Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

DataPlane-ML: An Integrated Attack Detection and Mitigation Solution for Software Defined Networks

Ranyelson N. Carvalho* | Lucas R. Costa | Jacir L. Bordim | Eduardo A. P. Alchieri

Department of Computer Science, University of Brasília, Distrito Federal, Brazil.

Correspondence
*Ranyelson N. Carvalho. Email: ranyelson.carvalho@aluno.unb.br

Abstract

Software Defined Network (SDN) is a paradigm that emphasizes the separation of the control plane from the data plane, offering advantages such as flexibility and programmability. However, from a security perspective, SDN also introduces new vulnerabilities due to the communication required between these planes. SYN Flood attacks are typical distributed denial-of-service (DDoS) attacks that especially challenge network administrators since they produce a large volume of semi-open TCP connections to a target, compromising its availability. Most of the current solutions to detect and mitigate these attacks are designed to operate at the control plane, imposing an additional overhead on controller functions. Moreover, traffic-blocking mechanisms, a widely used alternative to protect network resources, have the drawback of restricting legitimate traffic. This work proposes DataPlane-ML, an integrated solution to detect and mitigate DDoS attacks on SDN, acting directly in the data plane. DataPlane-ML uses machine learning techniques for attack detection and a mitigation solution based on the node's reputation to avoid blocking legitimate traffic during an attack. Experimental results show that DataPlane-ML is $\approx$ 26% faster than statistical-based solutions for attack detection while presenting better accuracy. Moreover, the DataPlane-ML mitigation solution can preserve more than 95% of legitimate traffic during an attack.

KEYWORDS:
Software Defined Network, Distributed Denial of Service, Attack Detection, Syn Flood, Machine Learning, Reputation, Mitigation, Data Plane.

1 | INTRODUCTION

Software defined networks (SDN) emerged as a new network paradigm that decouples the data plane from the control plane. The data plane is responsible for forwarding and routing actions, using devices such as switches and routers [1]. On the other hand, a software-based controller (control plane) is responsible for coordinating the network resources and policies management. The communication between the two planes uses standard protocols such as the OpenFlow [2]. The SDN architecture also defines an application plane that executes network applications in order to help the controller to configure the network. This architecture makes the network flexible, i.e., the network administrator can enhance the controller with new functionalities, including security functions.

Despite these benefits, security is still an important concern in SDN since the separation between the planes increases the attack's surface. Distributed denial of service (DDoS) attacks [1] are particularly challenging because they aim to exhaust the network resources, usually by sending a large volume of traffic from different locations to the attack target. In the SDN architecture, the controller is an attractive target for attacks since

its dysfunction can potentially disrupt the network. In this case, the main goal of DDoS attacks is to overload the controller in order to limit its capacity to handle legitimate network traffic [3].

DDoS attacks can be classified into two main groups: volumetric attacks and non-volumetric attacks (low volume or low rate) [4]. In this work, we restrict our attention to volumetric SYN Flood attacks, which consist of sending numerous TCP connection requests, using spurious source addresses, in the form of SYN segments to the server. The server responds to each received message with an SYN-ACK and then waits for the ACK segment to establish the connection. Since the source address of the SYN packet was forged, the ACK response will never arrive, causing the connection to remain in a semi-open state, consuming controller resources. Consequently, the TCP connection queue fills up overloading the controller, making it unable to respond to legitimate TCP client requests [5].

Statistical and Machine Learning (ML) DDoS attack detection methods have been used to classify legitimate and spurious traffic on SDN [6,7]. In general, these solutions use data collected in the data plane (i.e, traffic flows information) and try to discover whether a determined flow is related to an attack. A flow can be defined as a sequence of packets from a source node to a destination, which may be another node, a multicast group, or a broadcast domain. Usually, protocols to detect DDoS attacks are executed at the control plane [6,7,8]. Hence, the controller is required to maintain continuous communication with the data plane switches to obtain network information. Thus, while the controller facilitates automated network management and makes it easier to integrate and administer applications, the development of security solutions on the control plane is an additional burden to the controller. Given these concerns, SDN security techniques implemented directly at the data plane have been considered in the literature [9,10]. For instance, in [9], an entropy-based (statistical technique) DDoS attack detection mechanism was proposed to act at the data plane. Additionally, data plane solutions are able to detect DDoS attacks faster than solutions implemented at the controller [11].

Compared to statistical techniques, ML alternatives have been widely adopted to detect attacks in SDN networks [1]. Although ML techniques usually present better attack detection accuracy, they are yet to be implemented on the data plane. However, to implement security solutions at the data plane, one must cope with the data plane device's limitations, such as the need to customize the hardware or a particular communication protocol [12,13]. For this purpose, this work resorts to the use of the Programming Protocol-independent Packet Processors (P4), which is a domain-specific language for network devices that specify how data plane devices (e.g., switches, NICs, routers, filters) process the packets [14]. This language has promoted the development of solutions in the data plane with promising results [9,15]. However, P4 cannot handle floating point calculations. To overcome such limitations, the Apache Thrift [16] is used to allow communication among different applications via RPC (remote procedure calls). That is, the P4 functionalities are used to handle the traffic and the Apache Thrift to communicate with the ML and statistical procedures at the data plane. This strategy allows the use of ML techniques in the data plane to provide more elaborate solutions that operate close to the input flow without impacting the SDN controller. Furthermore, the P4 can be implemented on top of white box switches running on off-the-shelf chips, which allows the programmability of the network without being tied to a specific hardware manufacturer. In case of an attack, an eventual overload of the data plane switch would not compromise the controller resources, reducing the attack surface area. We also investigated the impact on the attack detection mechanism caused by different parameters, such as observation time and packet volume.

Trying to minimize the effects caused by DDoS attacks, we proposed a mitigation mechanism based on nodes' reputation, which also is executed in the data plane. A previous work proposed DoSSec [17] to mitigate such attacks by monitoring the flows and classifying them into different priority lists. When an attack is detected, flows are blocked based on the list they belong to. This work extends and combines our previous published papers, which appeared in [17,18], by using a more evolving solution using reputation algorithms, node priority, and ML detecting mechanism applied to the SDN data plane. As far as we are aware, this is the first work to apply ML techniques for attack detection and mechanisms for attack mitigation directly in the data plane. Also, as an additional contribution, we present an extensive experimental evaluation of DataPlane-ML considering different parameters and metrics such as time to detect an attack and CPU usage. The DataPlane-ML attack detection results are compared to traditional statistical approaches. The experimental results show that ML techniques can detect attacks faster ($\approx 26\%$) and with better accuracy than entropy-based solutions. Moreover, the mitigation mechanism based on node's reputation can preserve legitimate traffic with an accuracy of more than 95% during an attack, showing to be able to identify malicious nodes in the network.

In a nutshell, the main contributions of this paper are:

- a data plane solution to detect SYN Flood attacks using ML techniques;
- a mitigation mechanism based on node's reputation to minimize the effects caused by these attacks;
- an integrated attack detection and mitigation solution tailored for SDN.

The remainder of this paper is organized as follows. Section 2 presents the background for this work considering software defined networks, SYN flooding attacks, detection algorithms and mitigation techniques. Section 3 surveys related works. Section 4 discusses the solution we used to establish the reputation for each node. Section 5 describes the DataPlane-ML solution. Section 6 shows the performance evaluation for DataPlane-ML. Finally, Section 7 concludes the paper.

2 | BACKGROUND ON SDN AND DDOS SYN FLOODING ATTACKS

This section begins with an overview and definitions related to software defined networks and SYN Flood attacks. Afterward, a brief overview of frequently employed methods to detect and mitigate DDoS attacks in SDN is presented. Statistical and ML-based are commonly employed methods to identify network traffic deviation that can characterize an attack. The mitigation methods include actions such as blocking network traffic, connection control and resources management, in an attempt to attenuate the network attack effects.

2.1 | Software Defined Networks

This section presents some essential characteristics of software defined network (SDN) architecture, which is decoupled in three planes:

- **Data plane**: this plane contains the network devices responsible for forwarding and routing actions (e.g., switches and routers), as well as monitoring local information and collecting statistics. These devices are configured with a set of flow rules used to redirect incoming packets belonging to a flow. A flow is defined as a sequence of packets from a source node address to a destination node (unicast) or nodes (multicast).
- **Control plane**: this plane contains a controller that manages all the network by configuring the data plane (e.g., switches), being responsible for creating and coordinating the functionalities of the network devices. It communicates with the data plane by a southbound interface using standard protocols such as OpenFlow[2].
- **Application plane**: this plane comprises a set of network applications (e.g., load balancer and QoS services) that help the controller. It communicates with the control plane by the northbound interface, which provides a platform for network applications development.

Figure 1 shows the SDN elements, depicting the application, control and data planes, as well as the northbound and southbound interfaces. The decoupling between the implementation of network control logic and forwarding operations promotes more flexibility in network control and management. Moreover, recent advances in programmable data planes (PDP) allowed network administrators to redefine the behavior of data plane devices (switches)[19], bringing even more flexibility and facilitating the development of new network protocols. The PDP is usually composed by P4-switches that can interact with the incoming packets by using the P4 language[20]. This work explores this aspect and proposes an ML-based attack detection and a reputation-based mitigation mechanism that executes in the data plane, thus avoiding communication with a possibly overwhelmed controller.

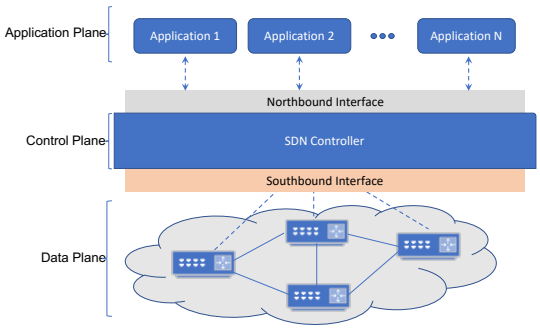


FIGURE 1 SDN plane separation: application, control and data plane.

2.2 | TCP Connection and DDoS SYN Flood Attacks

TCP connection establishment is based on the well-known three-way handshake. The handshake implies the exchange of TCP segments with special flags set between the client and server. The exchange of these TCP segments is performed in the following three steps: (1) the client sends a TCP segment with the SYN flag set to the server; (2) on receiving the TPC SYN, the server replies to the client with a TCP segment with the SYN-ACK flag on; (3) the client sends a TCP segment with the ACK flag on and the connection is established. The connection eventually ends with an RST (reset or tear down) or FIN (gracefully end the connection).

The SYN flood attack abuses the TCP three-way handshake to exhaust the victim resources[5]. The attacker uses spurious source addresses in the SYN segment. Consequently, the server allocates and initializes connection variables and buffers and replies with an SYN-ACK to a fake

address, and waits for the ACK reply to establish the connection. As the ACK is not received, the connection remains in a semi-open state (SYN-RECV) at the server, i.e., it remains in the server TCP connection queue. With numerous such semi-open connections, the attacker can overwhelm all available TCP connection queues on a targeted server machine, overloading the server that eventually will be unable to respond to legitimate TCP client requests.

2.3 | DDoS Attack Detection Algorithms

Distributed denial of service (DDoS) attacks aim to deplete device or infrastructure resources to cause service unavailability [3]. DDoS attack detection methods can be classified through data classification algorithms. They can be categorized into methods based on statistical algorithms and machine learning (ML) algorithms. Statistical algorithms perform analysis of various traffic properties in both phases: with and without spurious traffic. These properties are used to create a reference model to identify behavioral deviations. Entropy [6], ϕ-Entropy [21] and Chi-Square [22] are statistical methods commonly used to detect network traffic anomalies.

ML algorithms are trained to understand the network traffic behavior, learned from historical data, to identify potential threats [7,23]. We restrict our attention to supervised learning techniques that are discussed in the literature review presented in the subsequent section. The K-Nearest Neighbors (KNN) [24] has been widely employed to detect network anomalies. The algorithm works by searching the nearest k-neighbors based on the distance between points present in a dimensional space. KNN computational complexity is proportional to the size of the training dataset for each test sample. The KNN runs in $O(nd)$ time, where n is the number of samples, and d is the number of dimensions [25]. Similarly, Support Vector Machine (SVM) [26] has gained increasing attention lately. SVM algorithm searches for a hyperplane that better separates the data of each class that maximizes the separation margin. To find a suitable hyperplane, the algorithm runs in $O(n^3)$ time, where n is the number of samples [27]. Random Forest (RF) [28] is another ML technique widely employed to detect network attacks. RF consists of a collection of classifiers structured in a decision tree, where the classification result is the class with the larger number of votes among the trees present in the forest for a given selection of random samples. RF runs in $O(mkn \log n)$ time, where m is the number of random trees, n is the number of samples and $k \ll m$ is the number of variables drawn randomly on each node to build the random trees [29].

2.4 | DDoS Attack Mitigation Techniques

The defense mechanisms against DDoS attacks in SDN include different mitigation techniques. Mitigation techniques can be categorized into three classes: (i) blocking (or discard); (ii) control and (iii) resource management [30]. Each technique has associated advantages and disadvantages, ranging from a more severe application, such as blocking all traffic, or even a more refined control, which searches to preserve a greater volume of legitimate traffic in the network.

Blocking and discard classes are the simplest and fastest mitigation techniques that completely block possible sources of attack [12,31,32,33]. This technique provides immediate mitigation to network resources. However, if a legitimate node (host) or port is compromised, it will be completely blocked, resulting in the discard of any legitimate traffic from such sources. In this way, it is necessary to have an intrusion detection system capable of reducing false-positive rates to maintain network connectivity for legitimate users. The control class aims to delay, i.e., redirect spurious traffic, by reducing the bandwidth or number of messages sent to the controller when an attack is detected [34,35]. This technique is more complex than blocking, increases processing time and network traffic, but legitimate nodes or traffic declared malicious by the detection system still have a chance to communicate with the network. On the other hand, the attacker will continue to consume some network resources, as it is not immediately blocked by the solution. Finally, the class resource management configures and manages the network through additional resources present in the infrastructure [36,37]. Thus, the solutions use auxiliary mechanisms such as IDS (Intrusion Detection System), CICFlowMeter [38] and sFlow-RT [39]. However, with the use of this type of technique, the solutions may be limited to the resources available by the infrastructure for the implementation of the proposed mechanism.

3 | RELATED WORKS

As discussed in the previous section, the use of ML-techniques for detection and methods such as blocking, control, or resources management for mitigation of network anomalies has gained increasing attention in the past few years. As software defined networks mature and become more popular, security concerns are still challenging network operators, fostering the development of SDN-tailored solutions. This subsection discusses closely related works, focusing on the DDoS mitigation and attack detection, in particular the SYN Flood attacks on SDN.

A method to detect DDoS attacks using SVM-based algorithms has been proposed in [8]. The method works on the control plane and uses the flow information available to the controller to learn and classify traffic into normal and spurious traffic. Obaid et al. [40] proposed a similar approach

employing distinct ML algorithms such as RF, J48, and KNN. Deepa et al.[41] proposed an approach that combines different ML algorithms (KNN, SVM, and Self Organizing Maps - SOM) to perform the attack traffic classification. In the same line, Tuan et al.[42] proposed the use of different approaches, combining entropy with KNN to detect DDoS attacks. Similarly, the authors in[43] proposed a solution that allows entropy to be combined with different ML algorithms, such as Multi-Layer Perceptron (MLP), KNN, and SVM. In[44], the authors proposed a solution to detect DDoS attacks anomalies using NIDS (Network Intrusion Detection System) on an SDN environment. Different ML classifiers (RF, SVM, Logistic Regression (LR), and KNN) have been employed. The impact of different observation windows on the detection performance of three classification algorithms (SVM, Naive Bayes, and KNN) has been evaluated by Samer et al.[45].

The following works search to reduce the impacts caused by the SYN Flood attack. The Avant-Guard is proposed in[12]. The idea is to turn the switch into a proxy. The proposal intercepts TCP connection information in one session and forwards only complete requests to the controller. As a side effect, the solution increases the delay caused in establishing legitimate connections due to the proxy. This allows attacking hosts with already validated IPs. Similar to Avant-Guard[12], the solution proposed in[31] turns the controller into a proxy, which always rejects the client's first connection. The solution maintains an SYN request counter, and if the number of SYN requests exceeds the limit, the controller performs the attacker's blocking action. Although the solution is efficient, if the attacker completes the first TCP handshake, then he/she has no further obstacles to attacking the victim. Kumar et al.[32] proposed a solution that uses entropy as a detection mechanism. The analysis is based on the destination IP of the packets to detect anomalies and identify the attack source. Mitigation involves discovering the malicious source and blocking it on the switch port. Although the solution is effective, blocking the switch port can penalize all legitimate flows linked to that port. Kim et al. proposed to detect the attack using the mechanism TCP Time Out and Round Trip Time (RTT)[33]. The solution discards the first SYN packet from any host and estimates the holding time based on the time between the first and second SYN packets, removing the half-open TCP sessions after the holding time. If the RTT of the ACK is less than the noted RTT, the ACK packet is forwarded to the server, otherwise is discarded by the solution[33]. Although the solution is effective, having a very short RTT can penalize legitimate hosts that have trouble completing a connection.

In[36], the authors proposed a methodology to detect the SYN flood attack by comparing the SYN count (extracted from the sFlow-RT analyzer) with an adaptive threshold, which is computed using EWMA (exponentially weighted moving average). When the attack is detected, a rule is applied to discard the flows in the source switch of the attack. Although the solution is efficient in detection, it does not show how legitimate and spurious flows are distinguished, which can generate numerous false positives. The authors in[37] proposed a solution that is based on two components: (i) the monitor; and (ii) the correlator. The first receives alerts from an IDS and synthesizes information related to the attack and sends it to the correlator. The second operates on the controller and verifies the information received from the monitor by analyzing the input flow information. After confirming the existence of an attack, the mitigation process is performed to block the attacker's IP and port and discard the attack traffic. Although the solution has efficiency, port-blocking can penalize legitimate flows.

Lin et al.[34] proposed a simple detection and blocking strategy. For detection, the authors analyzed of SYN/ACK and ACK/FIN packet ratio. Thus, if the ratio between these packets does not match, there is an indication of network anomalous traffic. The blocking strategy aims to limit and discard the attacker's traffic according to an IP range that is closest to the attacker considered suspicious, according to the IP address class (e.g, A, B and C). Although the solution limits attack traffic, it can penalize legitimate traffic if no prioritization rules are applied. The authors in[35] proposed a solution similarly to Lin et al.[34]. The solution applies a dynamic rate threshold to the number of packets sent to the controller. It then collects flow statistics from switches, based on packet and byte rate, to measure and control the rate of packets sent to the controller and set the detection threshold. When this limit is exceeded, the switch then reduces the number of flows sent to the controller. Although the solution performs the control of flows sent to the controller, it does not distinguish the type of flow to be delayed.

The authors in[46] analyzed the reliability of SDN applications, through reputation. The reputation is made through a beta distribution, granting a credit. The main factor for attributing the reputation is the feedback from the controller in relation to the customer's request. In[47] the authors used reputation to evaluate external applications. The solution takes historical data into account for establishing a reputation. Reputable flows have a higher priority to be served. The solutions presented do not explore the risks related to DDoS attacks, but present how the concept of reputation can be used in an SDN network.

It is observed that most of the detection solutions were designed to work in the control plane, such a strategy increases the controller duties and the communication latency to collect and manage routing/forwarding devices located at the data plane and can affect the processing time, resulting in controller overload and longer detection times. An alternative is to develop solutions that act on the data plane, thus reducing the detection time and overhead to the controller. Some works proposed to execute detection processing in the data plane, but they concentrated only on the use of statistical methods, due to the complexity of developing solutions to act on this layer[9,10,34]. In the same way, the state-of-the-art solutions[8,41,45] showed the potential of ML algorithms for data classification and their use in detecting DDoS attacks. To the best of our knowledge, the works in the literature consider the use of ML algorithms only in the control plane. Furthermore, it is necessary to propose methods to reduce the impacts caused by a DDoS attack, considering that most of the solutions presented proposed actions to block and discard spurious traffic, but they can penalize legitimate traffic as this type of attack can simulate the behavior of legitimate network traffic, making it difficult to differentiate

between legitimate and spurious traffic. The use of reputation and prioritization methods can assist in this process, allowing legitimate traffic to be served more efficiently by preventing it from being blocked.

The main contribution of this work, as will be presented in the next section, is to overcome these limitations. More precisely, we propose an ML attack detection mechanism that works on the data plane and encompasses a reputation mechanism to avoid blocking legitimate traffic, thus reducing the impacts caused by a DDoS attack.

4 | REPUTATION-BASED CONNECTIONS

This subsection presents the EigenTrust algorithm, which will be used later in the proposed DataPlane-ML to lessen the effects of a DDoS attack on an SDN infrastructure[48]. Node reputation, or trust, has been used in the context of P2P networks to stimulate nodes to share their resources. That is, nodes with a higher reputation are granted higher priority to certain services, such as faster downloads and other benefits. The EigenTrust, proposed by Kamvar et al.[48], is a reputation algorithm for computing trust on file sharing systems. Each node evaluates interactions with other nodes to establish a trust level. Let N denote the set of nodes in a communication network. Eigentrust uses the notion of transitive trust, where a node $n_i \in N$ trusting a node $n_j \in N$, would also trust the nodes that n_j trusts.

In the context of this work, when a node successfully completes a TCP connection, the transaction is classified as satisfactory, which will generate a positive trust impact for the node issuing the request. For failed connections, that is, those in which the node issuing the request was unable to complete the connection, the transaction is classified as unsatisfactory, which will generate a negative impact for the node. The trust calculation is performed based on connection success and failures as follows:

$$S_{ij} = \text{sat}_{i,j} - \text{unsat}_{i,j}, \tag{1}$$

where $\text{sat}_{i,j}$ is the number of successful connections issued by node n_i and received by node n_j and $\text{unsat}_{i,j}$ is the number of unsuccessful connections attempts from node n_i to node n_j.

The local trust values are aggregated, normalized and calculated according to the reputation of other nodes. The trust values obtained by EigenTrust are not contained in any defined interval, that is, each node may use distinct intervals to assign grades to each service. However, to optimize the performance of comparisons between reputations, EigenTrust performs a normalization for the local trust value. In such a way, it is ensured that all local trust values will be between 0 and 1. Thus, for each node $n_i \in N$, the normalized local trust value (C_{ij}) is defined as:

$$C_{ij} = \begin{cases} \frac{\max(S_{ij},0)}{\sum_j \max(S_{ij},0)} & \text{if } \sum_j \max(S_{ij},0) \neq 0, \\ 1/|P| & \text{otherwise.} \end{cases} \tag{2}$$

In Eq.(2), if the $\sum_j \max(S_{ij}, 0) = 0$, C_{ij} will be undefined by the algorithm. For the sake of simplicity, we consider the existence of a priory notion of trust. Other cases are discussed in detail in Kamvar et al.[48]. Let P, ($P \subset N$), denote a subset of trustworthy nodes. Then, a node n_i may set $C_{ij} = 1/|P|$ as its local trust value. Note that P can be obtained by exchanging trust information with other nodes. Once the local trust for each node n_i is obtained, a global trust value can be computed. The normalized local values are aggregated with other trust values in order to increase the knowledge about the nodes that participate in the network. The global trust value (T_{ci}) of a node n_i is obtained by summing the product of the node's local trust value (C_{ij}) and the corresponding global trust value of the other nodes (T_{cj}), defined as:

$$T_{ci} = \sum_j C_{ij} T_{cj}. \tag{3}$$

The global trust value of a node is obtained from the references collected from other participants and weighed with their own reputation. This feature has the aim to limit the influence of bogus reputation information that could be sent by malicious nodes, usually with low reputation.

The EigenTrust algorithm has been proposed in the context of peer-to-peer networks, where the client and server roles change over time. That is, a node n_i may be the server of a file F that is being downloaded by other clients (peers). Likewise, n_i could be a client downloading files from other servers (peers). In this work, we consider a client-server SDN architecture, where the roles of the clients and the servers are well-defined. More precisely, in this work, we are interested in establishing the client's reputation based on their connectivity patterns to a destination network that comprises some servers. The goal is to determine the reputation of the clients, based on connectivity patterns, and later classify them according to their trust values. Hence, in the SDN context, the set of hosts N includes the set of client nodes $n = n_0, n_1, n_2 \cdots, n_i$, $i \geq 0$ and the set of servers $h = h_0, h_1, \cdots, h_j$, $j \geq 0$. That is, $N = n \cup h$. For latter reference, we define the averaged global trust of n clients, in the context of SDN, as:

$$T_a = \frac{1}{n} \sum_{i=1}^{n} T_{ci}. \tag{4}$$

More precisely, T_a will be used as a threshold to determine whether the global trust T_{ci} of a node n_i is high enough to be trusted by a switch s_j that is processing n_i's connection requests.

5 | DATAPLANE-ML

This section presents the DataPlane-ML, which is an integrated solution to detect and mitigate DDoS attacks on SDN. The DataPlane-ML uses ML techniques for detection and traffic reputation mechanisms to avoid blocking legitimate traffic and, thereby, reduce the impacts caused by SYN flood attacks. The main challenge addressed by DataPlane-ML is to execute the ML and reputation algorithms at the data plane, without imposing additional overhead on the SDN controller.

5.1 | Attack Detection and Mitigation Overview

DataPlane-ML comprises two main modules: the detection and mitigation modules. The former module analyses and collects statistical information to identify network threats. In case of a network attack, the mitigation module is used to alleviate the attack effects by allowing trusted nodes to access network resources. These modules are depicted in Figure 2. The detection module consists of two submodules, the data acquisition submodule, and the attack detection submodules. Data acquisition and statistic computation are performed by the SDN data plane switches. In this work, the data plane switches are assumed to incorporate the P4 language to process the input data. That is, the P4-enabled switch extracts flow information and computes statistics from data obtained from the packet's header fields. Recall that a flow is a sequence of packets from a source node address to a destination node (or nodes). That is, the destination may be a single node or even a multicast group. The collected data (flows) can be grouped into discrete blocks w_i, $i > 0$, which we call "window". The collected data and the corresponding statistics are then submitted to the attack detection submodule. The DataPlane-ML takes advantage of the fact that SDN data plane switches can be implemented on top of white-box switches, using off-the-shelf chips, to run ML algorithms. The data plane switch runs the Apache Thrift [16] framework to connect the modules and allow the communication between different processes via Remote Procedure Calls (RPC). The ML model receives the data flow from the P4-switch and executes the ML algorithms to determine whether a flow is related to an attack or not. In case of an attack, this information is sent to the mitigation module. The mitigation module uses a reputation-based method to establish priority for incoming flows, assigning them to different priority lists. In case of an attack, the SDN may still provide services to those nodes/flows that meet a certain trust level, through list management. The trust level is computed based on the node/switch interaction and priority lists.

The attack detection and mitigation modules operate by monitoring incoming connections. The connection monitoring, which is part of the mitigation module, classifies the incoming connections as successful and unsuccessful. This information is used later to define the reputation of each node, based on its interactions over time. The incoming connections are grouped into flows, which are then stored in observation windows $w_i, i \geq 0$. As mentioned before, the observation windows can be set to hold a specific number of packets (flow-based windows) or to store packets over a time interval (time-based windows). The attack detection module uses the information stored on the observation windows to identify a possible network threat. In case of an attack, the information is relayed to the mitigation module, which enable new connections only to those nodes that have a high priority to access network resources. Such a strategy allows the network to take actions to prevent malicious nodes accessing network resources using both proactive and reactive strategies. The proactive strategy is executed by the mitigation module, by acting upon individual connections and categorizing the nodes in different priority lists. These lists are monitored and, based on node interaction, promote or restrict nodes to access network resources. The reactive actions are taken by the attack detection module in case of an attack, as mentioned before. The details of the attack detection and mitigation modules are presented next.

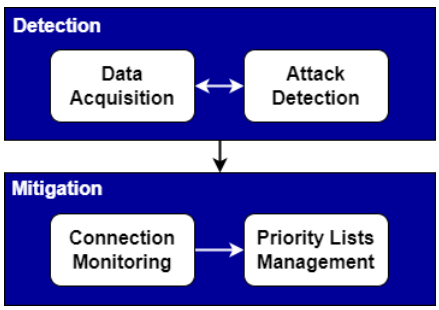


FIGURE 2 DataPlane-ML modules.

5.2 | Attack Detection Modules

This subsection presents the DataPlane-ML attack detection modules. These modules comprise the data acquisition and attack detection submodules, which are detailed next.

5.2.1 | Data Acquisition Submodule

The data acquisition submodule analyses the flows and forwards the corresponding statistics to the attack detection submodule. Hence, the first task of this submodule is to group packets into flows, which is performed at the P4-switch, shown in Figure 3. To determine the flow that a packet belongs to, DataPlane-ML considers a 4-tuple composed of its source and destination IP addresses and ports fields. First, the P4-switch receives the input packets and validates them, extracting packet-header information. Each flow is assigned a unique identification number, registered in the flow_id variable. Incoming packets from an existing flow have their statistics computed and registered for that flow, such as flow_bytes, flow_duration and flow_packets. This information is periodically sent to the detection module. More precisely, the input data is grouped into observation windows, where each window contains information of several flows. Each observation window w_i, $i \geq 0$, can be defined either by a fixed number of flows (flow-based window) or by a time interval (time-based window). In the former case, the observed windows contain the same number of flows (i.e., $|w_i| = |w_j|, 0 < i < j$). In the latter case, the number of flows may be $|w_i| \neq |w_j|, 0 < i < j$, as the number of flows may vary over time.

FIGURE 3 Grouping packets into flows.

5.2.2 | Attack Detection Submodule

This work focuses on supervised learning algorithms that act upon the input data (observation window) received from the data acquisition submodule. The attack detection submodule flow is shown in Figure 4. The input traffic data to the attack detection submodule contains the attributes acquired at the data acquisition submodule. Upon receiving this data, which is a collection of flows and their respective statistics grouped into observation windows, they are forwarded to the data preprocessing step. In this step, the data format is converted to transform nominal data into binary attributes. Such transformation is useful to execute the normalization process with the aim of avoiding unbalanced attributes. Then, the attack detection submodule verifies whether a trained classification model exists. If it exists, the prediction is performed according to the specified ML algorithm. Otherwise, the data goes through the dataset preparation step, which divides the dataset into training and testing to perform data prediction according to the defined ML algorithm, in which a portion of the data is used for training and another for the test (usually at the rate of 70% for training and 30% for testing). The training and testing are performed on synthetic traffic, where both legitimate and spurious traffic is generated (further detailed in Section 6). The corresponding labels for spurious and legitimate traffic are provided to the model training. For the testing step, the labels are removed, to analyze the behavior of the model after the training phase. The results of the training are used to evaluate the model behavior. After defining the ML algorithm, an optimization procedure is executed to search for the best ML parameters. These parameters are optimized through a cross-validation search, to avoid model overfitting by finding the best classification model [23]. The goal is to increase the classifier's reliability and precision. To update a trained classification model, the previous model is removed, forcing a new training process. This update depends on several factors, including the change in network traffic characteristics and the ability of the model to detect and mitigate network attacks. Hence, the most appropriate time to update the model may change from one network to another. While it is of interest to identify an appropriate time for updating the model, such a task is outside the scope of this work.

The best model for the ML algorithms is used to classify the network traffic into normal or attack. The result of this classification is sent back to the P4-switch at runtime, which may act to mitigate the attack (discussed in the subsequent section). Note that the above procedures can take advantage of the switch idleness periods to perform model training and optimization. Also, to better explore the switch idleness periods, the network administrator may use the information of the past updates as an approximation of the time needed for future updates.

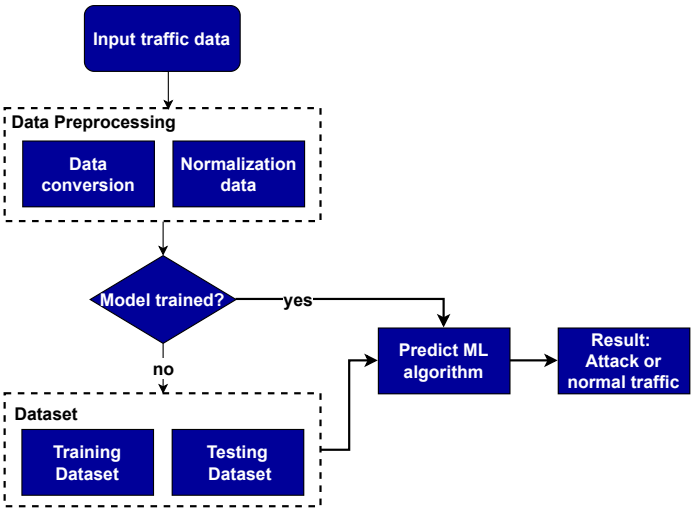


FIGURE 4 ML Attack Detection Module.

5.3 | Mitigation Module

As mentioned before, a DDoS attack works by exhausting the network resources so that legitimate traffic cannot be serviced. When the attack detection submodule reports a possible attack, the mitigation module will act to lessen the attack effects. That is, the mitigation module's goal is to maintain connectivity with trustworthy nodes even in the event of an attack. In addition, the mitigation module acts proactively, identifying misbehavior of nodes and imposing limits and restrictions on them. As will be discussed next, the mitigation module identifies trustworthy clients, so that the connections from these nodes can be prioritized.

5.3.1 | Priority Lists Overview

For each node, DataPlane-ML monitors its attempts to establish a TCP connection and uses four distinct lists for its classification. A flow from an unknown node, as well as flows from nodes that have not reached a determined level of trust, are categorized as unclassified. These nodes are inserted into the *unclassified list* (UL) and have no priority when handled by the P4-switch. Upon a successful TCP connection establishment, a node obtains a trust value that allows it to move from the UL to the *graylist* (GL). After, depending on its behavior, it can be promoted to the *whitelist* (WL) or downgraded to the *blacklist* (BL). Moreover, a node may lose its status and be downgraded to BL when considered suspicious, i.e., after some unsuccessful connections attempts. Additionally, a node in the whitelist can lose the trust it received and be characterized as unclassified. Finally, a node moved to the blacklist is temporarily blocked for $t_i = \Psi$ time, where $\Psi \geq 0$. Until the t_i timer for the node n_i expires, the P4-switch will not process a request from this node. Once the blocking time expires, the node may return to the unclassified list and corresponding requests will be serviced without any priority. Figure 5 shows the node migration among lists as its trust level changes.

Clearly, the aforementioned priority scheme requires a trust threshold for each list as well as a trust level for each node. The mitigation module uses two submodules to address these problems. The connection monitoring submodule is responsible to: *(1)* monitor the attempts to establish TCP connections and update the statistics for each node, which are used in the priority lists management submodule; *(2)* promote a node from UL to GL upon a successful connection, giving the opportunity for well-behaving nodes to reach faster a higher priority list; *(3)* downgrade a node to BL after some unsuccessfully attempts to establish a TCP connection, this is important to block a node that did not finish any connection; and *(4)* unblock a node in BL after its blocking time expires, giving the opportunity to establish new connections. On the other hand, the priority lists management submodule uses the collected statistics and the aforementioned reputation method to calculate a trust level for each node in the lists with higher priority, namely GL and WL. This submodule also monitors the node priority changes, and may impose restriction on nodes that have an unstable level of trust (i.e., temporarily blocking the node by adding it to the BL).

5.3.2 | Connection Monitoring Submodule

Connection monitoring register and analyzes connection requests from nodes, classifying them as success (satisfactory) or failure (unsatisfactory). The classification is performed by the P4-switch, which monitors the TCP flags exchanged between the source (client) and destination (server),

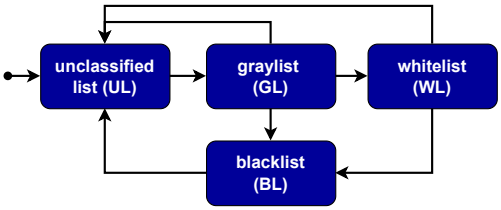


FIGURE 5 Classification flow of the priority lists.

realized during the three-way handshake process. A timeout is established for each TCP SYN request, which is monitored by the P4-switch. The timeout is defined based on historical data (previous TCP connection timeouts), considering an environment operating in normal conditions. In this way, a proper timeout is established from the point at which the P4-switch receives the first TCP SYN request up to the first ACK to complete the TCP connection. Successful TCP connections generate a positive impact on the node's reputation. On the other hand, unsuccessful TCP connections have a negative impact on the node's reputation. Let γ_i denote the number of failed connections attempts issued by the node n_i. If γ_i exceeds a predefined threshold Γ, the connection is discarded and the node is added to the blacklist. Recall that nodes in the blacklist are blocked by $t_i = \Psi$ time, where $\Psi \geq 0$. When the timer expires, the node is then removed from the blacklist and added to the unclassified list.

Algorithm 1 summarizes the above discussion. In the first step, the statistics of each successful connection are updated, whose connections are classified as satisfactory for each node belonging to the priority lists (graylist or whitelist). If the connection is classified as unsuccessful, the node is added to the blacklist in case $\gamma_i \geq \Gamma$ and promoted to the UL when the timer t_i expires.

Unlike the attack detection module that operates on a set of flows (observed windows), the connection monitoring algorithm is triggered based on TCP connection requests. That is, the node's TCP connection success and failures dictate whether a node can be promoted to a priority list or downgraded to a low priority list. To this end, Algorithm 1 monitors the TCP connections and acts upon a connection's success or failure.

Algorithm 1 Connection Monitoring for node n_i

- **Step 1:** Upon a successful TCP connection from node $n_i \in n$ where $n_i \notin$ BL do:
  Step 1.1: Promote: If node $n_i \in$ UL, then promote n_i from the UL to the GL;
  Step 1.1: Update: If $n_i \in$ GL or $n_i \in$ WL, then update the statistics for node n_i on the corresponding list (GL or WL);
- **Step 2:** Upon an unsuccessful TCP connection from node n_i, where $n_i \in$ GL or $n_i \in$ WL do:
  Step 2.1: Unsuccessful Connection: If $\gamma_i \geq \Gamma$, then add n_i to the BL, set $\gamma_i = 0$, and remove n_i from the priority list (GL or WL); Otherwise, set $\gamma_i = \gamma_i + 1$;
- **Step 3:** If $n_i \in$ BL and the timer t_i expires, then promote n_i from the BL to UL.

5.3.3 | Priority Lists Management Submodule

The priority list management is responsible for handling the nodes present in the priority lists (GL or WL). A global trust threshold is used to evaluate the transition among lists (either to elevate or to reduce a node's priority). The global trust threshold defines the sensitivity of the proposed mechanism. The global trust threshold is a weighted threshold defined as

$$T_g = \alpha \cdot T_a + (1 - \alpha) \cdot T_g, \tag{5}$$

where T_a is the averaged global trust computed by Eq.(4) and T_g is the previously calculated global trust threshold and α is a smoothing coefficient. The coefficient α assumes values in the range ($0 \leq \alpha \leq 1$). A higher α implies a heavier weight on T_a in contrast to the past computed values in T_g. In contrast, a lower α value implies a heavier weight on past T_g values, in contrast to current average trust value T_a. Thus, the threshold can be adjusted according to the network characteristics as well as to suit the administrator's needs.

For each priority list (graylist or whitelist), the global trust threshold T_g is computed. Thus, if the global trust T_{ci} of a node n_i belonging to the graylist is above the threshold (T_g), then the node n_i is promoted to whitelist. Conversely, a node belonging to a priority list, whose global trust T_{ci} is equal or below the T_g threshold, the node is moved to the unclassified list. The latter event is recorded in r_i, which counts the number of removals of n_i from the priority lists. If the node exceeds the number Δ of allowed removals from the priority list, the node is then added to the blacklist and the timer $t_i = \Psi$ is set.

The aforementioned steps of the priority lists processing are shown in Algorithm 2. Note that the algorithm may be executed at irregular intervals. That is, Algorithm 2 may be processed even before an observation window $w_i, i \geq 0$ is completed. This allows the list processing to be updated at a faster or slower pace accordingly to the administrator and application needs. If the network is under attack, the P4-switch prioritizes nodes in WL to be serviced, while the nodes in GL will only be serviced if there are no pending requests from a node $n_i \in$ WL.

Algorithm 2 List Processing

- **Step 1:** Compute the global trust threshold T_g using Eq. (5).
- **Step 2:** For each node $n_i \in$ GL, if its $T_{ci} > T_g$, then promote n_i to WL;
- **Step 3:** For each node n_i, if its $T_{ci} \leq T_g$, then add n_i to the UL; remove it from the corresponding list (GL or WL) and increment the removal counter $r_i = r_i + 1$;
- **Step 4:** If $r_i \geq \Delta$ for node n_i, then remove the node from the priority list (GL or WL) and add n_i to the BL, set $t_i = \Psi$ and $r_i = 0$;

6 | PERFORMANCE EVALUATION

This section evaluates the proposed DataPlane-ML. We begin by detailing the experimental environment parameters, the evaluation methodology, and the metrics used to assess the results. Next, the DataPlane-ML attack detection modules are evaluated using a set of ML algorithms. Finally, the mitigation modules and priority lists behavior are examined.

6.1 | Evaluation Setting and Methodology

The DataPlane-ML is evaluated in two phases: the first phase evaluates the attack detection module with KNN, SVM, and RF algorithms. These algorithms have been selected because they are widely used for data classification as well as to detect network attacks [23,41,45]. The algorithms are implemented in the data plane using the Scikit-Learn libraries [49]. The `RandomizedSearchCV` function has been used to select the best hyper-parameters to train and the detection model fitting using the k-fold cross-validation, listed in Table 1. Entropy, which is a statistical-based attack detection method, has been selected for comparison purposes [6]. The following metrics will be used to evaluate the attack detection phase:

- **Cross-validation CPU usage:** measures the CPU usage on the switch while executing the cross-validation process, as well as the CPU usage to detect an attack;
- **Cross-validation computing time:** time spent to compute the cross-validation;
- **Accuracy:** indicates the percentage of packets (legitimate or spurious) the model has correctly classified during the attack;
- **Detection time:** elapsed time between the beginning of the SYN-flood attack and its detection;
- **Windows size:** number of packets present in the observation window during the attack detection phase.

The second phase refers to the mitigation process, in which the connection monitoring and reputation algorithms are used. Recall that the connection monitoring (Algorithm 1) evaluates the success of TCP connections, while Algorithm 2 computes the node's average global trust to establish node priority. The following metrics will be used to evaluate the mitigation phase:

- **CPU usage:** measures the CPU usage on the switch to execute the reputation algorithm using distinct smoothing coefficients;
- **Local trust computation time:** indicates the time that the solution spends to compute the local trust value of the nodes;
- **Variability of global trust value:** indicates the variability of global trust value with varying observation window occupancy percentage. Given an observation window $|w_i| = f$ flows, the observation window occupancy percentage is defined as $f \cdot x$, where $(0 \leq x \leq 1)$;
- **Priority list smoothing coefficient variation:** indicates the impact of the smoothing coefficient to compute the global trust threshold;
- **Number of packets present in the lists according to the smoothing coefficient:** number of packets present in the switch according to the applied smoothing coefficient;
- **Accuracy:** indicates the percentage of packets (legitimate or spurious) the model has correctly classified.

The DataPlane-ML algorithms use a number of parameters, which can be used to tune their performance. Table 1 lists the parameters used in the experiments, detailed in the subsections 6.2 and 6.3. The Mininet [50] network emulator is used to provide the SDN environment, which is used in both phases (attack detection and mitigation). The experiments were performed on an i7, 3.0GHz CPU with 8Gb of RAM running Ubuntu 16.04 LTS. The SDN topology consists of 4 switches managed by a single controller and 10 hosts (9 clients and 1 server). The network is connected to the other networks via a gateway, from which it can communicate with the other networks and devices, as shown in Figure 6.

TABLE 1 DataPlane-ML experimental parameters.

Parameter	Value
Attack type	Syn Flood
Experimental time	30 seconds
Attack starts	13th second
Attack ends	22th second
Spurious packet rate	100 HTTP requests per second
Legitimate packet rate	10 HTTP requests per second
Bandwidth	100 Mbps
Flow-based window duration	$w_i \in \{125; 150; 200\}$ flows
Time-based window duration	$w_i \in \{0.5; 1.0; 2.0\}$ seconds
Hypeparameters algorithm ML: KNN	n_neighbors:{5} weights:{distance}
Hypeparameters algorithm ML: SVM	C:{1.0} Kernel:{RBF} Gamma:{1.0}
Hypeparameters algorithm ML: RF	n_estimators:{100} max_depth:{10} max_features:{log2} min_samples_split:{4} min_samples_leaf:{5}
Failed connections' threshold (Γ)	10
Blocking time (Ψ)	5 seconds
Smoothing coefficient (α)	$\alpha \in \{0.1; 0.5; 0.9\}$
List removal threshold (Δ)	2
Observation window occupation	25%; 50%; 75%; 100%
Experimental runs	5

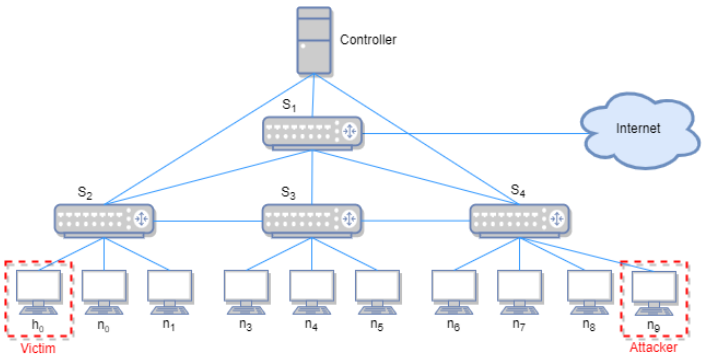


FIGURE 6 Network topology.

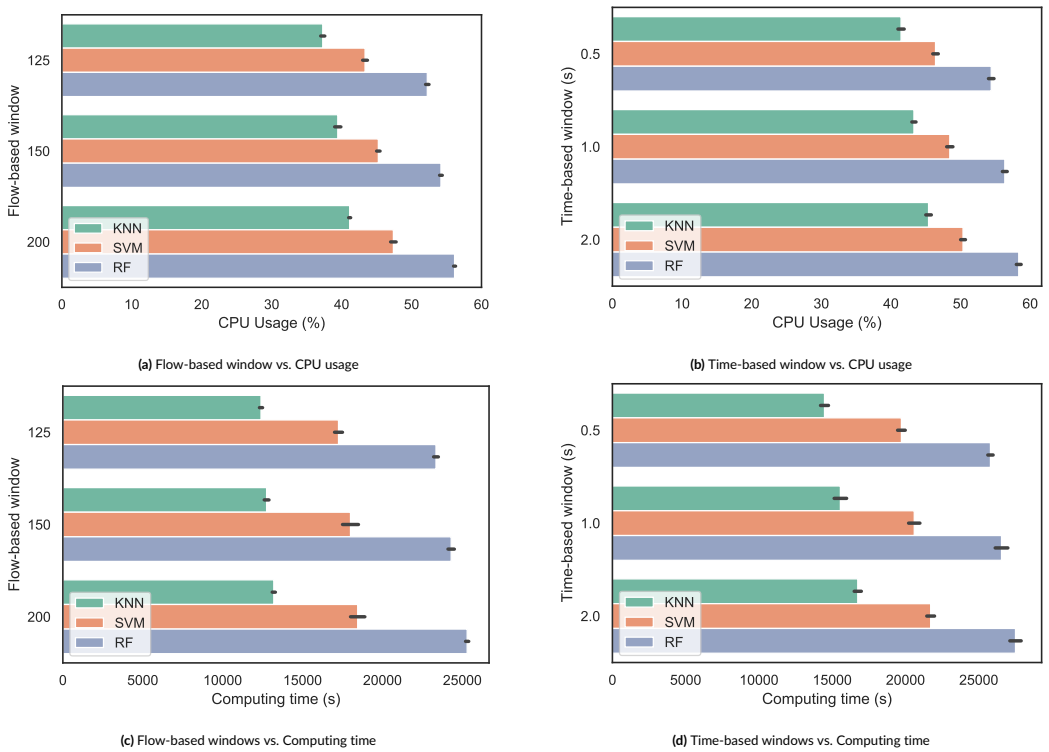
The bandwidth of each link is set to 100 Mbps. Recall that the DataPlane-ML algorithms for attack detection and mitigation are implemented in the data plane. Hence, the DataPlane-ML is implemented on SDN switches (S_1 to S_4). In the experiments, the attacker is assumed to have gained control of a legitimate host (n_9) to carry out the attack (SYN Flood), which is connected to the switch S_4. Note that the attack could have been directed from an external source. In either case, the spurious traffic will pass through the forwarding/routing devices where the DataPlane-ML is running. The victim (h_0) is a web server responsible for handling HTTP requests, connected to the switch S_2. Legitimate and spurious traffic are synthetically generated using the Scapy tool[51]. Legitimate traffic has a rate of 10 HTTPS request per second, while spurious traffic has a rate of 100 HTTP per second. The experiments run for 30 seconds. Spurious traffic is injected for 9 seconds (experimental time 13s up to 22s). The results are averaged over 5 experimental runs.

6.2 | Attack Detection Results

This section reports the experimental results for the attack detection phase using the metrics discussed in the previous subsection. We begin by evaluating ML training computational costs. Afterward, CPU usage, accuracy, and attack detection time are discussed, considering a scenario where the attacker sends numerous SYN packets with the spoofed IP address to the server.

Figure 7 presents the cross-validation CPU usage (7a and 7b) and cross-validation computing time (7c and 7d) for each ML algorithm. The figure shows results for time-base and flow-based windows. The results show that cross-validation CPU usage and cross-validation computing time increases with the number of flows in the observation window. Larger windows have more data to be processed, smoothly impacting these metrics. For instance, an observation window with 125 flows demands 40% of CPU usage with the KNN algorithm, while an observation window with 200 flows increases CPU usage by at most 5%. The CPU usage and computing time are higher for time-based windows, since it may hold more data than flow-based windows (discussed later).

The ML algorithm's complexity differences are noticeable in the experimental results. Among the ML algorithms, RF is the most expensive, both in terms of computing time and CPU usage, due to the number of subsets to look for the best split of a node (`max_features`), the number of trees created in the forest (`n_estimators`), processing of different trees according to the established depth level (`max_depth`), the number of samples needed to split an internal node (`min_samples_split`) and the minimum number of samples needed to be in a leaf node (`min_samples_leaf`). Among these parameters, `n_estimators` and `max_depth` are those that have the greatest impact on the model[49]. The ML parameters for each ML algorithm are listed in Table 1. Different SVM algorithms may use distinct kernel functions, including linear, polynomial, radial basis function (RBF), and sigmoid[52]. In particular, the RBF kernel function has been widely used given its learning ability and applicability to both low and high-dimensional spaces as well as to small and large samples[52]. In this work, the RBF kernel function is used. The RBF kernel requires optimization of a real-valued parameter `Gamma` > 0, which defines how far the influence of a single training example reaches. The larger `Gamma` is, the closer other examples must be to be affected. Similarly to `Gamma`, the parameter C must be carefully chosen, as it trades off of training examples against the simplicity of the decision surface. Finally, KNN presented the lowest cross-validation CPU usage and computing time, as shown in Figure 7. The reduced KNN costs are due to: (i) low complexity in the search for the nearest neighboring points, and (ii) the reduced number of attributes used to define the best classification model. The evaluated KNN attributes are the number of neighbors `n_neighbors` and their weight. In this work, the optimizing weight parameters have used both the `uniform`, which uses the same weight for all neighbors and the `Euclidean distance`, which assigns higher weights to nearest neighbors. Considering flow-based observation windows, KNN requires, on average, 13% and 27% less CPU than SVM and RF, respectively. While at the time-based observation windows, KNN requires, on average, 10% and 23% less CPU than SVM and RF, respectively. In terms of computing time, considering flow-based windows, the KNN algorithm was, on average, 28% and 47% faster than SVM and RF, respectively. Considering time-based windows, the KNN algorithm was, on average, 24% and 41% faster than SVM and RF, respectively.



(a) Flow-based window vs. CPU usage

(b) Time-based window vs. CPU usage

(c) Flow-based windows vs. Computing time

(d) Time-based windows vs. Computing time

FIGURE 7 Cross-validation CPU usage and computing time.

Once the model is trained, the next step is to evaluate the accuracy of the model to detect an attack. Entropy-based [6] methods are commonly used to detect networks traffic anomalies. Hence, entropy will be used to compare the ML algorithm's accuracy. Recall that the experiment lasts about 30 seconds and the SYN-Flood attack starts 13 seconds from the beginning of the experiments, lasting for about 9 seconds. Figure 8 shows the accuracy (8a and 8b), CPU usage (8c and 8d) and attack detection time (8e and 8f) for KNN, SVN, RF and entropy. In flow-based windows, there was a gradual increase in accuracy, CPU consumption, and detection time as the number of flows in a window increases. On the other hand, the time-based window accuracy decreases by about 0.77% when the window time increases from 0.5s to 2.0s. Interestingly, with a substantial increase in the number of flows in time-base windows, the ML algorithm's accuracy decreases. This degradation was caused by a strong dispersion in the data, making its separation difficult, resulting in a lower accuracy by the ML algorithms. Entropy, on the other hand, is able to better exploit the increasing number of flows in time-based windows to improve accuracy. Overall, ML algorithms outperformed the entropy-based attack detection method, with accuracy between 97% and 98.5% in the evaluated settings.

As shown in Figures 8c and 8d, the attack detection CPU demands were about 25% to 30%. As in the training results, RF still demands more CPU as compared to KNN, SVN, and entropy. On the other hand, the RF algorithm outperforms KNN, SVM, and entropy, showing an increase of 0.85% in detection accuracy, as it is able to perform a better correlation between attributes and trees created. However, it has a higher computational cost than the others, on average using up to 28% of the CPU resources, while SVM and KNN used, on average, 26.2% and 23.3%, respectively. In general, entropy consumed $\approx$ 13% more CPU than KNN due to its probability analysis, which requires more time to collect and analyze data. Finally, in time-based windows, the algorithms are $\approx$ 16% slower to detect an attack compared to flow-based windows, as they retain a larger number of flows. The KNN algorithm was $\approx$ 10% faster than the others, due low complexity in the search for nearest neighbors. Entropy was the algorithm that took longer to detect any change in the network. Compared to entropy-based detection, ML techniques were, on average, 26% faster. The reason for this behavior is that entropy needs to calculate observed frequencies to calculate entropy values, since it does not perform the training and testing process to define the best classification model.

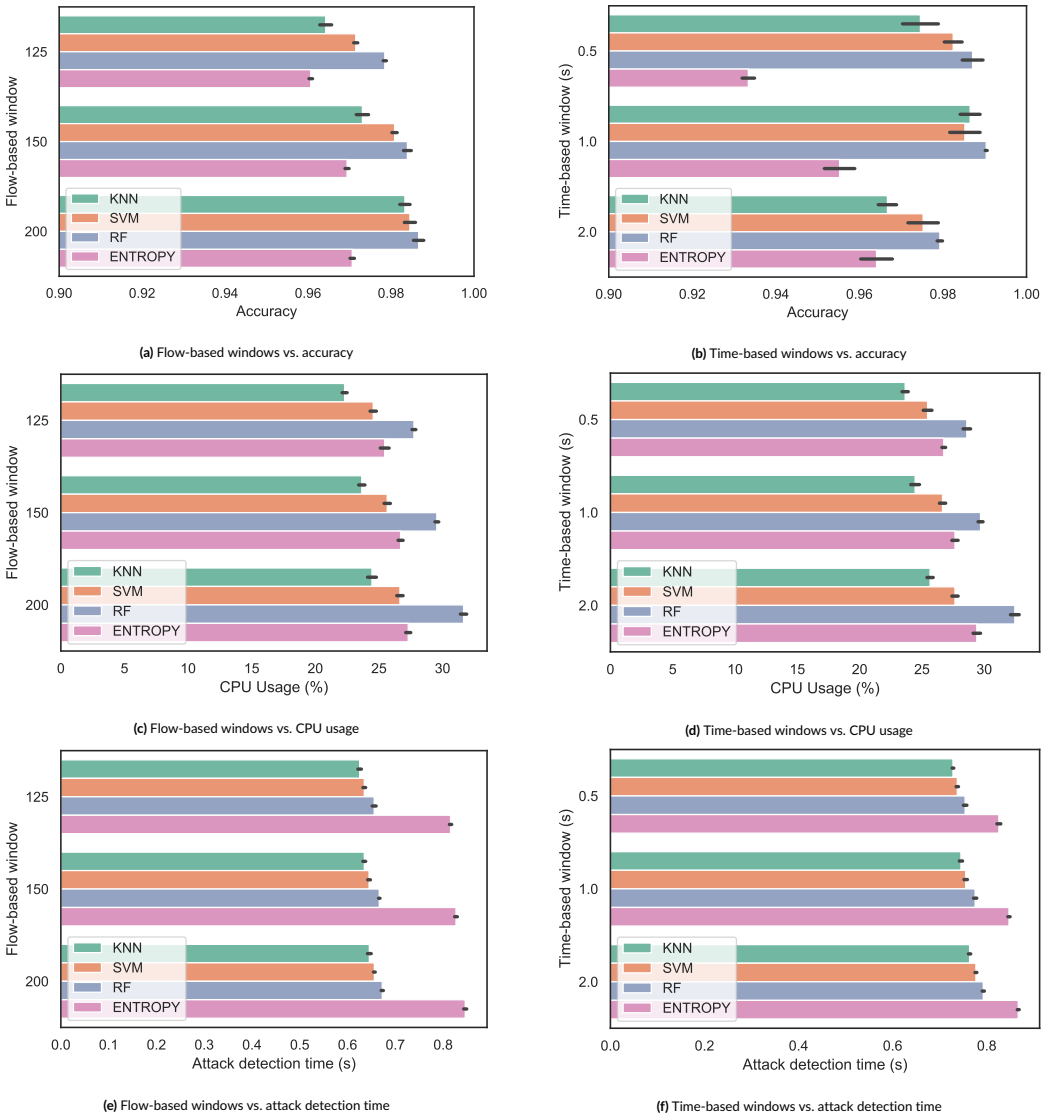
The number of packets present in time-based and flow-based windows when an attack is detected can be seen in Figure 9. In SYN flood attacks, spurious flows contain 2 packets, namely the TCP SYN sent by the attacker and a TCP SYN-ACK send by the server as a reply. On the other hand, legitimate HTTP requests contain several packets (e.g., TCP connection establishment, HTTP request and reply, tear down of the TCP connection). Hence, the number of packets in each flow varies, as can be observed in Figure 9. During an attack, the number of packets present in time-based windows (9b) is, on average, 135% higher than the number of packets in flow-based windows (9a). This difference impacts the detection time, resource usage and detection accuracy. In summary, the choice of the window type and size must take into account the attack volume and data dispersion. The correct choice of these parameters can lead to better accuracy, lower resource consumption and faster attack detection.

6.3 | Mitigation Results

This section reports the experimental results for the mitigation phase. First, we evaluate the window size variability impact on the node's reputation, including the computational costs to obtain the node's local trust (C_{ij}) and global trust (T_{ci}) thresholds. Next, we discuss the impact of the smoothing coefficient (α) in defining the global trust threshold (T_g) used to establish node priority.

The attack detection phase requires a considerable amount of information to detect changes in node behavior. Node reputation, on the other hand, can be computed based on the node's connections. However, computing the node reputation on connection-basis can impose a higher overhead on the P4-switches. As an alternative, we consider computing the nodes' reputation based on partial observation window information. Figure 10a show average time to compute the local trust value (C_{ij}) with distinct percentage of the observation window's occupation. The local trust (Eq.(2) is computed using four different observation window occupation, 25%, 50%, 75%, and 100%. Here, we consider a flow-based window, which has a size of 150 flows. The figure shows that the local trust computation time increases linearly with the percentage of the observation window. With an observation windows $w_i, i \geq 0$ set to 25% ($\approx$ 38 flows), the computation time takes about 30ms while a full window (w_i with 150 flows) takes 120ms. A more conservative reputation system may require the local trust to be computed at shorter intervals. This, in turn, would accelerate changes in the node's reputation and priority lists. Figure 10b shows the CPU usage to compute the local trust with varying observation window occupation percentages. When executing the algorithm on a 25% window occupation, the process is repeated 4 times to reach 100%, totaling 24.5% of CPU consumption, generating an increase of 21% when compared to a window of 100%. Thus, the CPU usage is slightly higher with lower window occupation percentages.

The local trust value is used to compute the node's global trust T_{ci}, defined in Eq.(3)), which is then used to assign it to the corresponding priority list accordingly to the Algorithm 2 rules. The T_{ci} value is directly influenced by the observation window occupancy, as can be observed in Figure 11. As the observation window occupancy increases, the variation in global trust values reduces. The global trust value is about 0.6715, when w is set to 25% and about 0.6048 when w is set to 100%. The higher variability in the nodes' global trust, in turn, impacts the priority lists, imposing higher costs for list management.



(a) Flow-based windows vs. accuracy

(b) Time-based windows vs. accuracy

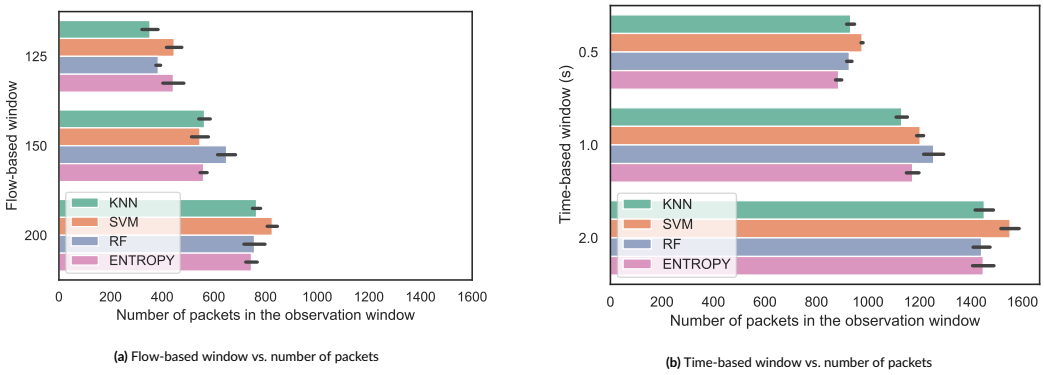
(c) Flow-based windows vs. CPU usage

(d) Time-based windows vs. CPU usage

(e) Flow-based windows vs. attack detection time

(f) Time-based windows vs. attack detection time

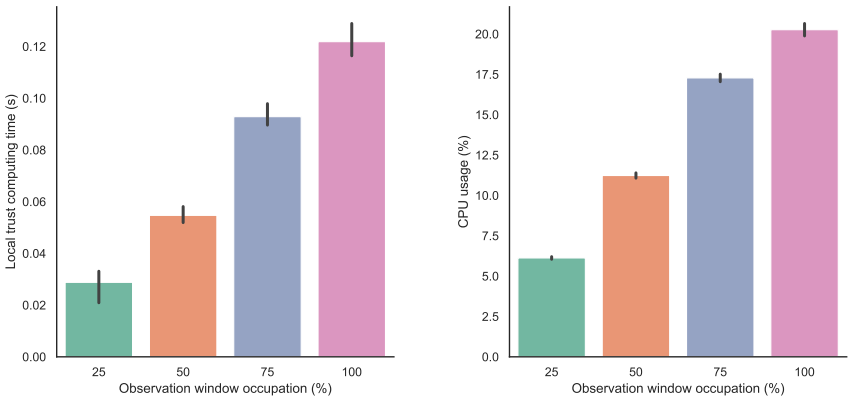
FIGURE 8 Detection accuracy, CPU usage and attack detection time.



(a) Flow-based window vs. number of packets

(b) Time-based window vs. number of packets

FIGURE 9 Average number of packets in the observation window.



(a) Computing time **(b)** CPU usage

FIGURE 10 Local trust (Eq.(2)) computing time and CPU usage.

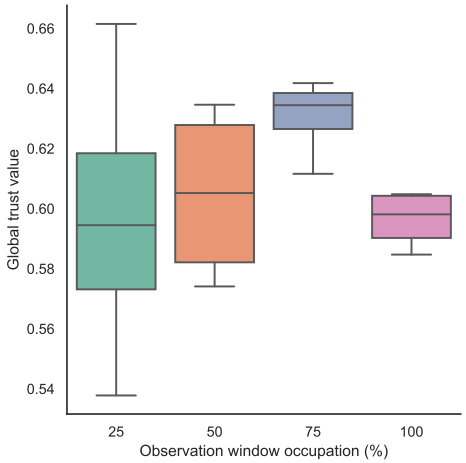
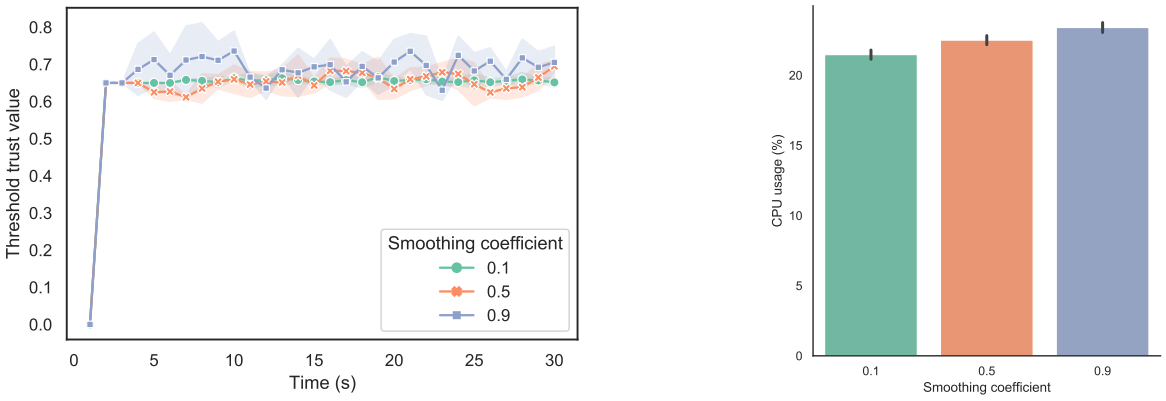


FIGURE 11 Observation window size impact on the node's global trust threshold (T_{ci}).

The impact of the smoothing coefficient (α) in defining the global trust threshold (T_g) is discussed next. In the experiments, α assumes the following values: 0.1, 0.5 and 0.9. The observation window occupancy (w) was set to 75% ($\approx$ 113 flows) and the KNN detection algorithm is used. The total simulation time was set to 30s, where the SYN-Flood attack begins after 13s of the beginning of the simulation, lasting for 9s. Figure 12a shows global trust threshold changes in 30s experimental time. The threshold is calculated according to Eq. 5. Note that, in the initial moments, the global trust threshold (T_g) expresses similar values independently of the coefficient used. As the simulation progresses, the coefficient influence on the global trust threshold becomes noticeable. From the moment that the local trust is obtained, it is possible to observe the variations on the global threshold. When a lower smoothing coefficient (α) is used, the global threshold tend to be more stable, as past information has a higher weight. Conversely, assigning a large value to the smoothing coefficient (α) makes the threshold react to recent node activity, this behavior is observed for the coefficient $\alpha = 0.9$. The smoothing coefficient impacts the priority list management, reflecting on the CPU usage, as can be observed in Figure12b. For example, for a coefficient $\alpha = 0.9$, there is an increase in CPU usage of about 8% as compared to the coefficient 0.1. Therefore, it is necessary to observe these aspects when choosing the smoothing coefficient for the global trust threshold for the priority list.

Next, we evaluate the smoothing coefficient as its impact on the priority list behavior. Recall that the DataPlane-ML is implemented on SDN switches (S_1 to S_4). The attacker is assumed to have gained control of a legitimate host (n_9) to carry out the attack (SYN Flood), which is connected to the switch S_4. For each second, we register the number of packets handled by the S_4 switch and the priority list that the sender node belongs to. Figure 13 shows the number of packets handled by the S_4 switch and the corresponding list that the sender nodes are associated with at that moment. Note that the number of packets for each list varies, as HTTP requests may contain multiple TCP segments (TCP connection establishment, request, response, termination and retransmission). The attacker in the initial moments (1s – 12s) has a behavior similar to the legitimate nodes,



(a) Observation window occupation 75%

(b) Observation window occupation 75%

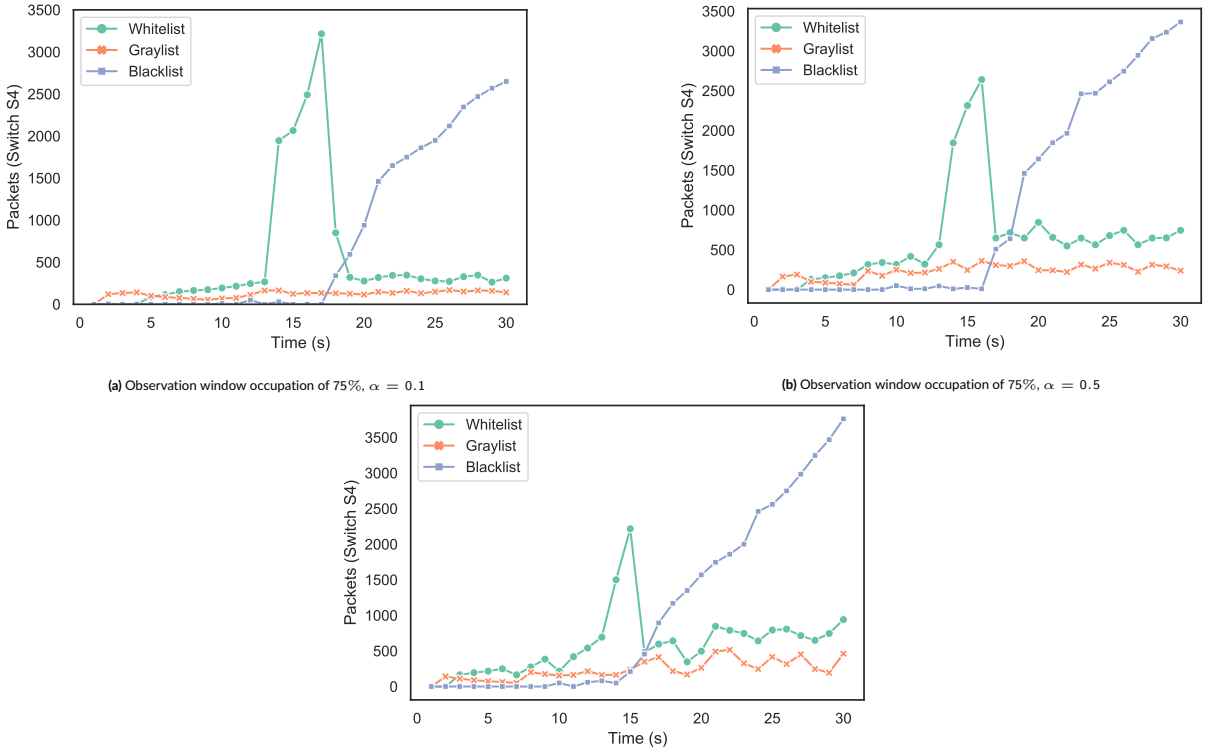
FIGURE 12 (a) Global trust threshold (T_g) changes in a 30s experimental time with distinct smoothing coefficients and, (b) the average CPU usage.

which allows it to ingress the priority lists and consequently access the highest priority list (whitelist). As the nodes are promoted to the whitelist, the percentage of packets from graylist nodes reduces. This behavior is evident in the figure, experimental time 2 to 4 seconds, independently of the smoothing coefficient value. Note that the attack starts in time 13 seconds, this moment there is an abrupt increase in the number of packets in the whitelist. Such behavior refers to the large volume of SYN requests received by the S_4 switch in a short time space. As the number of allowed failed connections' threshold (Γ) exceeds, the suspicious node is removed from the priority list (WL or GL) added to the blacklist and remains there for Ψ blocking time. This, in turn, causes an increase of the packets in the blacklist, as can be observed in the figure, experimental time 14 to 17 seconds. A lower smoothing coefficient delays the priority list's changes, as the threshold assigns a greater weight to past information. For example, with $\alpha = 0.1$, the suspicious node is blocked at time 17s, that is, 4s after the attack started. On the other hand, with $\alpha = 0.9$, the suspicious node is blocked at time 15s, that is, 2s after the attack started. Hence, setting α to higher values, allows the mitigation algorithms to block suspicious nodes faster. This, however, has an impact on CPU usage, as discussed above. Once the attack is detected, the number of packets from blacklist sources continue to rise, while the whitelist and graylist returns to values similar to those before the attack, after a few seconds later. That is, DataPlane-ML blocked the suspicious node and allowed trustworthy nodes to continue communicating with the destination server.

The accuracy of the mitigation mechanism is depicted in Figure 14 for the following intervals: prior to the attack (1s – 12s), during the attack (13s – 22s) and after the attack (23s – 30s). The accuracy is computed for distinct smoothing coefficient (α) values. Different α values may affect the accuracy, as can be observed even for intervals in which there is no presence of spurious packets. Thus, large value to the coefficient promotes a large number of removals of nodes from the lists, due to the weight given to the most recent information, which can lead to the blocking of a legitimate node if it exceeds the number of removals allowed for the lists (Δ), impacting the accuracy of mechanism. For example, for the coefficient $\alpha = 0.9$, there is an average reduction of 2% in precision when compared to the coefficient $\alpha = 0.1$, considering the intervals (before and after the attack). Note that there is an accuracy reduction of 3% during the attack as compared to the interval before the attack. This reduction is related to the volume of packets present in the priory lists together with the applied coefficient and the number of removals the nodes for the lists. Note that after the attack, the precision increases being similar to before the attack. The reason for this increase is that the nodes in the blacklist are eventually unblocked. Note that the blacklist could impose a more severe penalty for recurring malicious nodes, so that such nodes would be permanently blocked. Overall, the accuracy of the mechanism is over 95% for the experimental interval (1s – 30s). The obtained accuracy level shows that the proposed mitigation mechanism is efficient to block spurious traffic while maintaining the connectivity of legitimate nodes.

7 | CONCLUSION

This paper presented and analyzed DataPlane-ML, a solution for detecting and mitigating SYN Flood attacks on Software Defined Networks. DataPlane-ML, as the name suggests, operates on the SDN data plane and uses ML-based methods to detect traffic behavior deviation that could characterize an attack. Also, DataPlane-ML encompasses a reputation algorithm to establish trust between nodes, allowing the identification and restricting malicious nodes from accessing networks resources. In the evaluated scenarios, and compared to statistical attack detection mechanisms, the DataPlane-ML provided accuracy surpassing 98% while requiring $\approx$ 15% less CPU processing time. The DataPlane-ML mitigation uses a reputation mechanism to identify and reward trustworthy nodes. Nodes whose behavior is considered inadequate or suspicious are moved to low priority lists and may be temporarily blocked. The reputation mechanism uses a smoothing coefficient that can be adjusted to satisfy network



(a) Observation window occupation of 75%, $\alpha = 0.1$

(b) Observation window occupation of 75%, $\alpha = 0.5$

(c) Observation window occupation of 75%, $\alpha = 0.9$

FIGURE 13 Number of packets according to the global trust threshold.

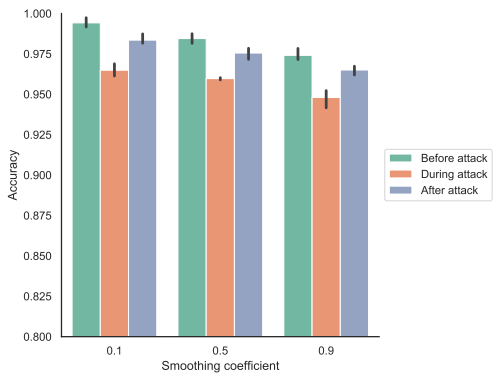


FIGURE 14 DataPlane-ML mitigation accuracy using an observation window occupation of 75%.

and application requirements. In the evaluated settings, the DataPlane-ML mitigation strategy managed to preserve legitimate traffic, achieving an average accuracy of 95%. The above results show that the proposed DataPlane-ML can improve the attack detection accuracy without imposing a higher burden on the P4-switch. Also, as the DataPlane-ML works at the data plane, there is no need for additional controller-switch communication, thus reducing the delay to take proper actions. In future work, we plan to improve DataPlane-ML to detect other types of attacks and evaluate time needed to update the models for the parameters used in this work.

ACKNOWLEDGEMENTS

This work was partially supported by the University of Brasília and Fundação de Apoio a Pesquisa do Distrito Federal.

CONFLICT OF INTEREST

The authors declare that they have no conflict of interest.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available on request from the corresponding author.

References

1. Kreutz D, Ramos F, Veríssimo P, Rothenberg C. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76* 2014. doi: 10.1109/JPROC.2014.2371999
2. Foundation ON. OpenFlow Switch Specs 1.0. *Open Network Foundation* 2009; 1: 1-44.
3. Feinstein L, Schnackenberg D, Balupari R, Kindred D. Statistical approaches to DDoS attack detection and response. In: Proceedings DARPA Information Survivability Conference and Exposition. ; 2003: 303-314
4. J.J.C Gondim RA. Mirror saturation in amplified reflection DDoS. *JNIC-Jornadas Nacionales de Investigación en Ciberseguridad* 2019.
5. Bani-Hani R, Al-Ali Z. SYN Flooding Attacks and Countermeasures: A Survey. In: International Conference on Information and Communication Security (ICICS). ; 2013.
6. Tritilanunt S, Sivakorn S, Juengjincharoen C, Siripornpisan A. Entropy-based input-output traffic mode detection scheme for DoS/DDoS attacks. In: International Symposium on Communications and Information Technologies (ISCIT). ; 2010: 804-809
7. Sahoo KS, Tripathy BK, Naik K, et al. An Evolutionary SVM Model for DDOS Attack Detection in SDN. *IEEE Access* 2020; 8: 132502-132513. doi: 10.1109/ACCESS.2020.3009733
8. Ye J, Cheng X, Zhu J, Feng L, Song L. A DDoS Attack Detection Method Based on SVM in Software Defined Network. *Security and Communication Networks* 2018; 2018(8).
9. Lapolli AC, Marques JA, Gaspary LP. Offloading Real-time DDoS Attack Detection to Programmable Data Planes. In: IFIP/IEEE Symposium on Integrated Network and Service Management (IM). ; 2019.
10. Dimolianis M, Pavlidis A, Maglaris V. A Multi-Feature DDoS Detection Schema on P4 Network Hardware. In: Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN). ; 2020: 1-6
11. Carvalho R, Costa L, Bordim J, Alchieri E. New Programmable Data Plane Architecture Based on P4 OpenFlow Agent. In: Springer Int. Publishing; 2020: 1355–1367.
12. Shin S, Yegneswaran V, Porras P, Gu G. AVANT-GUARD: Scalable and Vigilant Switch Flow Management in SDN. In: ACM SIGSAC Conference on Computer. ; 2013: 413–424
13. Ambrosin M, Conti M, Gaspari FD, Poovendran R. LineSwitch: Tackling Control Plane Saturation Attacks in Software-Defined Networking. *IEEE/ACM Transactions on Networking* 2017; 25(2): 1206-1219. doi: 10.1109/TNET.2016.2626287
14. Bosshart P, Daly D, Gibb G, et al. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 2014; 44(3): 87–95. doi: 10.1145/2656877.2656890
15. Fernandes LB, Camargos L. Bandwidth throttling in a P4 switch. In: 2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). ; 2020: 91-94
16. Foundation AS. Apache Thrift.; 2017.
17. Carvalho RN, Costa LR, Bordim JL, Alchieri E. DoSSec: A Reputation-Based DoS Mitigation Mechanism on SDN. In: Barolli L, Woungang I, Enokido T., eds. *Advanced Information Networking and Applications*Springer International Publishing. ; 2021; Cham: 757–770.

18. Carvalho RN, Costa LR, Bordim JL, Alchieri EAP. Detecting DDoS Attacks on SDN Data Plane with Machine Learning. In: 2021 Ninth International Symposium on Computing and Networking Workshops (CANDARW). ; 2021: 138-144

19. Costa Cordeiro dWL, Marques JA, Gaspary LP. Data Plane Programmability Beyond OpenFlow: Opportunities and Challenges for Network and Service Operations and Management. *Journal of Network and Systems Management* 2017; 25(4): 784–818. doi: 10.1007/s10922-017-9423-2

20. Bosshart P, Daly D, Gibb G, et al. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 2014; 44(3): 87–95. doi: 10.1145/2656877.2656890

21. Behal S, Kumar K. Detection of DDoS attacks and flash events using novel information theory metrics. *Computer Networks* 2017; 116: 96-110. doi: https://doi.org/10.1016/j.comnet.2017.02.015

22. Tajer J, Makke A, Salem O, Mehaoua A. A comparison between divergence measures for network anomaly detection. In: International Conference on Network and Service Management (ICNSCM). ; 2011: 1-5.

23. Müller A, Guido S. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media . 2016.

24. Aggarwal CC. *Machine Learning for Text*. Springer Publishing Company, Incorporated. 1st ed. 2018.

25. Wu X, Zhang S. Synthesizing high-frequency rules from different data sources. *IEEE Transactions on Knowledge and Data Engineering* 2003; 15(2): 353-367. doi: 10.1109/TKDE.2003.1185839

26. Coretes C, Vapnik V. Support-vector networks. *Machine Learning* 1995; 28: 273-297. doi: https://doi.org/10.1007/BF00994018

27. Ns A. Time complexity analysis of support vector machines (SVM) in LibSVM. *International Journal of Computer Applications* 2015; 128: 975-8887. doi: 10.5120/ijca2015906480

28. Breiman L. Random Forests. *Machine Learning* 2001; 45.

29. Louppe G. Understanding Random Forests: Theory to Practice.; 2015.

30. Imran M, Durad H, Khan F, Derhab A. Toward an optimal solution against Denial of Service attacks in Software Defined Networks. *Future Generation Computer Systems* 2018; 92. doi: 10.1016/j.future.2018.09.022

31. Fichera S, Galluccio L, Grancagnolo SC, Morabito G. OPERETTA: An OPEnflow-based REmedy to mitigate TCP SYNFLOOD Attacks against web servers. *Computer Networks* 2015: 89-100. doi: 10.1016/j.comnet.2015.08.038

32. Kumar P, Tripathi M, Nehra A, Conti M, Lal C. SAFETY: Early Detection and Mitigation of TCP SYN Flood Utilizing Entropy in SDN. *IEEE Transactions on Network and Service Management* 2018; 15(4): 1545-1559.

33. Kim D, Dinh PT, Noh S, Yi J, Park M. An Effective Defense Against SYN Flooding Attack in SDN. In: International Conference on Information and Communication Technology Convergence (ICTC). ; 2019: 369-371

34. Lin TY, Wu JP, Hung PH, et al. Mitigating SYN flooding Attack and ARP Spoofing in SDN Data Plane. In: Asia-Pacific Network Operations and Management Symposium (APNOMS). ; 2020: 114-119

35. Kuerban M, Tian Y, Yang Q, Jia Y, Huebert B. FlowSec: DOS Attack Mitigation Strategy on SDN Controller. In: IEEE International Conference on Networking, Architecture and Storage (NAS). ; 2016: 1-2

36. Oo NH, Htein Maw A. Effective Detection and Mitigation of SYN Flooding Attack in SDN. In: International Symposium on Communications and Information Technologies (ISCIT). ; 2019: 300-305

37. Chin T, Mountrouidou X, Li X, Xiong K. Selective Packet Inspection to Detect DoS Flooding Using Software Defined Networking (SDN). In: IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW). ; 2015: 95-99

38. Habibi Lashkari A. CICFlowmeter-V4.0 (formerly known as ISCXFlowMeter) is a network traffic Bi-flow generator and analyser for anomaly detection. https://github.com/ISCX/CICFlowMeter. *Research Gate* 2018. doi: 10.13140/RG.2.2.13827.20003

39. InMonCorp . sFlow-RT: Real-time analytics provides actionable intelligence to drive DevOps, Orchestration and SDN applications.; 2015.

40. Rahman O, Quraishi MAG, Lung CH. DDoS Attacks Detection and Mitigation in SDN Using Machine Learning. In: . 2642-939X. IEEE World Congress on Services (SERVICES). ; 2019: 184-189

41. Deepa V, Sudar KM, Deep P. Design of Ensemble Learning Methods for DDoS Detection in SDN Environment. In: IEEE International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN). ; 2019

42. Tuan NN, Hung PH, Nghia ND, Van Tho N, Phan TV, Thanh NH. A Robust TCP-SYN Flood Mitigation Scheme Using Machine Learning Based on SDN. In: International Conference on Information and Communication Technology Convergence (ICTC). ; 2019: 363-368

43. Kumar C, Kumar BP, Chaudhary A, et al. Intelligent DDoS Detection System in Software-Defined Networking (SDN). In: IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT). ; 2020: 1-6

44. Aiken J, Scott-Hayward S. Investigating Adversarial Attacks against Network Intrusion Detection Systems in SDNs. In: IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). ; 2019: 1-7

45. Khamaiseh S, Serra E, Li Z, Xu D. Detecting Saturation Attacks in SDN via Machine Learning. In: International Conference on Computing, Communications and Security (ICCCS). ; 2019: 1-8

46. Wang Y, Liu Y, Hu J, Zhang M, Wang X. Reputation and Incentive Mechanism for SDN Applications. In: International Conference on Mobile Ad-Hoc and Sensor Networks (MSN). ; 2018: 152-157.

47. Niemiec M, Jaglarz P, Jekot M, Chołda P, Boryło P. Risk Assessment Approach to Secure Northbound Interface of SDN Networks. In: International Conference on Computing, Networking and Communications (ICNC). ; 2019: 164-169.

48. Sepandar D. Kamvar HGM. The EigenTrust Algorithm for Reputation Management in P2P Networks. *WWW '03: Proceedings of the 12th international conference on World Wide Web* 2003(12): 640-651.

49. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 2011(12): 285-2830.

50. Team M. Mininet - An Instant Virtual Network on your Laptop (or other PC).; 2014.

51. Scapy . Packet crafting for Python2 and Python3.; 2008.

52. Smola A, Schölkopf B. A tutorial on support vector regression. *Statistics and Computing* 2004; 14: 199-222.