cases), and $P_p = 0.1 \times 10^{-3}$, LineSwitch can detect an attack after approximately $n = 23025$ packets sent by the attacker; therefore, under an attack rate of 6.5 Mbps, we have a detection time of ≈ 1.89 sec.

VII. Extending LineSwitch

The proposed version of LineSwitch is a simple but effective modification of the basic OpenFlow switch behavior, implemented completely at the data plane level to offer the highest possible level of protection to the control plane. In its original design [9], we defined LineSwitch as a data plane level-only protection mechanism. In designing our solution, we had two main goals: (1) avoiding any interaction with the control plane, to prevent opening unwanted vulnerabilities, and (2) preserving as much as possible the standard OpenFlow protocol, to ease integration with existing OpenFlow based networks. However, if we relax the above requirements by minimally involving the control plane in LineSwitch, we can provide several improvements to our original design. In this section, we look at possible extensions to LineSwitch, in order to increase its flexibility, enhance its performance, and offer overall better protection.

A. Opportunistic Proxying and PacketIn Rate Monitoring

A first possible improvement to the original LineSwitch design is opportunistic proxying based on detected PacketIn rate.

During the execution of the standard OpenFlow protocol, the control plane is contacted by the OpenFlow switches for each new inbound network flow through PacketIn messages. By keeping a per-switch PacketIn history at the control plane level, it is possible to obtain an average PacketIn rate (PiR) for normal network conditions. Similarly to [15], the controller can then sample the PiR at fixed intervals, in order to detect sudden variations (Δ_{PiR}) from the average, which indicate a possible ongoing SYN-flooding attack. Indeed, if Δ_{PiR} is greater than a given threshold (e.g., twice the PiR standard deviation), the controller will send a specially crafted OpenFlow *TOGGLE_LINESWITCH* message to the OpenFlow switch generating the unusual control traffic. The switch will then enable the base LineSwitch module on the fly, in order to shield the controller from the flooding and to identify the offending client(s). Once the PiR is back in a normal range, the controller will send another *TOGGLE_LINESWITCH* message, deactivating the module.

We implemented a proof-of-concept version of this opportunistic proxying mechanic. Figure 9 shows a preliminary evaluation of this approach, performed with the same setup as illustrated in Figure 6.

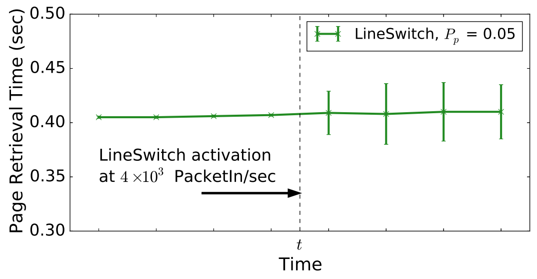


Fig. 9: Page retrieval time using opportunistic proxying. LineSwitch is activated at time t, when the PiR is above the detection threshold. Standard deviation in error bar.

As Figure 9 shows, when the PiR rises above the configured threshold, which we set to 4×10^3 PacketIn/sec, LineSwitch is automatically activated, shielding the controller from the flood. In our preliminary evaluation, LineSwitch runs with proxy probability $P_p = 0.05$.

This solution presents several advantages over the base LineSwitch module:

- Since the LineSwitch module is activated only when needed, the standard OpenFlow protocol is preserved under normal traffic conditions. As we argued in Section IV, this is extremely important as it inherently prevents buffer saturation attacks, as well as allows the correct establishment of TCP connections.

- Since, in general, the network will be under possible attack just for a fraction of its overall lifetime, activating LineSwitch on-demand allows to further reduce the overhead introduced (see results in Section VI-A). While the reduction in the overhead is limited in our preliminary evaluation, this implementation is just a proof-of-concept aimed to prove that LineSwitch can be further improved. Moreover, even with such an early-stage evaluation it is noticeable how the opportunistic activation of LineSwitch is beneficial to the network from the point of view of latency. Indeed, when using LineSwitch, our experiments register a high jitter (as shown by the high standard deviation after time t in Figure 9), which can be disruptive in cases where real-time interaction is needed like VoIP for instance. Activating LineSwitch only when effectively necessary mitigates this problem, allowing for the use of the normal OpenFlow pipeline when the network is not under attack.
- The additional control traffic required by this feature increases linearly with the number of controlled OpenFlow switches, i.e., it requires only one message per switch; therefore the introduced communication overhead is negligible.

Note that, since opportunistic proxying can be obtained without adding any new functionality over the standard OpenFlow protocol, it does not introduce any exploitable mechanics to the network.

The only downside of such modification is the introduction of a new OpenFlow message, and therefore it may not be backwards compatible with OpenFlow switches conforming to older versions of the specifications. However, for these switches it is possible to employ the base LineSwitch module.

Finally, while this proposed modification extends LineSwitch from data plane-only solution to cross-plane solution, the role of the control plane remains limited. Indeed, the controller is only required to opportunistically activate LineSwitch on the attacked OpenFlow switch (and deactivate it once the situation is back to normal). Therefore, we think it is acceptable to still consider LineSwitch a data plane solution and to evaluate it against other data plane solutions (e.g., AVANT-GUARD [4]).

B. Legit Client Protection

The default behavior of LineSwitch is to blacklist IP addresses from which a SYN-flooding is detected. A malicious user could exploit this by flooding the OpenFlow switch with a spoofed IP address in order to have another client blacklisted, thus performing a DoS attack on the victim. Since LineSwitch is designed to work on edge routers and the blacklist is local to each OpenFlow switch, this will work only under the following conditions: (1) the attacker is connected to the same edge router as the victim; and (2) the attacker knows the victim IP address. If these two conditions are met, then an attacker can successfully perform a DoS attack on the victim. It is possible to avoid this by introducing a slight modification in how LineSwitch works. At the switch level, in OpenFlow the traffic is further divided into in-ports (i.e., the physical port where the traffic came from). Taking this into consideration, all that is needed is to restrict the blacklist of a given IP only for the given in-port from where the SYN-flooding is detected. This way, even if a malicious user were to perform a SYN-flooding with a spoofed IP address of a client connected to the same switch, this would not influence the legit client as the blacklisting of the IP would be enforced only for the in-port of the attacker.

VIII. DISCUSSION AND COMPARISON

In this section, we will analyze and compare the collective state of the system while employing the standard OpenFlow protocol, the AVANT-GUARD [4] extension and our own proposal under different attack situations.

1) SYN Flooding Scenario: As demonstrated by our experiments presented in Section VI, the SYN flooding-based control plane saturation attack heavily affects the standard OpenFlow implementation, while being barely noticeable when AVANT-GUARD and LineSwitch are in use. The collective state of the system under SYN flooding, in the three cases we discussed, is summarized in Table V.

	Switch	**Controller**	**Network**	**Host**
OpenFlow	✗	✗	✗	✱
AVANT-GUARD	✓	✓	✓	✓
LineSwitch	✓	✓	✓	✓

TABLE V: System state under typical SYN flooding attack. Symbol ✓ indicates component working normally; symbol ✱ indicates component under distress; symbol ✗ indicates component unresponsive or under considerable distress.

While both AVANT-GUARD and LineSwitch defeat SYN flooding attack, the standard OpenFlow implementation leaves the system exposed to a considerable threat: if the attack rate is high enough, which is easily obtainable as demonstrated in Section VI-B, the controller will be overloaded and consequently the switch will be incapacitated. In general, since a single controller can manage multiple OpenFlow switches on a subnetwork, all the switches connected to it will become unresponsive to new flows, compromising parts of the network. Moreover, until the controller is completely overloaded, the flooding is forwarded to the server possibly causing considerable distress.

2) Buffer Saturation Scenario: In Section VI-C we showed that AVANT-GUARD is heavily vulnerable to buffer saturation attacks. Indeed, a modest attack rate is sufficient to fill the internal buffers of an OpenFlow switch running AVANT-GUARD, preventing it from migrating any new connection and possibly causing problems to a whole subnetwork. As we demonstrated in Section VI-C, even if we increase the size of the buffer for the AVANT-GUARD switch, the attack throughput required for a successful attack grows linearly. Furthermore, after a certain point, extending the amount of memory available for the OpenFlow switch buffer will hit a diminishing return point, where the efficiency of the lookup operations is severely impacted by the large data structures needed. By contrast, when LineSwitch is in use, the problem is highly mitigated through a quick host classification based on probability and blacklisting. As our results confirmed, it is easy to adjust the probability parameter, P_p, to render this attack ineffective, while retaining full protection against the control plane saturation attack based on SYN flooding. Table VI summarizes the collective state of the system under buffer saturation attack.

	Switch	**Controller**	**Network**	**Host**
OpenFlow	✓	✱	✓	✓
AVANT-GUARD	✗	✓	✱	✓
LineSwitch	✓	✓	✓	✓

TABLE VI: System state under buffer saturation attack, described in Section IV-A. Symbol ✓ indicates component working normally; symbol ✱ indicates component under distress; symbol ✗ indicates component unresponsive or under considerable distress.

IX. 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, 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 address these challenges, 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. We experimentally demonstrated 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.

References

[1] 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*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

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

[3] 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.

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

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

[6] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of Network-based Defense Mechanisms Countering the DoS and DDoS Problems," *ACM Computing Surveys*, vol. 39, no. 1, Apr. 2007.

[7] R. Kloti, V. Kotronis, and P. Smith, "Openflow: A security analysis," in *IEEE ICNP '13*, 2013, pp. 1–6.

[8] K. Benton, L. J. Camp, and C. Small, "OpenFlow Vulnerability Assessment," in *HotSDN '13*, 2013, pp. 151–152.

[9] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran, "LineSwitch: Efficiently Managing Switch Flow in Software-Defined Networking While Effectively Tackling DoS Attacks," in *ACM ASIA CCS '15*, 2015, pp. 639–644.

[10] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards Secure and Dependable Software-defined Networks," in *ACM SIGCOMM HotSDN '13*, 2013, pp. 55–60.

[11] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting Traffic Anomaly Detection Using Software Defined Networking," in *RAID'11*, 2011, pp. 161–180.

[12] J. Li, S. Berg, M. Zhang, P. Reiher, and T. Wei, "DrawBridge: Software-defined DDoS-resistant Traffic Engineering," in *ACM SIGCOMM '14*, 2014, pp. 591–592.

[13] R. Braga, Braga, E. Mota, Mota, and A. Passito, Passito, "Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow," in *IEEE LCN '10*, 2010, pp. 408–415.

[14] NOX, "http://www.noxrepo.org/nox/about-nox/."

[15] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "Scotch: Elastically Scaling Up SDN Control-Plane Using vSwitch Based Overlay," in *ACM CoNEXT '14*, 2014, pp. 403–414.

[16] Open vSwitch. http://openvswitch.org/.

[17] H. Mekky, F. Hao, S. Mukherjee, Z.-L. Zhang, and T. Lakshman, "Application-aware Data Plane Processing in SDN," in *ACM SIGCOMM HotSDN '14*, 2014, pp. 13–18.

[18] D. Kotani and Y. Okabe, "A Packet-in Message Filtering Mechanism for Protection of Control Plane in Openflow Networks," in *ACM/IEEE ANCS '14*, 2014, pp. 29–40.

[19] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, and Y. Zhang, "Dfence: Transparent Network-based Denial of Service Mitigation," in *USENIX NSDI'07*, 2007, pp. 24–24.

[20] D. J. Bernstein. Syn cookies. http://cr.yp.to/syncookies.html.

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

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

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

[24] V. Jacobson, "Congestion Avoidance and Control," in *ACM SIGCOMM '88*, 1988, pp. 314–329.

[25] "TCP Selective Acknowledgment Options," IETF, RFC 2018, October 1996.

[26] "TCP Extensions for High Performance," IETF, RFC 7323, September 2014.

[27] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the Characteristics and Origins of Internet Flow Rates," in *ACM SIGCOMM '02*, 2002, pp. 309–322.

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

[29] Mininet, "http://mininet.org/."

[30] POX Controller, "http://www.noxrepo.org/pox/about-pox/."

[31] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "My Botnet is Bigger Than Yours (Maybe, Better Than Yours): Why Size Estimates Remain Challenging," in *HotBots'07*, 2007, pp. 5–5.