Received: 24 July 2018 | Revised: 23 April 2019 | Accepted: 23 April 2019
DOI: 10.1002/cpe.5344

WILEY

RESEARCH ARTICLE

Toward a monitoring and threat detection system based on stream processing as a virtual network function for big data

Martin Andreoni Lopez[1,3] | Diogo M. F. Mattos[1,2] | Otto Carlos M. B. Duarte[1] | Guy Pujolle[3]

[1]Universidade Federal do Rio de Janeiro - GTA/COPPE/UFRJ, Rio de Janeiro, Brazil
[2]Universidade Federal Fluminense - TET/PPGEET/UFF, Niterói, Brazil
[3]Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, Paris, France

Correspondence
Martin Andreoni Lopez, Universidade Federal do Rio de Janeiro, GTA/COPPE/UFRJ, 21945-970 Rio de Janeiro-RJ, Brazil.
Email: martin@gta.ufrj.br

Funding information
CNPq; CAPES; FAPERJ; FAPESP

Summary

The late detection of security threats causes a significant increase in the risk of irreparable damages and restricts any defense attempt. In this paper, we propose a s**CA**lable **TR**Affic **C**lassifier and **A**nalyzer (CATRACA). CATRACA works as an efficient online Intrusion Detection and Prevention System implemented as a Virtualized Network Function. CATRACA is based on Apache Spark, a Big Data Streaming processing system, and it is deployed over the Open Platform for Network Functions Virtualization (OPNFV), providing an accurate real-time threat-detection service. The system presents a friendly graphical interface that provides real-time visualization of the traffic and the attacks that occur in the network. Our prototype can differentiate normal traffic from denial of service (DoS) attacks and vulnerability probes over 95% accuracy under three different datasets. Moreover, CATRACA handles streaming data under concept drift detection with more than 85% of accuracy.

KEYWORDS

big data, network traffic classification, stream processing, threat detection, virtual network function

1 | INTRODUCTION

The Internet is facing constant changes, from the diversity of the user, the complexity of its application, until the heterogeneity of the information producers.[1] As a consequence, traffic monitoring, a critical task in maintaining the stability, reliability, and security of computer networks, is facing new challenges.[2] Current network monitoring tools are inadequate for current speed and management needs of large network domains. To ensure network security, new systems must be designed since current security systems such as Security Information and Event Management (SIEM) are inadequate. While 82% of security threats occur in minutes, an intrusion can take up to 8 months to be detected.[3] It is essential that the detection time is the least possible so that intrusion prevention can be effective.[4]

Security incidents have increased their complexity, and simple analysis and filtering of packets are no longer sufficient. Attackers try to hide malicious traffic from the security tools by forging the source IP and dynamically changing TCP port. In this context, a promising alternative for classifying network traffic and detect threats is to apply Machine Learning (ML) techniques. These techniques are suitable for big data, with more samples to train the classifier, as methods have higher effectiveness.[5] With a large number of features, however, ML techniques perform results with high latency due to computational resource consumption. This high latency is a disadvantage for applications that use machine learning for real-time classification. For example, network monitoring applications must analyze data and detect threats as quickly as possible. In this context, real-time stream processing allows the immediate analysis of different types of data and consequently benefits traffic monitoring for security threat detection. Open source distributed processing platforms such as Apache Storm,[6] Apache Flink,[7] and Apache Spark[8] process big data with low latency.

In a previous work,[9] we evaluated the performance of our monitoring system as a virtual network function. Results show that our virtual network function can scale and migrate during traffic overload. Moreover, network traffic is processed in parallel to analyze big datasets. These results are our motivation to propose a CATRACA (A sCAlable TRAffic Classifier and Analyzer). The CATRACA system uses Network Function Virtualization (NFV) technology over a Network Function Virtualization Infrastructure (NFVI) to combine virtualization, cloud computing, and distributed stream processing to monitor network traffic and detect threats. The goal is to provide an accurate, scalable, and real-time threat

detection system to meet the peak of use, providing a high Quality of Service. Traffic monitoring and threat detection as a virtualized network function have two main advantages: the ability of self-adapting to different traffic volumes and the flexibility of installation and migration of sensors in the network to reduce the latency in monitoring.[10] Self-adaptation is reached with an elastic behavior, matching different traffic and processing rates. The system creates and destroys virtual machines when necessary. The system presents installation flexibility because it runs on the top of virtual machines that are hardware agnostic. Sensors are migrated using virtualization features. Thus, the system analyzes large volumes of data, while Machine Learning techniques classify the traffic into normal or threat, and finally, the knowledge extracted from the flows is presented in a user interface.*

The remainder of the paper is organized as follows. Section 2 discusses the related work. In Section 3, we describe CATRACA architecture. Offline CATRACA implementation is presented in Section 4.1 for a big dataset. In Section 4.2, we perform a traffic classification use case. In Section 4.3, we demonstrate a use case of CATRACA analyzing streaming data. Finally, Section 5 concludes the work.

2 | RELATED WORK

Many proposals using Apache Storm stream processing platform to perform real-time anomaly detection are found in the literature. Du et al use Apache Flume and Apache Storm to monitor network traffic to detect anomalies. They use the k-nearest neighbours (k-NN) algorithm to classify network traffic.[11] Performance evaluation is presented; however, the paper lacks results regarding classification metrics such as accuracy, precision, and recall, among others. Moreover, the prototype only receives data from a single source, ignoring data from distributed sources. Similar to previous work, the work of Zhao et al uses Apache Kafka and Apache Storm for the detection of network anomalies,[12] characterizing flows in the NetFlow format. He et al propose a combination of the distributed processing platforms Hadoop and Storm, in real time, for anomaly detection. In this proposal, a variant of the k-NN algorithm is used as the anomaly detection algorithm.[13] The results show a good performance in real time, however, without using any reaction and threats prevention. Villar-Rodriguez et al propose the use of Support Vector Machine (SVM) to detect identity theft in social networks.[14] The authors monitor user profiles based on connection time information. SVM classifies the legitimate user and the attackers' profiles. Li et al propose an approach to anomaly detection in traffic monitoring. The authors use Principal Component Analysis (PCA) over the Random Forest machine learning algorithm to identify the most important features.[15] Results show good performance under traditional KDD'99 dataset[16] and from authors' campus dataset. Nevertheless, none of these proposals work with streaming data.

The Open Security Operations Center (OpenSOC)[17] is a collaborative development project that integrates open source software aimed at an extensible and scalable security analysis tool. Thus, OpenSOC is an analytical security framework for monitoring big data using distributed stream processing. OpenSOC was discontinued and gave rise to the Apache Metron project,[18] proposing a new architecture that aims to facilitate the addition of new data sources and better exploit the parallelism of the Storm tool. Metron architecture comprises acquisition, consumption, distributed processing, enrichment, storage, and visualization of the data layers. The key idea of this framework is to allow the correlation of security events from different sources such as application logs and network packets. To this end, the framework employs distributed data sources such as sensors in the network, event logs of active network security elements, and enriched data called telemetry sources. The framework also relies on a historical foundation of network threats from Cisco. Apache Spot† is a project similar to Apache Metron, and it is still in incubation. Apache Spot uses telemetry and machine learning techniques for packet analysis to detect threats. The creators mention that the big difference with Apache Metron is the ability to use standard open data models for networking. Stream4Flow‡ uses Apache Spark with the ElasticStack stack for network monitoring. The prototype serves as a visualization of network parameters. Stream4Flow,[19] however, lacks the intelligence to perform anomaly detection. Hogzilla§ is an intrusion detection system (IDS) with support for Snort, SFlows, GrayLog, Apache Spark, HBase, and libnDPI, which provides network anomaly detection. Hogzilla also allows realizing the visualization of the traffic of the network.

CATRACA, like Metron, was also inspired by OpenSOC and aims to monitor large volumes of data using stream processing. The CATRACA system is implemented as a Virtual Network Function (VNF) over the Open Platform for Network Function Virtualization (OPNFV) environment. CATRACA focuses on real-time packet capture, data preprocessing, machine learning, and a countermeasure mechanism for immediate blocking of malicious flows. Thus, the CATRACA system acts as a Virtualized Network threat detection and prevention function that reports flow summaries. In addition, CATRACA as a VNF can be linked to other network virtualized functions[20] as defined in the network function chain patterns, Service Function Chaining (SFC) and Network Service Header (NSH).

3 | THE CATRACA SYSTEM

Current enterprise networks rely on middleboxes. Middleboxes are intermediary devices that add new functionalities to the network such as intrusion detection system, firewall, and proxy. Middleboxes are dedicated hardware nodes, which perform a specific network function. Hence,

* The system, as well as its documentation and complementary information, can be accessed at http://gta.ufrj.br/catraca.
† http://spot.incubator.apache.org. Accessed April 2018.
‡ https://github.com/CSIRT-MU/Stream4Flow. Accessed April 2018.
§ http://ids-hogzilla.org/. Accessed April 2018.

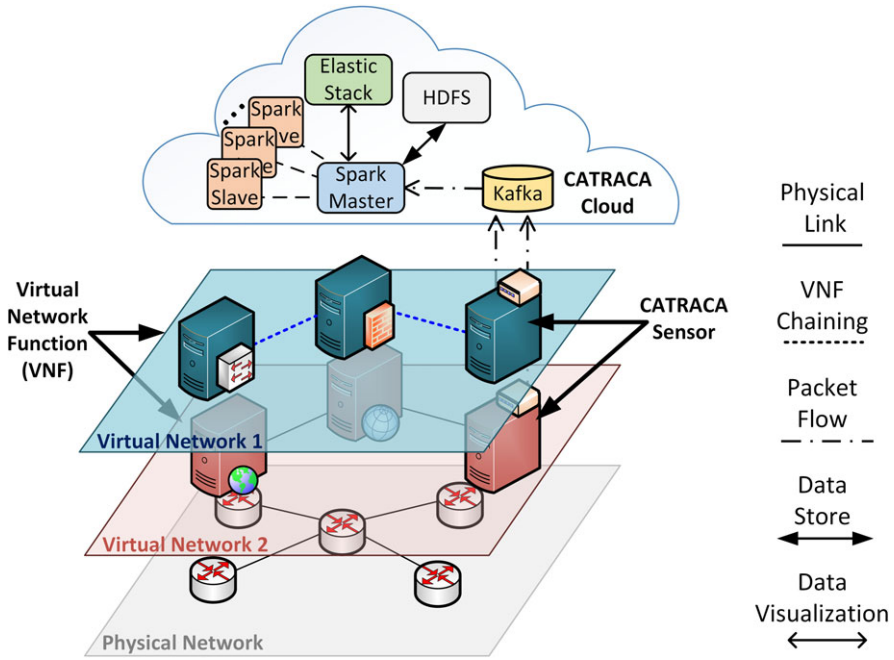

FIGURE 1 CATRACA as a Virtual Network Function. CATRACA sensors mirror traffic to Apache Kafka. Network packets are summarized as flows, which Apache Spark analyzes. Machine Learning models are obtained and stored in the Hadoop Distributed File System (HDFS) and results are displayed in the ElasticStack

middlebox platforms come with high Capital Expenditures (CAPEX) and Operational Expenditures (OPEX). In this way, the Network Function Virtualization (NFV) comes to leverage standard virtualization technology into the network core and to merge network equipment into commodity server hardware.[21] In NFV, the network functions are deployed into a virtualized environment and is thus called Virtual Network Functions (VNF). Virtual network functions (VNF) are virtual machines performing functions on the network layer to replace the numerous hardware-specific middleboxes. Moreover, Service Function Chaining (SFC) allows an infrastructure provider to select the appropriate VNF from multivendor implementations. An Intrusion Detection System (IDS) and a Firewall is a good example of a service function chaining.[20]

CATRACA aims to use NFV technology and its cluster infrastructure to combine virtualization, cloud computing, and distributed stream processing to monitor network traffic. The objective is to provide an accurate, scalable, and real-time threat detection facility capable of attending usage peaks. The traffic monitoring and threat detection as a virtualized network function present two main advantages: capacity self-adaptation to different traffic network load and high localization flexibility to place or move network sensors reducing latency.

CATRACA is deployed as a Virtual Network Function (VNF) as shown in Figure 1. CATRACA sensors are deployed in virtual networks. The goal of the sensors is to mirror traffic to CATRACA cloud. CATRACA cloud is composed of Apache Kafka that receives the mirrored traffic and sends it to Apache Spark, responsible for data processing. Apache Spark creates a machine learning model that is stored in the Hadoop Distributed File System (HDFS), and finally, results are displayed in the ElasticStack that contains the Elastic Search and Kibana for data visualization.

The CATRACA architecture is composed of three layers: Visualization Layer, Processing Layer, and Capture Layer, as shown in Figure 2. The first layer, the Capture Layer, captures network packets through traffic mirroring by the libpcap library. A Python application based on *flowtbag* abstracts the packets into flows. Many open-source software resume packets into flow features such as tcptrace,[¶] flowtbag,[#] Traffic Identification Engine (TIE),[‖] flowcalc,[**] Audit Record Generation and Utilization System (ARGUS),[††] among others. We apply *flowtbag* because it abstracts more packet features than others. *Flowtbag* get 45 features, (5) flow tuple information (IP/ports/protocol), (4) packets/bytes in forward/backward direction, (8) packets statistics forward/backward direction, (8) time between packets forward/backward direction, (8) flow time statistics, (4) subflow packets/bytes forward/backward direction, (4) TCP flags, (2) Bytes used in headers, (1) type of service, and (1) Quality of Service.

We define a flow in CATRACA as a sequence of packets with the same 5−tuple source IP, destination IP, source port, destination port, and protocol during a time window. In all, 45 flow features and one feature for the label are extracted and published in a publisher/subscriber service of Apache Kafka. The service operates as a low latency queue and data flow manipulation system, where the Processing Layer consumes queue features.

[¶] Tcptrace http://www.tcptrace.org. Accessed April 2018.
[#] flowtbag: https://github.com/DanielArndt/flowtbag. Accessed April 2018.
[‖] Traffic Identification Engine http://tie.comics.unina.it/doku.php. Accessed April 2018.
[**] flowcalc http://mutrics.iitis.pl/flowcalc. Accessed April 2018.
[††] ARGUS http://www.qosient.com/argus. Accessed April 2018.

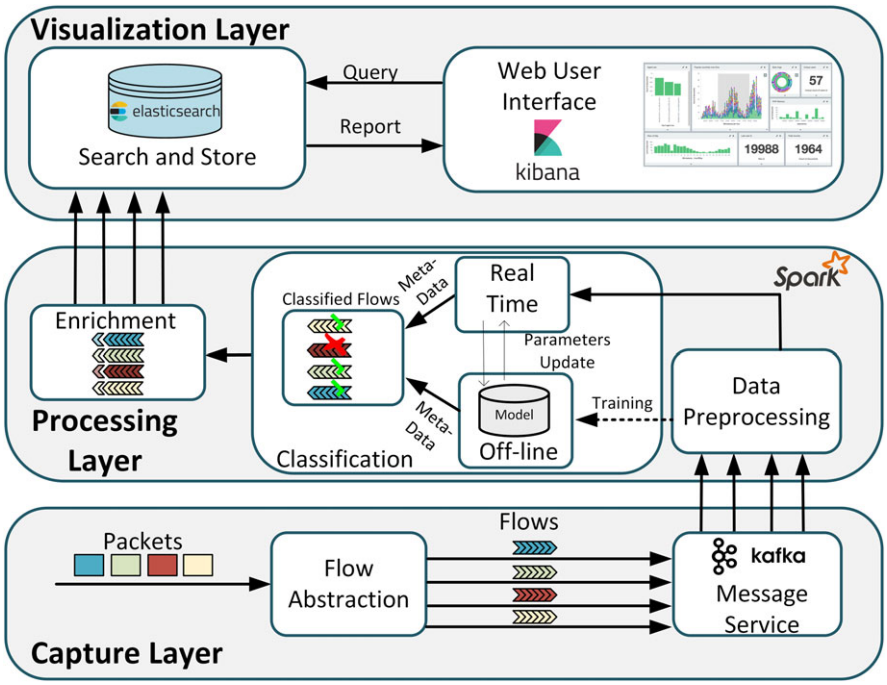

FIGURE 2 The layered architecture of the CATRACA system: the capture layer, the processing layer, and the visualization layer. The capture layer abstracts network packets into stream flows. All operations such data preprocessing, machine learning, and data enrichment are implemented in the processing layer. Visualization layer stores data and graphically displays the results

A dedicated cloud for classification hosts the Processing Layer, and its core is the Apache Spark. We choose Spark framework among the different stream-processing platforms because it presents the best fault tolerance performance,[22] making CATRACA more robust in case of failure. Spark runs in a cluster following the master/slave model, where slaves can expand and reduce resources, providing scalability to the system. Once a flow arrives in the Processing Layer, a data preprocessing method normalizes and selects the most critical features for threat classification. Finally, CATRACA classifies flows as malicious or benign through machine learning (ML) algorithms based on decision trees.

CATRACA runs in two modes: offline and online modes. Offline mode analyzes stored big security datasets which are time invariable. Figure 3 shows the flowchart of the offline application of CATRACA. Boxes are the name of the functions, and the arrows show the communication between the functions. First, we upload a file, expressed in comma-separated values (`csv`) representation to the Hadoop Distributed File System (HDFS). The `Read File` function reads the file and transforms it into a Resilient Distributed Dataset (RDD) in the Spark context. The RDD abstracts the files into lines, and each line represents a network flow. The `dataPreparing` function processes the lines, separating each feature by a comma, converting the values to float, and removing the label or classes for each line. Data and the classes are separately processed. The `Feature Correlation` function uses our feature selection algorithm to select the most important feature for each flow. An index represents each feature, from 1 to 45. Once selected, the indexes are stored in the HDFS. Next, the data and the indexes are sent to the `Matrix Reducer` function, where we reduce the original matrix to the one which contains only the previous selected indexes. The `pass2libsvm` transforms the reduced matrix in a `libsvm` format. LibSVM is a library for implementing support vector machine classification. CATRACA, however, only uses the data format of this library as input for machine learning algorithms in the Spark context. The format of the libsvm library is `<label> <index1>:<value1> <index2>:<value2>`, where *label* is the class of the flow, *index* are the features, and *value* are the numerical values of features. Once the data are ready, the `Divide Data` function divides them into training and testing sets in a rate of 70% for training and 30% for testing. The training set feeds the `Create Model` function that creates the machine learning model. In CATRACA, we use decision tree as the machine learning mechanism. After creating the model, we store it in the HDFS for further use. Finally, the `Classify` function obtain the model and evaluate it against the testing set. This function also compares the predicted values with the original dataset classes, and the metrics such as accuracy, precision, and F1-score are obtained. The metrics are finally stored also in the Hadoop Distributed File System.

CATRACA online mode is presented in Figure 4. This mode works similarly than the offline mode. In contrast to the offline mode that analyzes static data, online mode processes streaming dynamic data. As a result, the streaming data are unlabeled data as they arrive with no class annotation since they are created in real-time. First, the `getStream` function get the streaming of flow messages that came from Apache Kafka. The function defines the parameters of the Apache Kafka receiver inside the Apache Spark. Then, the `convert2JSON` function process the streaming data and parses them to the JavaScript Object Notation (JSON), which is easier to handle. `ExtractIPs` function get the IPs source and destination address from each flow; these IPs are passed to the `addLocation` where the geographical coordinates of each IP are added. On the other side, the features without the IP addresses are inputs for to the `convert2float` function. This function transforms all data into float values. Next, the `Matrix Reducer` function is inherited from the offline mode. This function takes the stored indexes from the HDFS, which

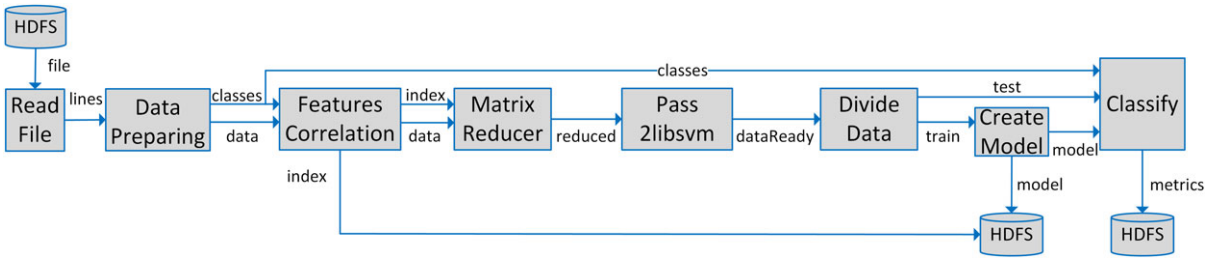

FIGURE 3 Flowchart of CATRACA running in offline mode. The system loads a network flow dataset from a repository stored on HDFS. Spark system is the processing core of CATRACA. Feature selection and data classification are applications running on the top of Spark

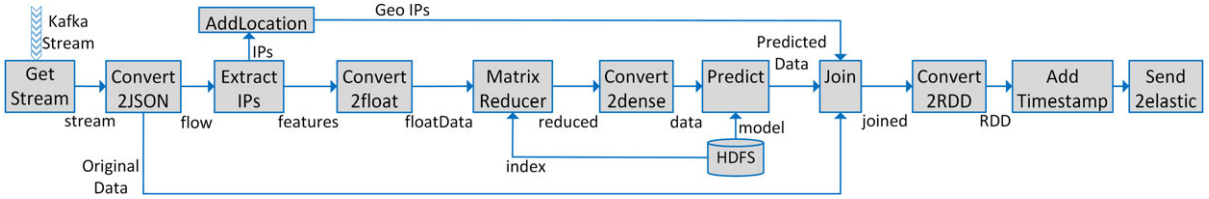

FIGURE 4 Flowchart of CATRACA running in online mode. Flows arrive from the network as message in Kafka. The online application consumes flow messages from Kafka and enriches the flow features. Online mode introduces data transformations to adapt streaming data to Spark machine learning libraries

were calculated in the offline mode and applies a reduction on the data. Then, we convert the reduced feature matrix into a new matrix with the selected indices as columns. This step is similar to pass2libsvm in the offline mode. However, the libsvm format is unsuitable to a streaming data environment. The `predict` function load the machine learning model obtained during the offline model and classify the flows in 0 as normal flow or 1 as an alert. The `join` function combines the original data with both the result of the classification and the geographical location. All data are merged into a single vector and forms a new RDD. A timestamp, the flow processing time label, enriches the data. Finally, the `send2elastic` function sets the parameters and adapts the data to send to ElasticSearch.

Our system enriches original metadata with information to enhance the quality of the detection. The idea behind enrichment is to add information in the streaming data about the environment of the attack. We first extract the IP address, and we add the geographical location of the analyzed IPs. With this information, our system can detect attacks in different geographical location and have a global view of the attack. As a consequence, a Distributed Denial of Service (DDoS), for example, is easier to detect. Other sorts of meta-data can enrich the streaming data such as the timestamp of the processed flow, which would let us handle the data as a time series. Moreover, information about the user, such as if it is a residential or a corporate user, increases the knowledge about an attack.

Finally, the Visualization Layer deploys ElasticStack and Kibana. The ElasticStack allows real-time custom event viewing. Thus, the output of the Processing Layer follows to the ElasticSearch,[‡‡] which provides a fast search and storage service. A user interface runs in the Kibana environment, which communicates with the stored data in ElasticSearch through queries. The combination allows exposing the results in real-time.

4 | CATRACA EVALUATION

CATRACA runs on the top of the Open Platform for Network Function Virtualization (OPNFV). We perform the experiments in the OPNFV Danube 2.0 environment. Our OPNFV environment runs on five bare-metal servers, two controllers, two compute nodes with 96 GB of RAM, 700 TB of storage, and 128 cores of Intel Xeon processors with a clock frequency of 2.6 GHz. Besides, fuel manager runs on a Quad-core, 4-GB RAM, 128-GB SDD storage with three network interfaces of 1/GBs.

We use machine learning algorithms against three different datasets to evaluate CATRACA. The NSL-KDD dataset is a modification of the original KDD-99 dataset and presents the same 41 features and the same five classes, Denial of Service (DoS), Probe, Root2Local (R2L), User2Root (U2R) and normal, as the KDD 99.[23] The NSL-KDD dataset was improved when compared with original KDD 99. Improvements of the NSL-KDD over KDD 99 are the elimination of redundant and duplicate samples, to avoid a biased classification and overfitting, and a better cross-class balancing to avoid random selection. The second dataset is the GTA/UFRJ[§§] that combines real network traffic captured from a laboratory and network threats produced in a controlled environment.[24] Network traffic is abstracted in 26 features and contains three classes, DoS, probe, and normal traffic. The third dataset is the NetOp,[¶¶] a real dataset from a Brazilian operator.[25] The dataset contains anonymized

[‡‡] ElasticSearch and Kibana are open-source code and belong to the ElasticStack. https://www.elastic.co/products. Accessed April 2018.
[§§] Anonymized data are available by emailing contact to the authors.
[¶¶] Anonymized data are available by emailing contact to the authors.

TABLE 1 Summary of datasets used for evaluation

Dataset	Format	Size	Attacks	Classes	Type
NSL-KDD	41 Features	150k Flows	80.5%	5	Synthetic
GTA/UFRJ	26 Features	95 GB	30%	3	Synthetic
NetOp	46 Features	5 M Flows	-	2	Real

access traffic of 373 broadband users of the South Zone of the city of Rio de Janeiro. We collected data during one week of uninterrupted data collection, from February 24 to March 4, 2017. We summarized packets in a dataset of 46 flow features, associated with an IDS alarm class or the legitimate traffic class. We summarize all datasets in Table 1.

In this work, we present three use cases of CATRACA. The first two present CATRACA when it runs in offline mode, and the last one shows the streaming behavior of CATRACA. As a consequence, we first analyze real network traffic dataset. After, we compare the performance of three machine learning algorithms in three different security network datasets. Finally, we evaluate CATRACA when handling streaming data.

4.1 | Offline dataset evaluation use case

We use CATRACA to analyze and characterize the NetOp dataset. The NetOp dataset is a big dataset obtained from real network traffic. With big dataset, a computer cluster must be used to introduce low latency when processing the data. A previous work[25] analyzed just one day of the network traffic. In this paper, we analyze the entire dataset, in total of one full week. We loaded the entire dataset into the Hadoop File Distributed System (HDFS). In this use case, CATRACA analyzes the main features chosen by the Principal Component Analysis (PCA)[26] method. The PCA method selects the features that are the most representative components of eigenvectors with the most significant eigenvalues. Then, CATRACA extracts statistics about network flows using the core of Apache Spark. We analyze the relation between normal traffic and alerts. Figure 5 shows the number of threats and normal flow in each day of the dataset in 2017. We can see that almost all days contain around 30% of alerts. Only one day, February 17 contains a smaller number of alerts. The maximum number of alerts occurs on Saturday, February 25, reaching 1.8 Million alerts.

Figure 6 shows the source and destination ports of the flows. The figure focuses on the 1024 first ports (from 0 to 1023), as they are the operating system-restricted ports. Usually, daemons that execute services with system administrator privileges use these ports. Our flow definition assumes that the source port initiates the TCP connection. As the dataset portrays home users, most connections are destined to restrict and dynamic ports. Thus, we remark that the number of alerts coming from connections that the destination port is in the range of restricted ports is low to the total number of connections on these ports (Figure 6B). When considering the flows, in which the source port is in the range of restricted ports, almost all flows are classified as alerts by CATRACA, as shown in Figure 6A. Another important fact is that most of the analyzed flows reflect the use of the DNS service (UDP 53) and HTTPS and HTTP services (TCP 443 and 80). The prevalence of HTTPS services over HTTP reflects the shift that major Internet content providers such as Google and Facebook have done to use encrypted service by default to ensure users' privacy and security.

The relation between the most accessed services and flow duration are shown in Figure 7A and 7B. The duration of the analyzed flows is less than 40 ms, characterizing the use of DNS, HTTP, and HTTPS services. Regarding the protocols used, the prevalence of UDP flows is evident and refers to DNS queries. It is worth mentioning that the number of alerts generated by UDP flows is more than 10 times greater than the number of alerts generated by TCP flows. Another important point is that the number of flows that generate alerts is approximately 26% of total flows.

Figure 8 shows the characterization of the number of packets per flow in uplink and bytes per flow in the downlink direction. In uplink direction, Figure 8A, 80% of alerts starts with 20 packets or less while normal traffic starts with almost 80 packets. This behavior is typical from

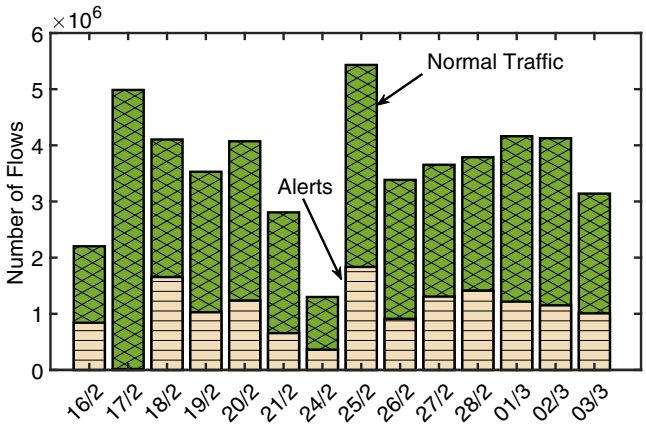

FIGURE 5 Number of alerts and normal traffic flows in network operator dataset

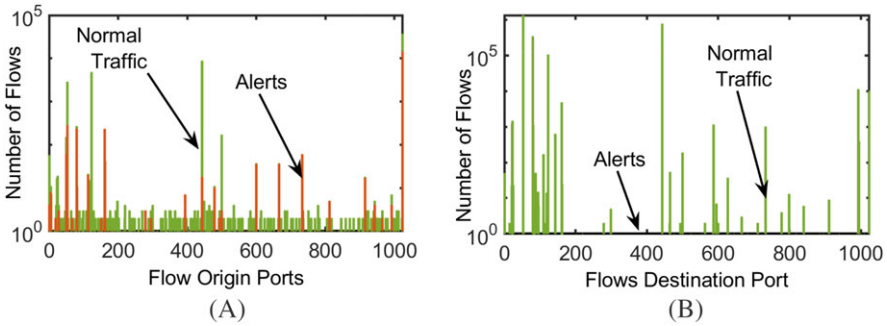

FIGURE 6 Source and destination ports in flows. Comparison of the use of the lowest 1024 ports (restricted ports) in the evaluated flows. As users are fundamentally residential, the largest number of flows originating from these ports are flows that generate alerts. A, Source ports distribution; B, Destination ports distribution

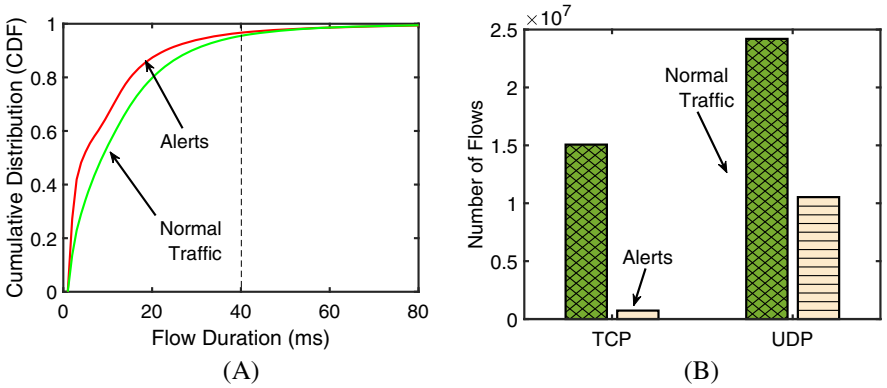

FIGURE 7 Cumulative Distribution Function (CDF) for the duration of flows in milliseconds and number of flows per transport protocols. (A) The flows that generate alerts are shorter in duration than the average flow. (B) The legitimate flows with UDP are numerous due to DNS (port 53 UDP). The number of alerts in UDP is more than 10 times greater than in TCP flows. A, Flow duration in NetOp dataset; B, Transport protocols used in NetOp dataset

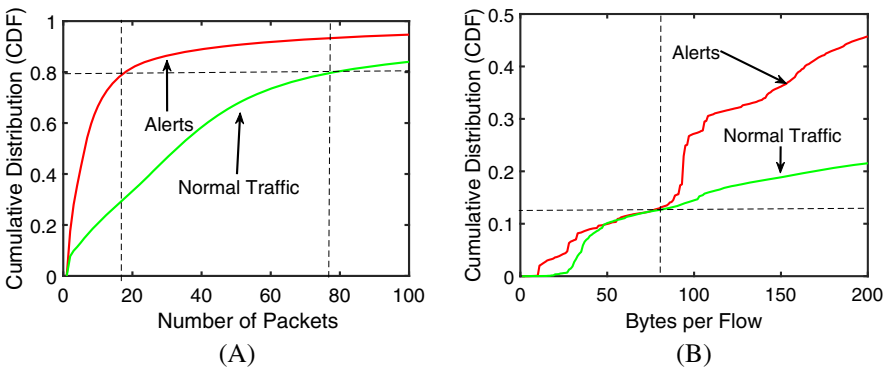

FIGURE 8 Cumulative Distribution Function (CDF) for the number of packets per flow. Flows that generate alerts tend to have fewer packets. A, Flows in the uplink direction; B, Flows in the downlink direction

probe or scans attacks that send small amounts of packets to discover target vulnerabilities. In Figure 8B, alerts and normal traffic show a similar pattern of 11% of flows. However, alerts use more than 100 Bytes in more than 30% of flows.

Considering the amount of data transferred in each flow, Figure 9 compares the round-trip flows concerning volume in bytes. The disparity of the traffic volume in both directions of communication is evident. While, in one way, 95% of traffic presents a maximum volume of 100 B; in the other way, the same traffic share presents up more than 500 B. This result demonstrates that the residential broadband user profile is a content consumer. Another point is that the flows that generate alerts have a similar traffic volume profile in both directions. Asymmetric traffic is more typical of legitimate users.

Figure 10 shows the behavior of the subflows generated in each connection. A new subflow is created whenever a flow reaches the idle state. Both Figure 10A and 10B, subflow size in uplink and downlink, show a very similar behavior. Over 20% of normal traffic flows reach 900 B, but

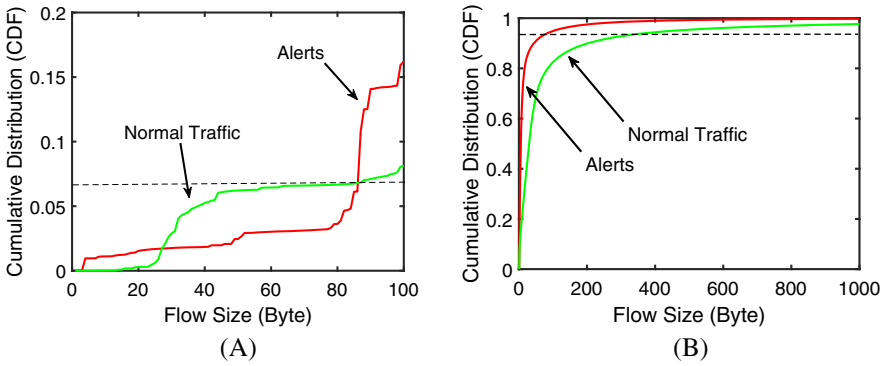

FIGURE 9 Cumulative Distribution Function (CDF) for volume in bytes by flow. Flows that generated alerts tend to have smaller volumes in transferred bytes. A, Flows in the uplink direction; B, Flows in the downlink direction

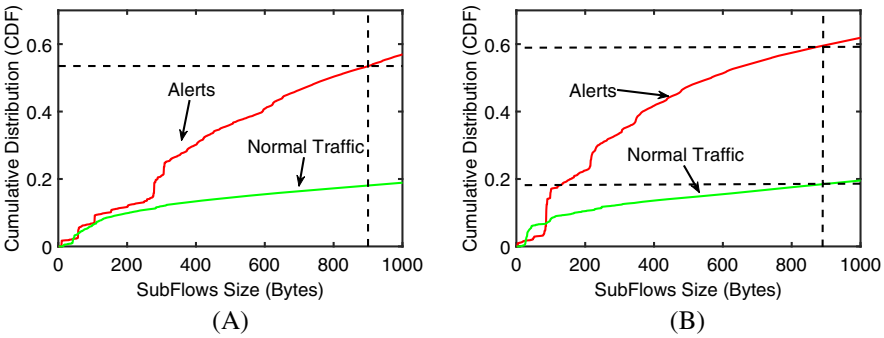

FIGURE 10 Cumulative Distribution Function (CDF) for volume in bytes by subflow in each flow. Flows that generate alerts tend to have smaller volumes in bytes transmitted in subflows. A, Subflows in the uplink direction; B, Subflows in the downlink direction

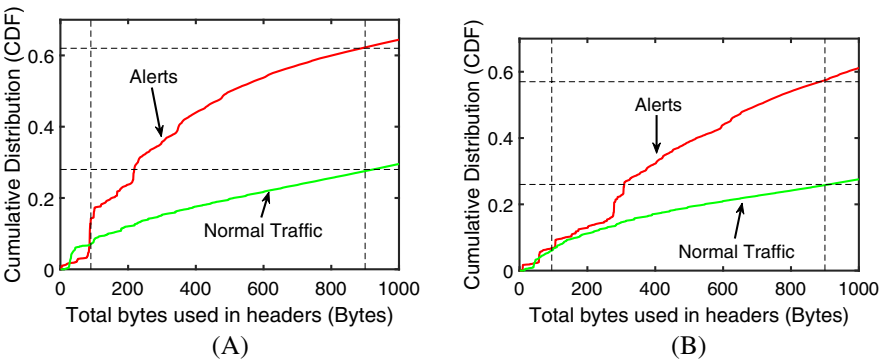

FIGURE 11 Cumulative Distribution Function (CDF) for volume of packet header data. The behavior of traffic that generates alerts is very similar to total traffic. A, Flows in the uplink direction; B, Flows in the downlink direction

this value is reached in almost 60% of the flows. Values of subflows are almost ten times bigger than the values represented in Figure 9, where flow size in Bytes are shown. Flows are short duration, as evidenced in Figure 7A, do not pass to the idle state, and thus, do not generate subflows.

Another important feature is the total amount of data in packet headers. Figure 11 shows that, in both directions, alert and normal flows have the same behavior. In particular, there is symmetry in the round-trip traffic regarding the volume of data in the headers. It highlights that malicious traffic does not rely on the usage of header options. Moreover, in both directions, uplink and downlink show similar behavior. Until 90 Bytes per header alerts and normal traffic are similar, however, with 900 Bytes headers, it represents almost 30% of normal traffic and close to 60% of alerts flows.

Figure 12 shows which are the main classes of alerts triggered by the IDS. Alerts for attacks against HTTP are the most frequent. This class of alerts includes SQL injection attacks through HTTP calls and XSS attacks (cross-site scripting). Home users can execute these attacks as they use the parameters of HTTP calls to insert some malicious code into the servers and, therefore, are not filtered by access rules. Other important alerts are port scanning and execution of malicious applications (trojan and malware). The scans are generally intended to identify open ports and vulnerabilities in user premises such as the home gateway. Alerts for *trojan* and *malware* identify activities typical of known malicious applications

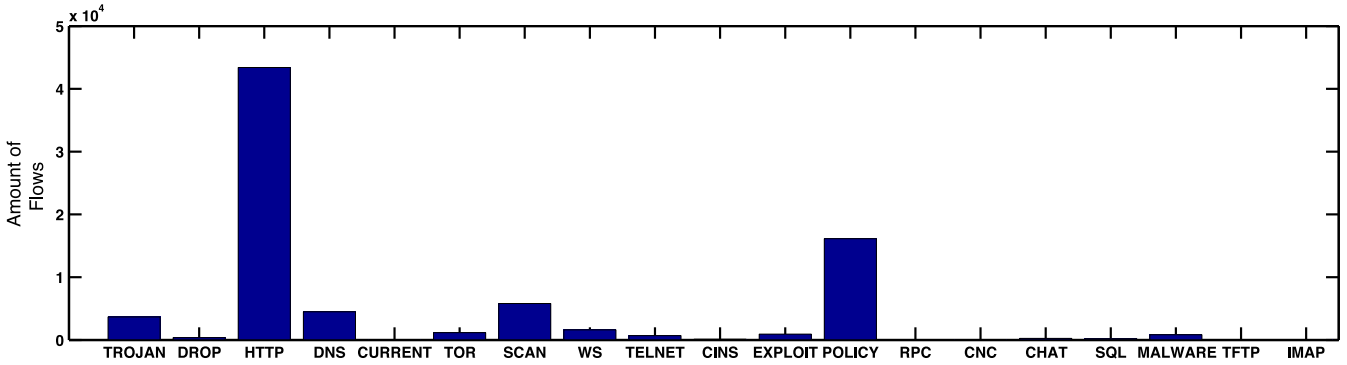

FIGURE 12 Distribution of the main types of alerts in the analyzed traffic

that aim at creating and exploiting vulnerabilities in the devices of the home users. Other alerts refer to information theft and Byzantine-attack signatures on common protocols such as IMAP and Telnet.[##]

4.2 | CATRACA for traffic classification use case

Classification begins with the preprocessing in the selection of the most important features of the flows. Our system can then operate in either real-time or offline mode. Offline traffic classification comprises processing of the micro-batches Spark platform. In this mode, large volume datasets are loaded in a distributed file system such as the Hadoop Distributed File System (HDFS). The data set is separated into a set of training and a test set at a ratio of 70 % to training and 30 % to the test. Thus, Spark performs processing by the technique map-reduce. Applications in Spark run as independent processes in the cluster which are coordinated by the master node. The master node receives the application split it into tasks. Moreover, the manager node is responsible for scheduling tasks in the worker nodes. Once the worker finished the assigned task, the results are reported to the manager node. We train a decision tree algorithm to obtain the classification model, and we store it in the Hadoop distributed file system (HDFS). The manager will be responsible for loading the newly published model and deploying to the worker nodes. Results were obtained with 10-fold cross-validation, in four virtual machines (VM), one Spark master and three Spark slaves, using Ubuntu 16.04 with 4 GB of RAM and two cores for each VM.

We compare the performance of the machine learning algorithms for traffic classification use case. For this experiment, we test eight machine learning algorithms for classification: *K*-Nearest Neighbors (K-NN), Artificial Neural Network Multilayer Perceptron (Neural Net. MLP), Random Forest, Support Vector Machine (SVM) with Radial Basis Function (RBF) Kernel and with linear kernel, Gaussian Naive Bayes, Stochastic Gradient Descendent, and Decision Tree. For the training set, we choose the day 25/2 of NetOp dataset since it is the day that contains the most significant amount of flows. Moreover, we validate the performance under 10-fold cross-validation. Results are shown in Figure 13. Figure 13A shows the performance of four evaluation metrics Accuracy, Precision, Sensitivity, and F1-Score, while Figure 13B presents a comparison between Training and Classification Times, in which time is represented in a logarithmic scales. As we can see, Decision Tree presents the best performance under the four evaluated metrics. Even if Decision Tree does not present the smallest training time, the performance presented during the classification time is one of the best.

The decision tree classification algorithm runs in the core of CATRACA for a good trade-off between its training/classification times allied to its high accuracy and precision. The decision tree is a greedy algorithm that performs a recursive binary partitioning of the resource space. Each sheet, in CATRACA, a feature, or a combination of them is chosen by selecting the best separation from a set of possible divisions, to maximize the gain of information in a tree node. The division into each node of the tree is chosen from the $\text{argmax}_d GI(CD, d)$, where *argmax* is the point where function gets its maximum value and $GI(CD, d)$ is the information gain when a division d applies to a set of CD data. The idea of the algorithm is to find the best division between features to classify threats. For that, we use the heuristic of Information Gain. The gain of information GI of the system CATRACA is the impurity of Gini, $\sum_{i=1}^{C} f_i(1 - f_i)$, which indicates how separated the classes are, where f_i is the frequency of class i in a node and C is the number of classes. The model is stored in the file system and loaded in to be used in real-time traffic classification mode online. Thus, it is also possible to validate the model with the 30% test set obtained earlier.

Tables 2, 3, and 4 show the confusion matrix of the three evaluated datasets. We consider a network flow sampling as a sliding window of 2 s duration since Lobato et al suggest that it is the best trade-off between classification accuracy and decision latency.[24] The confusion matrix specifies the rate of false positives and other metrics of each class in the test data set. The rows represent the elements that belong to the real class, and the columns represent the elements that were classified as belonging to the class. Therefore, the prominent diagonal elements of this array represent the number of elements that are correctly classified. Moreover, the tables present metrics complementary to the confusion matrix. By observing the values of Accuracy and Precision, it is possible to see the good performance of the decision tree algorithm in off-line

[##] Mainly used for remote configuration of network equipment.

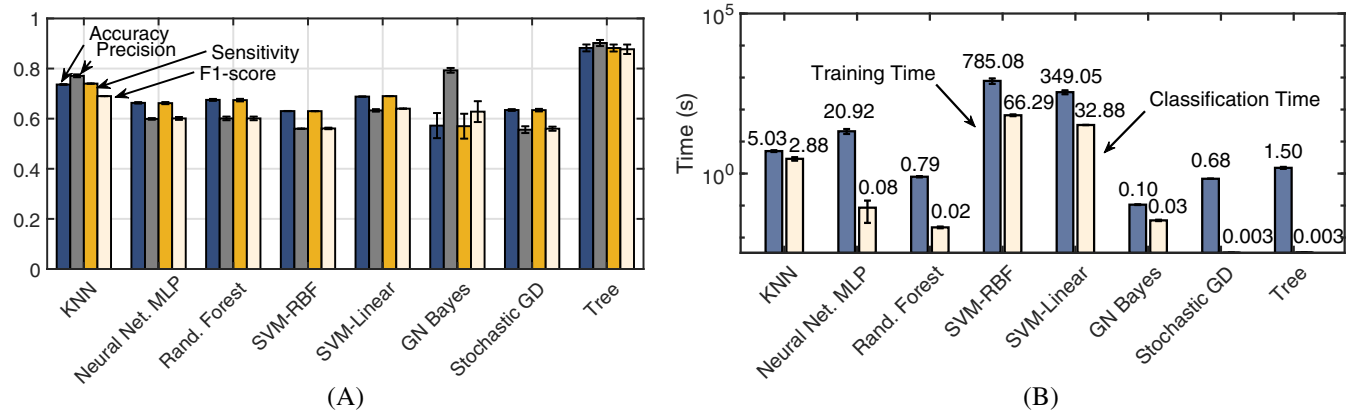

FIGURE 13 Performance comparison for eight machine learning algorithms running on the top of CATRACA. In (A), we evaluate Accuracy, Precision, Sensitivity and F1-Score, and in (B), we compare training and classification time. A, Metrics evaluation; B, Training and classification time

TABLE 2 Confusion matrix and evaluation metrics of decision tree for NSL-KDD dataset

	Normal	DoS	Scan	R2L	U2R	Prec.	Acc.
Normal	9556	5	24	125	0	98.414%	98.394
DoS	14	7443	0	1	0	99.799%	
Scan	16	2	2399	5	0	99.05%	
R2L	157	0	0	2727	3	94.458%	
U2R	3	0	0	7	57	85.075%	
Recall	98.05%	99.906%	99.009%	95.183%	95%		

TABLE 3 Confusion matrix and evaluation metrics of decision tree for GTA/UFRJ dataset

	Normal	DoS	Scan	Prec.	Acc.
Normal	29126	1	0	99.97%	95.99%
DoS	60	5845	0	98.94%	
Scan	8	1782	9434	84.05%	
Recall	99.76%	76.62%	100%		

TABLE 4 Confusion matrix and evaluation metrics of decision tree for NetOp dataset

	Normal	Threat	Precision	Accuracy
Normal	3713600	30140	99.19%	98.74%
Threat	22350	416100	94.90%	
Recall	99.40%	93.24%		

classification. Table 2 presents the results for the NSL-KDD dataset. The table verified that the algorithm presented a high accuracy in almost all classes, with a low false positive rate. Another way to see the false positive rate is to observe the values outside the main diagonal. Root to Local (R2L) and User to Root (U2R) classes present the lowest values of recall and precision. These two classes are the most infrequent in the dataset. Thus, the classifier confuses the most imbalanced classes. One solution to this problem is to use dataset balancer such as Synthetic Minority class Oversampling TEchnique (SMOTE)[27] or Adaptive Synthetic (ADASYN) Sampling Approach.[28] Table 3 shows the same evaluation for the UFRJ/GTA dataset. In this dataset, the worst precision was Scan class. As we can see in the main diagonal of the matrix, the machine learning algorithm misclassified mostly all scan class with DoS. We believe the synthetic nature of the attacks causes the behavior above. All DoS attacks were created with random parameters. Thus, parameters such as source IP addresses or ports can be forged. Sommer and Paxson[29] have already identified this effect with an artificial dataset. Finally, a similar result is shown in Table 4, where NetOp dataset was used to evaluate the decision tree. This dataset contains two classes, threat and normal. We can see that the false positives, the values outside the main diagonal, also increase; however, the overall accuracy increases.

After obtaining the classification model from the historical base, one can evaluate the accuracy of the system with data arriving in real time. The operation of the CATRACA system in real time uses the stream module of the Spark platform. Thus, abstracted packets in streams, captured on different virtual machines in the cloud, are processed as they reach the Spark platform. In CATRACA, we consider a flow as a stream. When a stream arrives at the detection system, it is summarized in characteristics using a feature selection algorithm, to reduce processing time. Thus,

the vector of selected features is evaluated in the model obtained in the off-line processing. After extracting the analytic data from the flows, the results are stored in a database for further analysis. The stored data contain the information collected during the detection of threats and can be reprocessed offline to calculate the parameters to be used in the real-time model. To make the system more accurate, when a new threat is detected, offline parameters are updated, obtained feedback between online and offline detection.

We measure the performance of CATRACA concerning processing throughput and processing time per message. We inject the GTA/UFRJ dataset into the system in its totality and replicated as many times as necessary. As the processing layer of the proposed system consumes a message at a time, the method is still being stream processing. The whole dataset is injected in the Kafka message service, but the processing engine consumes micro-batches at a time. Although this setup avoids the evaluation of the total latency from the data generation until the data consumption and the processed result, it assures that the processing engine always has enough incoming data to reach its maximum throughput and to reach the worst-case scenario processing time per sample. The experiment calculates the consumption and processing rate of the entire system. We consider a scale-out strategy to evaluate the speed-up factor that CATRACA reaches. We add new processing cores to the cluster and evaluate the time for processing messages on new architectures in relation to a serial architecture. Figure 14 shows the results of the throughput experiment. In the y-axis, it is shown the throughput as the number of flows processed per second by the system. The throughput reaches the maximum at 20 cores available. This upper-bound is due to the trade-off between communication overhead and the portion of the code that benefits from the parallelism. We also estimate the speed-up in latency of the system. The speed-up factor is given by $S_{latency} = \frac{La1}{La2}$, where $La1$ is the latency of the system when parallelism is equal to one and $La2$ is the latency of the system with the variation of the parallelism parameter. Latency is the processing time per message, which accounts for the difference between the time a message arrives at the processing and the time it leaves. Figure 14B compares the evaluated speed-up curve and the estimated Amdahl's Law for the theoretical speed-up of the system. The Amdahl's Law[30] is given by $S_{\text{latency}}(s) = \frac{1}{(1-p)+\frac{p}{s}}$, where s is the number of processing cores that are running the task, p is the parallel portion of the task, and S_{latency} is the theoretical speed-up in latency. In order to estimate the parallel portion of the task, we fit the Amdahl's Law to the experimental curve. We achieve the best fit for the experimental speed-up curve using the least square method. The fitting reveals that the $p = 0.815$, which means that approximately 80% of the task runs in parallel, while 20% is serial, much of which is due to communication overhead.

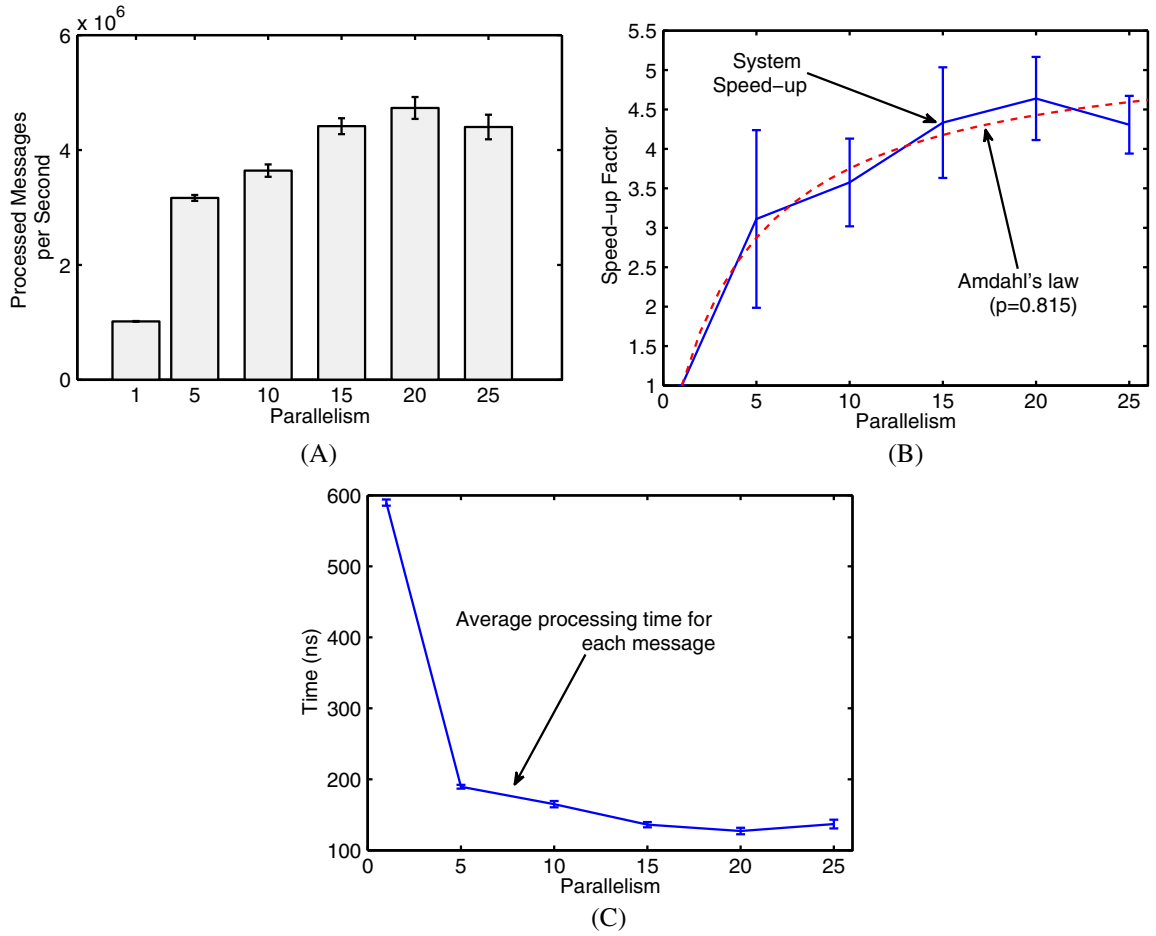

FIGURE 14 Scaling-out CATRACA system. (A) Processed-message throughput is maximum with 20 processing cores. (B) The speed-up factor follows the Amdahl's law for a 0.815 portion of parallel code. (C) The processing time per message is under-bounded by 100 ns. A, Processed-message throughput for CATRACA as a function of the number of processing cores; B, Speed-up factor as a function of the number of processing cores; C, Processing time per message as a function of the number of processing cores

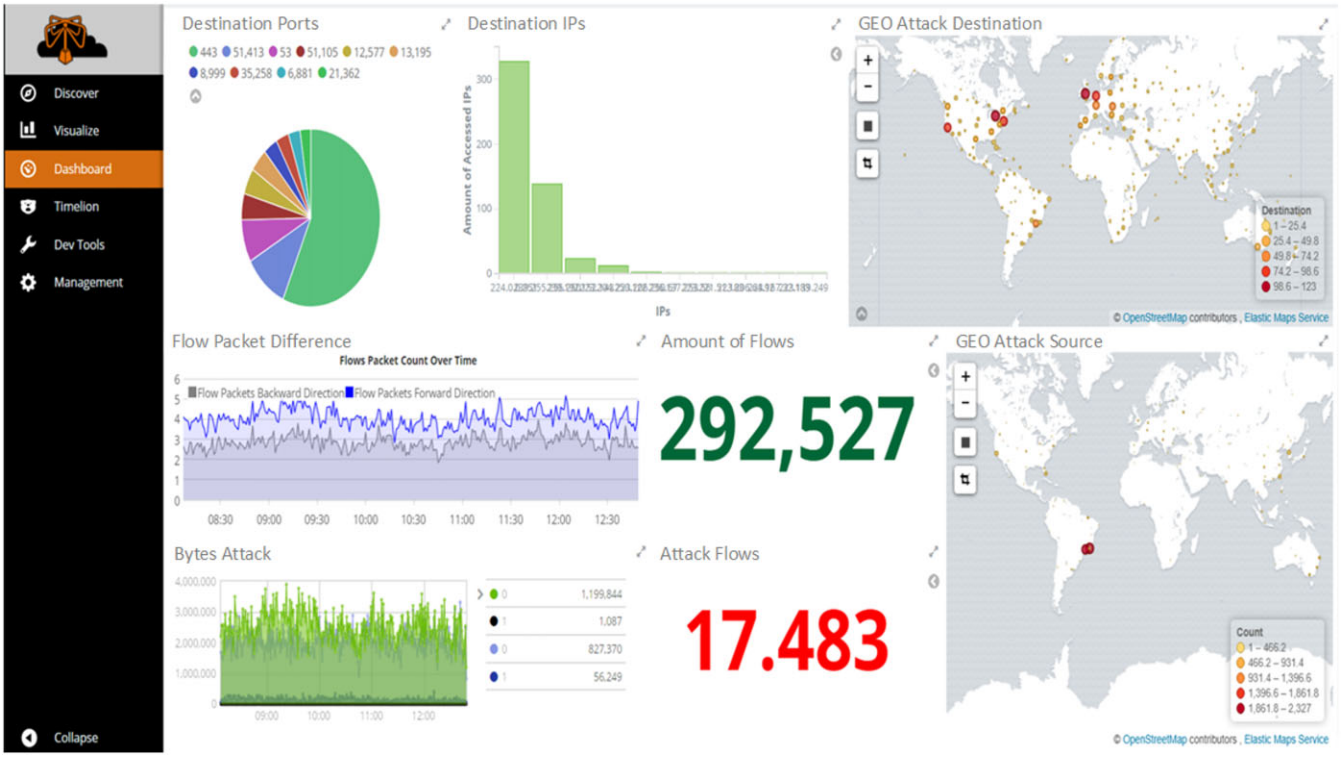

FIGURE 15 Dashboard panel view of CATRACA system. The dashboard is deployed as a Kibana application operating with streaming data in real-time; results are stored at ElasticSearch

Our proposed function can improve the processing capacity up to twenty cores in parallel. The same behavior is shown for latency and for throughput. The system can handle over five million flows per second as throughput; the latency to analyze a flow is shown in Figure 14C and reaches the minimum at 20-cores parallelism. Moreover, the latency speed factor with a parallelism of twenty is reached around 4.65, and this indicates that the system can parallelize almost five times with twenty virtual machines running one core each.

4.2.1 | Real-time visualization of enriched data

The visualization of the enriched data occurs through a simple and friendly web interface to allow the user to monitor the different parameters of the network in real time. The open source viewer Kibana, a component of the Elastic stack, was used for the development of the web interface, as it allows the visualization of the data in a fast and straightforward way allied to the performance of processing of queries with large volumes of data with low latency.

Figure 15 reveals some of the different scenarios shown in the control panel, such as the most accessed destination/source ports, the most used destination IP addresses, the average size of the flows in the round-trip directions, and the number of analyzed flows, among others. It is worth emphasizing the visualization of the attacks in progress through a map that portrays the origin, the destination, and the number of occurrences. It is possible for the enrichment of the data through the correlation with geolocation metadata in the processing module.[‖‖] Thus, both data and threats are viewed in real time. Moreover, all data are stored with a timestamp, allowing the processing of the data through time series.

4.3 | Analyzing streaming data use case

In stream processing Machine Learning methods, data arrive continuously, one sample at the time. Stream processing methods must process data under strict constraints of time and resources. Nevertheless, in case of a change in the stationary behavior of the sample statistics, the classification may present low accuracy, a behavior known as concept drift. An adaptive Machine Learning approach is necessary to ensure that the model is up-to-date. The adaptive approach must detect a change in data when new data is available. Since model training is resource consuming, we must ensure that a new model is essential. In an occurrence of concept drift, we must train a new model. We use the approach presented in Figure 16 to detect concept drift. Gray boxes present the batch processing, while the white boxes represent the streaming data.

Let $W = (w_1, w_2, \ldots, w_n)$ be a sliding window of samples. Each w_i contains N samples. We consider the first window w_n of size M, where $M \gg N$. This premise allows us to train a model before starting the process. Then, each window w_i is preprocessed. We apply feature scaling

[‖‖] For geolocation metadata, we use GeoIP library https://pythonhosted.org/python-geoip/. Accessed April 2018.

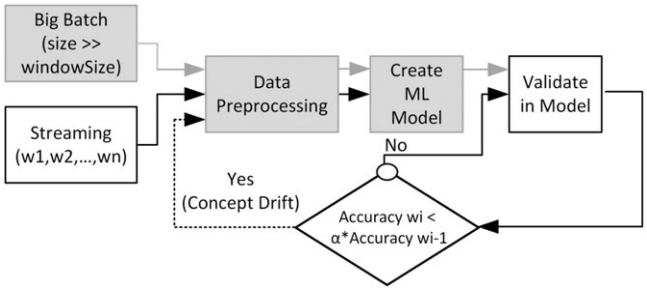

FIGURE 16 Flowchart used for concept drift detection

and normalization to consider all feature equally in their representation. Moreover, a feature selection algorithm is applied in this step to reduce processing time. Window w_i is validated with the model obtained in the first window w_0. Besides, the accuracy metric is monitored to verify if a concept drift is detected. We focus our approach on the degradation of the system performance. To measure system performance, we use the accuracy metric. Accuracy gives a notion of classification rate as the number of correct prediction over the total number of prediction, formally in Equation (1), where number of correct prediction is the True Positive (TP), and True Negative (TN) and the total number of prediction is the True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). The monitoring comprises analyzing the accuracy metric of each window. If the accuracy of window falls below a threshold α, it means that a concept drift is detected, and the current model is deprecated. As a result, a new model must be obtained from the values of the window w_i. We consider concept-drift, as represented in Equation (2), where $ConceptDrift = 1$ means a model re-training

$$Accuracy = \frac{TP + TN}{Number\ of\ Samples} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{1}$$

$$ConceptDrift = \begin{cases} 1, & \text{if } Acc_{w_i} < \alpha * Acc_{w_{i+1}} \\ 0, & otherwise \end{cases}. \tag{2}$$

In the experiment, we measure the impact of the concept-drift on the final accuracy. We analyze one day from NetOp dataset. As NetOp dataset contains labeled data, we can measure the model accuracy. In production environments, where labeled data are expensive, periodical re-training is usually performed. Periodical re-training can be unnecessary when the model remains unchanged. Other unsupervised approaches consider unlabeled data. Statistical comparison of arriving samples or clustering sample, with those samples used to train the system, assumes that a concept-drift is detected whenever new groups are found on new data.[31,32] These methods are resource consuming since measures based on distances are performed over arrived samples. We use decision tree machine learning algorithm. To perform the training in the first window, M is established in 10 000 samples, and we variate the value of N in 500, 1000, 2000, and 5000 samples. Moreover, we also variate the value of α from 80%, 85%, 90%, and 95% of accuracy. The lower is the value of α, the more the monitoring is permissive, and the more significant is the value of α, the more the system is rigorous. As NetOp dataset is real traffic for a network operator, no ground truth is presented for concept-drift detection. Thus, our definition of concept-drift is based on the degradation of the prediction performance.

Figure 17 shows the monitoring of the concept drift. For display purposes, we limited the number of windows to 1000. In Figure 17A, we see the accuracy behavior of a window with 500 samples. The variation of the α value shows a deep impact on accuracy. With a very strict value of $\alpha = 95\%$, the model is invalid for further windows, and accuracy maintains a value lower than 0.3. Similar behavior is shown with a value of $\alpha = 90\%$. A value $\alpha = 85\%$ shows an improvement; however, after a small number of windows, $\alpha = 85\%$ also maintains a low accuracy. The best results are shown with $\alpha = 80\%$, a more permissive value. This value allows performing an accuracy higher than 78% for over 600 window steps. Nevertheless, in CATRACA, we need to reach high accuracy. An improvement in accuracy is reached in Figure 17B, with a $\alpha = 80\%$, the system was able to maintain accuracy over 70% during all the experiments. Similar than before, stricter values of α do not perform correctly. Figure 17C and 17D show an interesting behavior. In Figure 17C, the values of 80% and 85% if α overlap, maintaining high accuracy over 70%. With a sample of 5000, in Figure 17D, the overlap is produced by three values, $80\%, 85\%$, and 90% of α values. Nonetheless, the strict value of 95 follows a similar behavior at the beginning, and even if this value presents a low accuracy during a small period, $\alpha = 90\%$ presents the highest accuracy of all the experiment.

In this experiment, we show the impact of the concept drift detection on the variation in the sliding window and the accuracy. In Figure 18A is presented the number of concepts drift detected while varying the sliding window and the α value. On the other hand, in Figure 18 is analyzed the mean accuracy when varying the same parameters. With a strict value of $\alpha = 95\%$ and with a sliding window of 500, a high rate of concept drift is detected. However, when compared with the same values in Figure 18, the mean accuracy is below 0.35. Differently, with a more permissive value of $\alpha = 80\%$, the concept drift detection is below 500 in the whole dataset. Nevertheless, the accuracy with the same values is under 0.4. With a permissive value of $\alpha = 80\%$, no concept drift is detected, but the accuracy is kept in more than 70%. We conclude that a good performance is presented when the sliding window contains N samples that tend to be high and with a more permissive value of α.

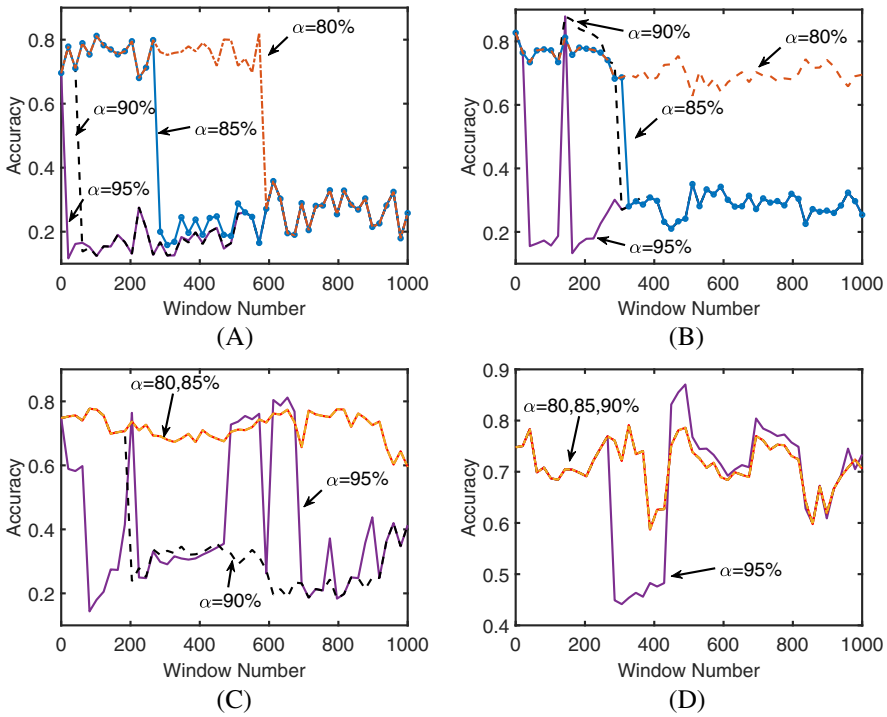

FIGURE 17 Comparison of the accuracy with the variation of the samples in the sliding window. Moreover, the parameter α is variate to analyze the impact in the accuracy. A, Window containing 500 samples; B, Window containing 1000 samples; C, Window containing 2000 samples; D, Window containing 5000 samples

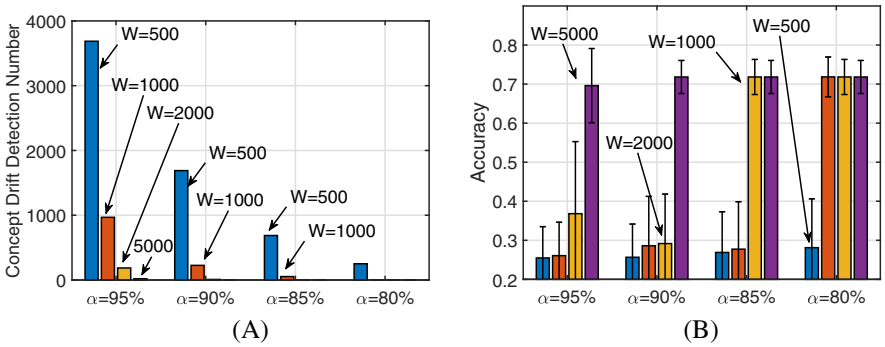

FIGURE 18 A good performance is presented when the sliding window contains a number of samples N that tends to be high and with a more permissive value of α. A, Relation of concept drift detection and α value; B, Mean accuracy with the variation of α value

As NetOp dataset is a real network traffic trace collection, there is no ground-truth knowledge for how many concept drifts. Thus, we compare our concept-drift detection dataset with the Page-Hinkley (PH) test. Page-Hinkley is a technique used to detect modifications of a signal processing assuming a Gaussian distribution produced it. For every sample x_i, PH test calculates a cumulative variable m_n based on the Equation (3). The variable m_n is the difference between the sample x_i, Equation (4), and the mean of the last n observed values $\bar{x}_n$. If the difference is smaller to a value θ, a concept drift is detected. The value θ corresponds to the magnitude of changes that are allowed. In the experiments, we use $\theta = 0.005$ and $\lambda = 50$. We use the decision tree algorithm as a machine learning model for classification

$$m_n = \sum_{i=1}^{n} (x_i - \bar{x}_n - \theta) \leq \lambda \tag{3}$$

$$\bar{x}_n = \frac{1}{N} \sum_{i=1}^{n} x_i. \tag{4}$$

We compare our proposal with the PH test in three different datasets. One day of NetOp dataset, validating the previous results, NSL KDD, and SpamsAssasins dataset.[33] NSL KDD and SpamsAssisins are used in the work of Sethi and Kantardzic[32] to detect concept drift in unlabeled data. Table 5 represents the results obtained. Table 6 shows the comparison of our proposal with Page-Hinkley test in NSL-KDD and SpamsAssasins datasets. We see that our approach does not achieve to detect concept-drift with small windows size. Nevertheless, our approach detects only one concept drift in all window size. On the other hand, for bigger windows size, our approach outperforms Page-Hinkle test. In NSL-KDD dataset, the accuracy of our approach remains constant in 0.98, and for PH test, the best accuracy 0.53 with a window size of 100. In SpamAssassin,

TABLE 5 Results obtained by Sethi and Kantardzic

Dataset	Mean Accuracy	Drift Signaled
NSL KDD	0.893	1
SpamsAssasins	0.853	2

TABLE 6 Concept-Drift detection. Comparison of our proposal with Page-Hinkley test in NSL-KDD dataset and SpamsAssasins dataset

		NSL-KDD Dataset		SpamsAssasins Dataset	
Method	Window Size	Mean Accuracy	Drift Detected	Mean Accuracy	Drift Detected
Page-Hinkley	50	0.69	0	0.99	0
Proposal $\alpha = 80$	50	0.98	0	0.79	2
Proposal $\alpha = 85$	50	0.98	0	0.81	4
Proposal $\alpha = 90$	50	0.98	0	0.80	8
Proposal $\alpha = 95$	50	0.98	0	0.80	11
Page-Hinkley	100	0.53	2	0.98	0
Proposal $\alpha = 80$	100	0.98	0	0.83	0
Proposal $\alpha = 85$	100	0.98	0	0.83	0
Proposal $\alpha = 90$	100	0.98	1	0.78	3
Proposal $\alpha = 95$	100	0.98	1	0.81	7
Page-Hinkley	200	0.52	4	0.98	0
Proposal $\alpha = 80$	200	0.98	1	0.83	0
Proposal $\alpha = 85$	200	0.98	1	0.83	0
Proposal $\alpha = 90$	200	0.98	1	0.83	0
Proposal $\alpha = 95$	200	0.98	1	0.82	2
Page-Hinkley	500	0.52	6	0.97	0
Proposal $\alpha = 80$	500	0.98	1	0.83	0
Proposal $\alpha = 85$	500	0.98	1	0.83	0
Proposal $\alpha = 90$	500	0.98	1	0.83	0
Proposal $\alpha = 95$	500	0.98	1	0.79	1
Page-Hinkley	1000	0.52	8	0.96	0
Proposal $\alpha = 80$	1000	0.98	1	0.83	0
Proposal $\alpha = 85$	1000	0.98	1	0.83	0
Proposal $\alpha = 90$	1000	0.98	1	0.83	0
Proposal $\alpha = 95$	1000	0.98	1	0.78	0
Page-Hinkley	2000	0.52	10	0.93	0
Proposal $\alpha = 80$	2000	0.98	1	0.83	0
Proposal $\alpha = 85$	2000	0.98	1	0.83	0
Proposal $\alpha = 90$	2000	0.98	1	0.83	0
Proposal $\alpha = 95$	2000	0.98	1	0.78	0

dataset Page-Hinkle accuracy performs between 0.93 and 0.99, while our approach is between 0.78 and 0.83. Our approach detects the same amount of changes than the aforementioned work approach while obtaining a better accuracy in NSL-KDD, and for SpamsAssasins dataset, we were able to detect the same amount for $\alpha = 80, 95$ with different window sizes.

5 | CONCLUSION

CATRACA system proposed a virtualized network function for real-time and offline threat detection. We perform threat detection using machine learning algorithms running on a stream processing platform. The proposed system achieves threat analysis on incoming traffic in real time or on a historical base in differentiated time. CATRACA runs in two different modes online and offline. Thus, we applied three different uses cases. The first use case analyzes and characterizes a network operator big dataset. The second use case performs traffic classification under three different security datasets. Finally, the last use case shows the behavior of CATRACA when dealing with stream data in real time. Our system was able to adapt the classifiers under concept drifting, a change in the relationships between the created model and output data. The system runs on OPNFV open source platform as a virtual network function and displays the knowledge extracted from the enriched data through a friendly graphical

interface for viewing different analyses and the geographical location of the source and destination of the threats in real time. The system can be obtained at http://www.gta.ufrj.br/catraca, where the user manual, which details the procedures of installation and use of the system, the documentation, which allows understanding the project and more details about the system code, as well as other useful information are provided.

ACKNOWLEDGMENTS

The authors would like to thank Antonio Gonzalez Pastana Lobato, Igor Drummond Alvarenga, and Igor Jochem Sanz for their significant contributions to obtain the results. This research is supported by CNPq, CAPES, FAPERJ, and FAPESP.

ORCID

Martin Andreoni Lopez https://orcid.org/0000-0002-4170-4341

REFERENCES

1. Bär A, Finamore A, Casas P, Golab L, Mellia M. Large-scale network traffic monitoring with DBStream, a system for rolling big data analysis. Paper presented at: IEEE International Conference on Big Data; 2014; Washington, DC.
2. Hu P, Li H, Fu H, Cansever D, Mohapatra P. Dynamic defense strategy against advanced persistent threat with insiders. Paper presented at: 2015 IEEE Conference on Computer Communications (INFOCOM) IEEE; 2015; Kowloon, Hong Kong.
3. Verizon Enterprise. Data Breach Investigations Report. www.verizonenterprise.com/resources/reports/rp_DBIR_2016_Report_en_xg.pdf. 2016.
4. Wu K, Zhang K, Fan W, Edwards A, Yu PS. RS-forest: a rapid density estimator for streaming anomaly detection. Paper presented at: IEEE International Conference on Data Mining (ICDM); 2014; Shenzhen, China.
5. Mayhew M, Atighetchi M, Adler A, Greenstadt R. Use of machine learning in big data analytics for insider threat detection. Paper presented at: IEEE Military Communications Conference (MILCOM); 2015; Tampa, FL.
6. Toshniwal A, Taneja S, Shukla A, et al. Storm@Twitter. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data; 2014; Snowbird, UT.
7. Carbone P, Fóra G, Haridi S, Tzoumas K. Lightweight asynchronous snapshots for distributed dataflows. arXiv preprint arXiv:1506.08603. 2015.
8. Franklin M. The Berkeley data analytics stack: present and future. Paper presented at: IEEE International Conference on Big Data IEEE; 2013; Silicon Valley, CA.
9. Lopez MA, Lobato AGP, Duarte OCMB, Pujolle G. An evaluation of a virtual network function for real-time threat detection using stream processing. Paper presented at: IEEE Fourth International Conference on Mobile and Secure Services (MobiSecServ); 2018; Miami Beach, FL.
10. Lopez MA, Mattos DMF, Duarte OCMB. An elastic intrusion detection system for software networks. *Ann Telecommun*. 2016;71(11-12):595-605.
11. Du Y, Liu J, Liu F, Chen L. A real-time anomalies detection system based on streaming technology. Paper presented at: Sixth International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC); 2014; Hangzhou, China.
12. Zhao S, Chandrashekar M, Lee Y, Medhi D. Real-time network anomaly detection system using machine learning. Paper presented at: 11th International Conference on the Design of Reliable Communication Networks (DRCN); 2015; Kansas City, MO.
13. He G, Tan C, Yu D, Wu X. A real-time network traffic anomaly detection system based on storm. Paper presented at: 7th International Conference on Intelligent Human-Machine Systems and Cybernetics; 2015; Hangzhou, China.
14. Villar-Rodríguez E, Del Ser J, Torre-Bastida AI, Bilbao MN, Salcedo-Sanz S. A novel machine learning approach to the detection of identity theft in social networks based on emulated attack instances and support vector machines. *Concurrency Computat Pract Exper*. 2016;28(4):1385-1395. https://doi.org/10.1002/cpe.3633
15. Li B, Zhang S, Li K. Towards a multi-layers anomaly detection framework for analyzing network traffic. *Concurrency Computat Pract Exper*. 2017;29(14):e3955.
16. Lee W, Stolfo SJ, Mok KW. Mining in a data-flow environment: experience in network intrusion detection. In: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 1999; San Diego, CA.
17. Santos O. *Network Security with NetFlow and IPFIX: Big Data Analytics for Information Security*. Indianapolis, IN: Cisco Press; 2015.
18. Apache Software Foundation. Apache Metron. Apache. 2017.
19. Jirsik T, Cermak M, Tovarnak D, Celeda P. Toward stream-based IP flow analysis. *IEEE Commun Mag*. 2017;55(7):70-76.
20. Sanz IJ, Mattos DMF, Duarte OCMB. SFCPerf: an automatic performance evaluation framework for service function chaining. Paper presented at: IEEE/IFIP Network Operations and Management Symposium (NOMS 2018); 2018; Taipei, Taiwan.
21. Jeon H, Lee B. Network service chaining challenges for VNF outsourcing in network function virtualization. Paper presented at: International Conference on Information and Communication Technology Convergence (ICTC); 2015; Jeju Island, South Korea.
22. Lopez MA, Lobato AGP, Duarte OCMB. A performance comparison of open-source stream processing platforms. Paper presented at: IEEE Global Communications Conference (GLOBECOM); 2016; Washington, DC.
23. Tavallaee M, Bagheri E, Lu W, Ghorbani AA. A detailed analysis of the KDD CUP 99 data set. In: Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications; 2009; Ottawa, Canada.
24. Lobato AGP, Lopez MA, Sanz IJ, Cárdenas A, Duarte OCMB, Pujolle G. An adaptive real-time architecture for zero-day threat detection. Paper presented at: IEEE International Conference on Communications (ICC); 2018; Kansas City, MO.
25. Lopez MA, Silva SR, Alvarenga ID, et al. Collecting and characterizing a real broadband access network traffic dataset. Paper presented at: 1st Cyber Security in Networking Conference (CSNet); 2017; Rio de Janeiro, Brazil.

26. Jolliffe I. *Principal Component Analysis*. Berlin, Germany: Springer; 2011. *International Encyclopedia of Statistical Science*.
27. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: synthetic minority over-sampling technique. *J Artif Intell Res*. 2002;16:321-357.
28. He H, Bai Y, Garcia EA, Li S. ADASYN: adaptive synthetic sampling approach for imbalanced learning. In: Proceedings of the 5th IEEE International Joint Conference on Neural Networks; 2008; Hong Kong.
29. Sommer R, Paxson V. Outside the closed world: on using machine learning for network intrusion detection. Paper presented at: IEEE Symposium on Security and Privacy; 2010; Berkeley/Oakland, CA.
30. Zahedi SM, Llull Q, Lee BC. Amdahl's law in the datacenter era: a market for fair processor allocation. Paper presented at: IEEE International Symposium on High Performance Computer Architecture (HPCA); 2018; Vienna, Austria.
31. Jordaney R, Sharad K, Dash SK, et al. Transcend: detecting concept drift in malware classification models. In: Proceedings of the 26th USENIX Conference on Security Symposium; 2017; Vancouver, Canada.
32. Sethi TS, Kantardzic M. On the reliable detection of concept drift from streaming unlabeled data. *Expert Syst Appl*. 2017;82:77-99.
33. Katakis I, Tsoumakas G, Vlahavas I. Tracking recurring contexts using ensemble classifiers: an application to email filtering. *Knowl Inf Syst*. 2010;22(3):371-391. https://doi.org/10.1007/s10115-009-0206-2

How to cite this article: Andreoni Lopez M, Mattos DMF, Duarte OCMB, Pujolle G. Toward a monitoring and threat detection system based on stream processing as a virtual network function for big data. *Concurrency Computat Pract Exper*. 2019;e5344. https://doi.org/10.1002/cpe.5344