ORACLE: An Architecture for Collaboration of Data and Control Planes to Detect DDoS Attacks

Sebastián Gómez Macías
Universidad de Antioquia
sebastian.gomez3@udea.edu.co

Luciano Paschoal Gaspary
Universidade F. do Rio Grande do Sul
paschoal@inf.ufrgs.br

Juan Felipe Botero
Universidad de Antioquia
juanf.botero@udea.edu.co

***Abstract*—The possibility of programming the control and data planes, enabled by the Software-Defined Networking (SDN) paradigm, represents a fertile ground on top of which novel operation and management mechanisms can be fully explored, being Distributed Denial of Service (DDoS) attack detection based on machine learning techniques the focus of this work. To carry out the detection, this paper proposes ORACLE: cOllaboRation of dAta and Control pLanEs to detect DDoS attacks, an architecture that promotes the coordination of control and data planes to detect network attacks. As its first contribution, this architecture delegates to the data plane the extraction and processing of traffic information (with per flow granularity). This is done to ease the calculation and classification of the feature set used in the attack detection, as the needed flow information is already processed when it arrives at the control plane. Besides, as the second contribution, this architecture breaks the limitations to calculate some features that are not possible to implement in a traditional OpenFlow-based environment. In the evaluation of ORACLE, we obtained up to 96% of accuracy in the testing phase, using a K-Nearest Neighbors model.**

***Index Terms*—DDoS, Security, SDN, P4, Machine-Learning**

I. Introduction

A DDoS attack is considered a powerful weapon against Internet security due to its ability to bring down even the largest websites by overloading server resources [1]. In turn, Software-Defined Networking (SDN) is an emerging network paradigm that decouples the control and data planes where the network control is logically centralized and programmable. Considering the different proposed DDoS detection approaches, several leverage the SDN advantages: global network view and control plane programmability. Among these approaches, those based on Machine Learning (ML) algorithms are considered the best in terms of key metrics such as accuracy, detection speed, and reliability [2], [3].

A traditional ML-based detection system is mainly based on: i) a traffic information collecting mechanism, ii) a feature calculation module, and iii) a classification model; all of them implemented at the control plane. The collecting mechanism obtains traffic information from the packets that arrive at the SDN controller (*Packet_in*) and also from the statistics provided by OpenFlow (SDN southbound interface). The feature calculation module processes the collected information to build the set of descriptors, and, finally, the classification module classifies this set of features in search of possible ongoing DDoS attacks. However, in the implementation of these detection systems, we observe two main design challenges, which we enumerate and describe next.

978-3-903176-32-4

The **first challenge** is to prevent overloading the control plane due to the execution of tasks related to polling information from the network traffic, as well as the calculation and classification of the feature set. To carry out these tasks, the detection system has to store, update, and filter different information samples, which are sorted and processed per flow, connection, or source-destination pair. The execution of these tasks in the presence of large amounts of packets and flows can easily overload the control plane.

The **second challenge** lies in extracting, in real-time, the information needed to calculate the feature set selected as the best traffic descriptors. In an OpenFlow-based SDN environment, limited processing can be performed on a per-packet basis at the data plane. The extraction of sufficient traffic information with fine granularity needed to calculate a wide variety of features in real-time becomes impractical. Therefore, from a huge amount of traffic features proposed in the literature, just a specific group (e.g., flow's byte/packet count and duration) can be implemented.

In order to overcome these challenges, we propose ORACLE, a novel approach based on the cOllaboRation of dAta and Control pLanEs to detect DDoS attacks using the paradigm of programmable data planes. The proposed approach fully explores the programming of packet processing at a switch level. To program the data plane, we use the P4 programming language [4] together with the P4Runtime interface (to provide communication with the control plane). We propose a hash-based data structure to store customized per-flow information directly at the data plane device. We also propose a mechanism to periodically report such digested information from the data plane to the control plane, where the feature set is subsequently calculated and classified per each reported flow. As **main research contributions**, we highlight:

- An original architecture that delegates to the data plane the task of collecting, processing, and storing the required per-flow information. This facilitates the calculation and classification of the feature set at the control plane.
- A new approach that, based on data plane programmability and ingenious coordination of efforts between planes, allows for the extraction of a wide variety of features that are not implementable using OpenFlow.

II. Background and Related Work

In the last decades, a wide variety of traffic features have been investigated to detect anomalies. According to [5], these features are grouped into five categories: *packet level* (e.g., packet length mean), *flow level* (e.g., average packet length and packets per flow), *connection level* (e.g., advertised window sizes), *intra-flow level* (e.g., features based on inter-arrival times between packets of the same flow), and *multiflow*. Moreover, features at the flow or intra-flow level vary according to the flow type: *unidirectional* flow, comprised of packets sharing the same fields (source/destination IP, source/destination TCP/UDP ports, and transport protocol number); *bidirectional* flow, consisting of a pair of unidirectional flows that are going in opposite directions between the same source and destination IP addresses and ports; and *full* flow, a bidirectional flow captured during its entire lifetime.

Recent research efforts indicate that SDN, together with the OpenFlow protocol, made it comfortable implementing DDoS detection systems based on traffic feature classification. As an example, in [6], authors perform DDoS detection by polling OpenFlow statistics per second and per each unidirectional flow. The system stores several samples of each bidirectional flow (computed on the control plane from the unidirectional flows) to calculate features based on average and standard deviation (*std*) at flow level, reaching 96% of accuracy with a classifier based on Random Forest.

In [7], the authors propose a framework that comprises two operational phases at the control plane: 1) a lightweight processing phase that consists of monitoring the flows looking for entropy deviation anomalies; and 2) a heavyweight processing phase that is only invoked when an abnormal flow is detected. The heavyweight process uses the OpenFlow statistics (*packet_count*, *byte_count*, and *duration*) as the flow-level feature set to detect DDoS attacks, reaching an accuracy score of 88% with a Support Vector Machine-based model.

In the previously mentioned approaches and others such as [8], the implemented features are limited to a specific group at packet or flow level due to the restricted variety of statistics provided by OpenFlow. Besides, the control plane is overloaded with the deployment of multiple repetitive tasks, namely collecting, filtering, processing, calculating, and classifying the set of features per-flow or connection. This critical issue was identified by [9], which proposed an approach based on IP entropy anomaly detection that can be entirely offloaded to a programmable data plane. Although the solution leads to high accuracy and low resource usage, it only analyzes a single feature. Moreover, in a production scenario, it can result in a high false-positive rate, e.g., in the case of flash crowds.

Programming the data plane to detect DDoS attacks is a powerful trend to follow. Nevertheless, the primitives of the existing platforms (e.g., P4) are still very limited to implement robust solutions based on ML models directly at the devices. For this reason, **ORACLE takes advantage of the best of both trends** to propose a detection architecture that alleviates the load stress on the control plane and provides higher versatility in the use of features (benefiting from P4 to help in the systematic extraction of traffic statistics).

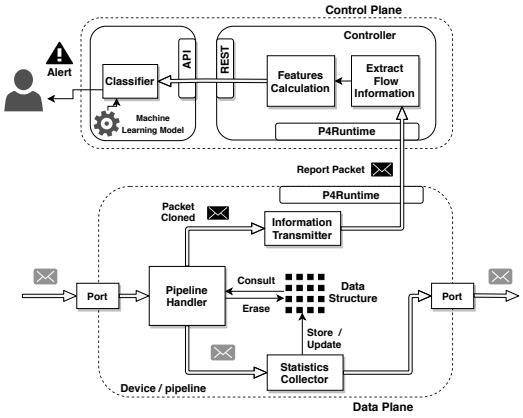

Fig. 1. ORACLE architecture

III. ORACLE: cOLlaboRation of dAta and Control pLanEs to detect DDoS attacks

A. Architecture

ORACLE refers to the manner how works the detection architecture proposed. This architecture is organized in two modules, namely *Data Plane* and *Control Plane*, which we show in Fig. 1 and describe next.

Data Plane: ORACLE relies on a time window-based implementation. During each window, the data plane performs two tasks simultaneously: it i) collects new flow information, and ii) reports flow information collected and processed in the previous time window to the control plane. The operation of the device's pipeline is composed of the following four components developed using P4 primitives such as registers, clone packet, hashing and basic binary operations (+,-,*).

- *Data Structure*: It is used for the statistics organization, management, and storage per unidirectional flow. To access them, the flow identifier (flow-id) is mapped, through a hash function, to the respective index (position where the statistics are stored) in the data structure.
- *Statistics Collector*: In charge of extracting and processing the needed information from the bit string of every packet traversing the device. With this information, the statistics of the flow associated with the packet are updated.
- *Pipeline Handler*: In charge of controlling the time window duration and validating if the data structure contains new flow information to report. If so, the incoming packet is cloned and forwarded to the Information Transmitter component along with the per-flow information that will be subsequently sent to the controller.
- *Information Transmitter*: This component transforms the cloned packet in a report packet that is forwarded to the control plane using the P4Runtime interface. The cloned packet is modified by eliminating its payload and adding the flow information as a new custom header.

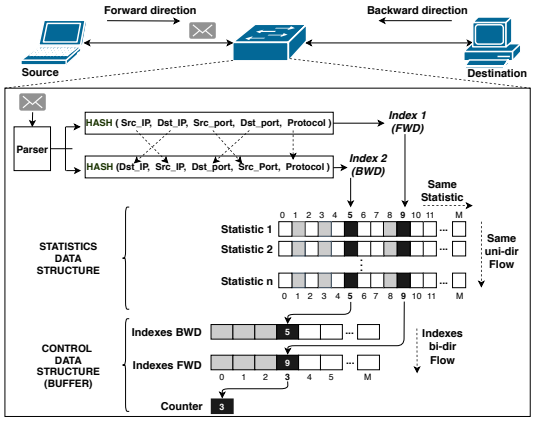

Fig. 2. Mechanism to store statistics per flow.

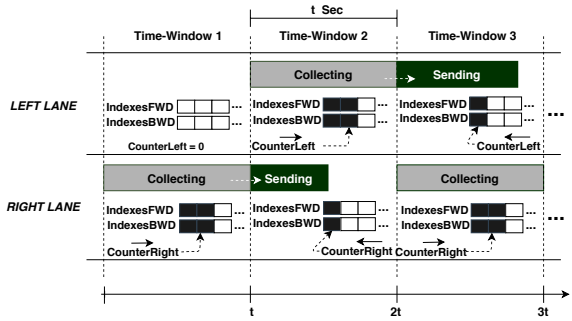

Fig. 3. Lane-based mechanism to handle time windows and buffers.

Control Plane: It is composed of an SDN controller and a traffic classification component connected via API-REST. To classify the reported flows, three components are involved:

- *Extract Flow Information*: This component extracts the custom header from the report packet and parses it to get the information of every flow sent by the data plane.
- *Feature Calculation*: It builds a tuple with the features set calculated with the flow information.
- *Classifier*: An already loaded ML model classifies every tuple received in DDoS or Benign class. When a possible attack is detected, an alarm is issued.

B. Data Plane Storage Mechanism

ORACLE collects and stores statistics of packets belonging to each flow on the programmable switches. It guarantees that flow statistics can be consulted and updated each time a new packet belonging to the flow arrives to the P4 device.

We use P4 registers to define a data structure to store flow statistics. A register is a fixed-size array that uses an index to point to the stored elements. To consolidate the structure, we use a set of such registers to build a matrix-like structure, where each row stores values of the same statistics belonging to different flows, and each column stores different statistics of the same unidirectional flow (see Fig. 2). To designate a unique index (column) to each unidirectional flow, we use the hash function with an input value (Flow-Id) that consists of source/destination IP, source/destination port, and transport protocol. The hash's output is the index mapped to the flow.

Figure 2 shows how per-flow statistics are stored and updated by the Statistics Collector component. Once a packet enters the P4 device, its bit string is parsed to extract and map each header field value to P4 variables. From these variables, the collector builds the 5-tuple (Flow-Id) to calculate the hash function. Once having the flow index, the flow statistics are updated in the data structure with the needed information that is obtained by processing the P4 variables.

As we are working with bidirectional flows and the method stores unidirectional flows, it is important to identify the pair of flows (indexes) defining the bidirectional flow within the data structure. To know the index of the opposite direction of the flow, we calculate again the hash function exchanging the order of the flow identifier fields (see Fig. 2). Hence, we define forward (FWD) as the direction of the first packet of the flow and backward (BWD), the opposite direction.

C. Data Plane Flow Reporting Mechanism

The Pipeline Handler component is in charge of reporting flow information to the control plane at the beginning of each time window. In order to forward the information of all bi-directional flows collected at the previous time window to the control plane, it is necessary to know the indexes that point to the specific flows within the data structure. To ease this process, each time a new flow is instantiated at the statistics data structure, both indexes (FWD and BWD) are stored in a new data structure composed of two registers and a counter (see Fig. 2). One register stores the indexes of the FWD unidirectional flows, and the other stores the indexes of the BWD flows. Hence, each column will contain the indexes of a different bidirectional flow and the counter will be just a reference to know the position of the last pair of stored indexes. This data structure works like a LIFO buffer: indexes are stored sequentially as the flows are instantiated. When the information is to be sent to the control plane, the indexes are consumed from the buffer in the reverse arrival order.

The implementation of the control data structure with a single buffer implies that the data plane interrupts the process of collecting information from new flows until the buffer is emptied. As the system must be able to perform both tasks (collecting and reporting flows to the control plane) simultaneously, we propose to abstract the idea of two independent monitoring "lanes", each one with its buffer. Therefore, is possible to delegate, over a same time window, one task to each lane, avoiding the process to be interrupted (see Fig. 3).

Using this strategy based on independent buffers to store the indexes of bidirectional flows, the Pipeline Handler component can validate, in each time window, if there are flow statistics available to forward to the control plane. If so, this component looks for per-flow statistics from the data structure, clones the number of incoming packets needed to transmit them, and delegates the transmitting process to the respective component. The Information Transmitter component embeds the information of each flow into a custom header that should

TABLE I
SELECTED FEATURE SET

#	Feature name	Description	Level / Type
F1	FlowDuration	Flow duration in microseconds	Flow / (Bidirectional)
F2	FlowIATStd	Standard deviation of packet inter-arrival times of the same flow.	Intra-Flow / (Bidirectional)
F3	avgPacketSize	Average of packet payload sizes of the same flow.	Flow / (Bidirectional)
F4	BwdPktLenStd	Standard deviation of packet payload sizes of the same flow in the backward direction	Intra-Flow / (Unidirectional)

have a size (number of fields) as big as the network MTU value to avoid injecting too many report packets into the network.

IV. DETECTION MECHANISM IMPLEMENTED ON ORACLE

As proof of concept, we implemented a DDoS attack detection system that uses the feature set listed in Table I. These features are recommended in [10], where the authors used the RandomForestRegressor technique to select, among 80 features, a small set considered the most adequate for DDoS attack detection. This set is made up of two features at flow level (F1 and F3) and two at intra-flow level (F2 and F4). On the other hand, there are features that are calculated considering the statistics in both flow directions (bidirectional), and one of them (F4) considering only the BWD direction.

In Table I, F2, F3, and F4 are based on average (avg) and standard deviations (std) operations, calculated using Eq. 1 and 2, respectively. x_i corresponds to an information unit obtained from packet i, which belongs to a group of n packets of the same flow. The meaning of x_i changes depending on the feature to calculate; for F3-F4, it corresponds to the size in bytes of the packet's payload i, and for F2, it corresponds to the packet inter-arrival time (IAT) that is calculated subtracting the timestamps of the packets i and $i-1$.

$$avg = \overline{x} = \frac{\sum_{i:1}^{n} x_i}{n} \tag{1}$$

$$std = \sqrt{\frac{\sum_{i:1}^{n} (x_i - \overline{x})^2}{n-1}} = \sqrt{\frac{\sum_{i:1}^{n} x_i^2 - 2\overline{x}\sum_{i:1}^{n} x_i + n\overline{x}^2}{n-1}} \tag{2}$$

In an SDN/OpenFlow environment, the features at flow level (F1 and F3) can be calculated from OpenFlow statistics. The real challenge is to calculate the intra-flow features (F2 and F4) because: i) OpenFlow does not permit to know the packet IAT, and ii) to calculate the *std* of an intra-flow measure, every x_i value is required at the control plane to subtract the mean from it when the sum is being computed (see Eq. 2, left side) and OpenFlow does not permit to get every value.

ORACLE allows overcoming these limitations as it stores individual information for each packet, even IAT values, and reports it periodically to the control plane. However, this may result in high latency and storage/processing resource consumption. A more ambitious solution would be to calculate these features directly on the data plane and avoid sending every x_i. This solution is not possible because programmable data planes are usually limited in the available mathematical operations (e.g., division and square root are not allowed).

TABLE II
INFORMATION PER-FLOW REPORTED BY THE DATA PLANE

#	Feature / Statistic	Description
F1	FlowDuration	Flow duration in microseconds
S2	TotFwdPkt	Total number of packets in the FWD direction
S3	TotBwdPkt	Total number of packets in the BWD direction
S4	TotLenFwdPkt	Aggregation of packets' payload size in FWD direction $(\sum x_i)$
S5	TotLenBwdPkt	Aggregation of packets' payload size in BWD direction $(\sum x_i)$
S6	TotLenBwdPktSqrt	Aggregation of packets' payload size square in BWD direction $(\sum x_i^2)$
S7	IATTotal	Aggregation of the packet inter-arrival times $(\sum x_i)$
S8	IATTotalSqrt	Aggregation of the packet inter-arrival times square $(\sum x_i^2)$

Therefore, we propose the following alternative. The traditional *std* (Eq. 2) is transformed using arithmetic properties to have the right side representation. Now, this new representation depends on two sums: the aggregation of values $(\sum x_i)$, and the aggregation of square values $(\sum x_i^2)$. Using the data plane mechanisms proposed in ORACLE, we calculate, directly at the data plane, the result of both sums per each flow, as the ongoing packets traverse the data plane device. Therefore, when these aggregation results are reported together with the packet counters to the control plane (see Table II), the controller has only to replace them in Eq. 1 and 2 to calculate F2, F3, and F4 per each flow in an easy and fast way, as specified below. Regarding F1, it is calculated directly in the data plane and reported to the control plane together with the statistics.

- *FlowIATStd (F2):*

$$n = S2+S3-1 \quad ; \quad \overline{x} = \frac{S7}{n} \quad ; \quad std = \sqrt{\frac{S8 - 2*\overline{x}*S7 + n\overline{x}^2}{n-1}} \tag{3}$$

- *avgPacketSize (F3):*

$$n = S2 + S3 \quad ; \quad \overline{x} = \frac{S4+S5}{n} \tag{4}$$

- *BwdPktLenStd (F4):*

$$n = S3 \quad ; \quad \overline{x} = \frac{S5}{n} \quad ; \quad std = \sqrt{\frac{S6 - 2*\overline{x}*S5 + n\overline{x}^2}{n-1}} \tag{5}$$

V. EVALUATION

A. Experimental Setup and Methodology

We replicated the packet trace used in [10] over a real scenario. This experiment consisted of a 20 minute-long DDoS attack generated from different devices located in an external network, trying to deny the service provided by a web service located in the DMZ of a local network. This trace, which has traffic captured for 90 minutes, is composed of 762,973 DDoS packets and 1,382,900 benign packets, stored in a PCAP file.

Fig. 4 shows the implemented scenario. The data plane, emulated using Mininet, consists of: i) a BMV2 virtual switch with P4 support (in charge of executing the data plane strategy); ii) a host responsible for reproducing the traffic trace

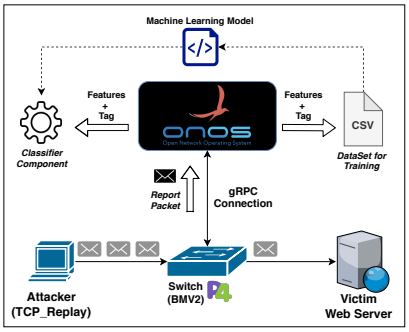

Fig. 4. Experimental setup

using the TCPReplay tool (representing the external network where the DDoS attack is originated); and iii) a victim host (Web Server), where all replicated traffic is forwarded to. In turn, the control plane is composed by the ONOS controller, which is connected to the BMV2 switch using the P4Runtime interface. This plane has a classifier component, which loads a previously trained ML-model. This scenario was implemented on a Core i5 5200 computer, with 12 GB of RAM and the Ubuntu 18.04 LTE operating system.

The implementation of our proposed system was exposed to two different workloads, one for the training phase of the ML model (WL1) and another for the testing phase of the same model (WL2). To create both workloads, the pcap file was divided into 18 sets of 5 minutes each, labeled from 1 to 18 considering their order in the trace. The sets with odd numbering were put together to build WL1. Then, the sets with even numbering were also joined to build WL2.

Before training the classification model, we had to build a database containing flow information. To create it, an ONOS application was developed to store, inside a .csv file, the feature set, and the respective tag (class: "Benign", or "DDoS") for every flow reported by the data plane (see Fig 4). These tags, needed to train the supervised model, were obtained directly from the packets belonging to WL1. They were previously marked (00: Benign, 11: DDoS) by modifying the last two bits of the IPv4 ToS header field.

Besides focusing on training the best ML model to detect DDoS attacks, we also decided to investigate how the detection system could be affected by different time window values at the data plane. For this assessment, we built a different training dataset per each time window value. Table III shows the characterization of the training datasets constructed considering different time window values (5, 20, 40, and 60 seconds).

TABLE III
TRAINING DATASET CHARACTERIZATION

Dataset per Time Window Duration				
Tag / Time Window	**5s**	**20s**	**40s**	**60s**
Benign Flows	61,736	40,462	31,894	29,670
DDoS Flows	49,012	45,688	44,511	43,255
Total Flows per Dataset	110,748	86,150	76,405	72,925

After creating the mentioned databases, the ML models are trained in an offline manner. We used and compared two ML classifiers per each time window: RF and K-Nearest Neighbor (KNN) (8 models in total). To build the models, we implemented cross-validation of 10 iterations using the datasets shown in Table III. In each iteration, a grid search was performed to find the best set of hyperparameter values that lead to the most accurate models. After ten iterations, the best model was chosen (among the ten evaluated). Table IV shows the best set of hyperparameter values to train the best model found for each time window duration.

TABLE IV
BEST HYPERPARAMETERS / TRAINING TIME

Random Forest (RF)				
-	**M1: 5s**	**M2: 20s**	**M3: 40s**	**M4: 60s**
Max_depth	6	6	6	6
N_estimators	300	500	500	300
Training time [s]	332.8 ± 6	304.4 ± 8.1	**264.4 ± 20**	272.6 ± 20
K-Nearest Neighbor (KNN)				
-	**M5: 5s**	**M6: 20s**	**M7: 40s**	**M8: 60s**
# Neighbors	3	3	3	3
Training time [s]	59.8 ± 2	37.7 ± 1	36.3 ± 3	**26.7 ± 1**

In the testing phase, each model was pre-loaded into the classifier component. Then, the WL2 was replicated eight times per model for the online evaluation. In these experiments, all WL2 packets were also tagged to compare the tag of each flow with the classification result (in real-time). In this phase, both the feature set and the flow tag were not stored. Instead, they were sent to the classifier component.

The metrics used in the testing phase to evaluate the detection performance are based on the number of true positive (*TP*), true negative (*TN*), false positive (*FP*) and false negative (*FN*) classes (DDoS and Benign) obtained during the experiment. The metrics are introduced as follows:

- *Accuracy* is the number of correctly detected cases in the total flow sample, considering both classes.

$$Accuracy = \frac{TN+TP}{TN+TP+FP+FN} \quad (6)$$

- *Recall* represents the proportion of DDoS samples correctly detected in relation to the total DDoS flow samples. It is best known as the True Positive Rate (TPR).

$$Recall = TPR = \frac{TP}{TP+FN} \quad (7)$$

- *Selectivity* represents the proportion of Benign samples correctly detected in relation to the total Benign flow samples; also known as the True Negative Rate (TNR).

$$Selectivity = TNR = \frac{TN}{TN+FP} \quad (8)$$

- *F1_Score* can be seen as the weighted average of the precision and recall metrics values.

$$F1\_score = \frac{2TP}{2TP+FP+FN} \quad (9)$$

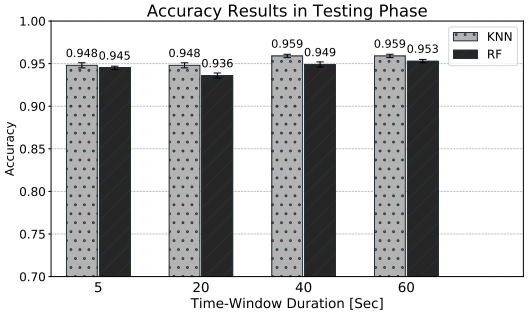

Fig. 5. Accuracy of the evaluated models as a function of window duration

B. Experimental Results

Table IV shows the average time to calculate the best set of hyperparameters and train the model with them. We notice that KNN is the best option when it comes to the performance of the training phase, as it is faster than RF.

Fig. 5 shows the detection accuracy results as a function of the time window duration (in the testing phase). It is possible to observe that the eight models trained lead to an accuracy higher than 93%, which is within the upper range seen in the literature [6], [7]. Besides, the KNN accuracy results are higher than the RF ones for all time windows. Nevertheless, the difference between models does not exceed 1.2%.

Increasingly better results are obtained as the time window duration increases, reaching the best values for time windows of 60 seconds. We highlight here a vital trade-off between accuracy and detection delay. A significant time window duration might lead to an intolerably high latency to detect and mitigate a DDoS attack. We advocate that as the detection rates vary just slightly (1.1% for KNN and 1.7% for RF), one should favor lower time window durations.

Table V shows the results obtained for the remaining metrics. One can observe how accurate every model is in classifying each class separately (Recall: "DDoS", Selectivity: "Benign"). Also, the F1 score shows the second indicator of accuracy. Note that the main interest of the detection system is to have a classification model very precisely classifying DDoS (low FN) and, in turn, presenting the least possible number of false alarms (low FP). According to the metrics, a high Recall value represents a model accurately detecting DDoS, and a high Selectivity value represents a model with a low number of false alarms (see Eq. 7 and 8). According to that, the M1 model is the most accurate in detecting DDoS attacks, with a Recall value of 98%. However, M1 has the lowest selectivity value, which makes this model one of the least reliable, since the false alarm rate would be higher than the other models. This analysis is reflected in the low value of the F1_Score.

Aiming at a better trade-off among accuracy, false alarms, and time window duration, we consider that M3 and M7, both in the 40-second window, are the best models to use by our detection system. These models have the closest Recall and Selectivity values, which means a more balanced choice concerning the mentioned optimization aspects.

TABLE V
COMPARISON OF METRIC RESULTS OBTAINED IN THE TESTING PHASE

Random Forest (RF)			
T.W. / Metric	**Recall**	**Selectivity**	**F1 Score**
M1: 5s	0.982 ± 0.001	0.920 ± 0.003	0.934 ± 0.002
M2: 20s	0.933 ± 0.005	0.939 ± 0.003	0.933 ± 0.003
M3: 40s	**0.957 ± 0.002**	**0.957 ± 0.003**	**0.952 ± 0.003**
M4: 60s	0.963 ± 0.003	0.941 ± 0.002	0.959 ± 0.002
K-Nearest Neighbor (KNN)			
T.W. / Metric	**Recall**	**Selectivity**	**F1 Score**
M5: 5s	0.967 ± 0.002	0.935 ± 0.003	0.937 ± 0.003
M6: 20s	0.951 ± 0.003	0.943 ± 0.004	0.945 ± 0.003
M7: 40s	**0.970 ± 0.003**	**0.967 ± 0.003**	**0.962 ± 0.002**
M8: 60s	0.970 ± 0.003	0.946 ± 0.003	0.964 ± 0.002

VI. CONCLUSION AND FUTURE WORK

This paper presented ORACLE, an architecture implemented over an SDN/P4 environment that promotes the cooperation of the control and data planes to detect network attacks using ML models. ORACLE leverages data plane programmability to extract flow information at a higher level of granularity trying to improve the calculation and classification of the feature set in the attack detection process. As proof of concept, we implemented a DDoS detection system based on the computation of intra-flow level features, which cannot be implemented in a traditional SDN/OpenFlow environment. Obtained results show an accuracy rate of up to 96% while reducing the processing complexity performed by the controller. As future work, we plan to extend ORACLE to detect different types of network attacks.

ACKNOWLEDGMENT

This paper has been supported by the Ibero-American Science and Technology Program CYTED (Project: 519RT0580).

REFERENCES

[1] S. Weisman. (2019) What is a distributed denial of service attack (ddos) and what can you do about them? norton. [Online]. Available: https://us.norton.com/internetsecurity-emerging-threats-what-is-a-ddos-attack-30sectech-by-norton.html

[2] J. C. C. Chica et al., "Security in sdn: A comprehensive survey," *Journal of Network and Computer Applications*, p. 102595, 2020.

[3] N. Sultana et al., "Survey on sdn based network intrusion detection system using machine learning approaches," *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 493–501, Mar 2019.

[4] P. Bosshart et al., "P4: Programming protocol-independent packet processors," *SIGCOMM CCR*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[5] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE communications surveys & tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[6] G. A. Ajaeiya et al., "Flow-based intrusion detection system for sdn," in *IEEE ISCC 2017*. IEEE, 2017, pp. 787–793.

[7] A. S. da Silva et al., "Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn," in *IEEE NOMS 2016*. IEEE, 2016, pp. 27–35.

[8] C. Yoon et al., "Enabling security functions with sdn: A feasibility study," *Computer Networks*, vol. 85, pp. 19–35, 2015.

[9] Â. C. Lapolli, J. A. Marques, and L. P. Gaspary, "Offloading real-time ddos attack detection to programmable data planes," in *IEEE IM 2019*. IEEE, 2019, pp. 19–27.

[10] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." in *ICISSP*, 2018, pp. 108–116.