# TeMIA-NT: ThrEat Monitoring and Intelligent data Analytics of Network Traffic

Lucas C. B. Guimarães, Gabriel Antonio F. Rebello, Felipe S. Fernandes,
Gustavo F. Camilo, Lucas Airam C. de Souza, Danyel C. dos Santos,
Luiz Gustavo C. M. de Oliveira e Otto Carlos M. B. Duarte

Grupo de Teleinformática e Automação (GTA/PEE/COPPE)
Universidade Federal do Rio de Janeiro (UFRJ)

***Abstract*—Cybernetic attacks have been increasingly common and cause great harm to people and organizations. Late detection of such attacks increases the possibility of irreparable damage, with high financial losses being a common occurrence. This article proposes TeMIA-NT (*ThrEat Monitoring and Intelligent data Analytics of Network Traffic)*, a real-time flow analysis system that uses parallel flow processing. The main contributions of the TeMIA-NT are: i) the proposal of an architecture for real-time detection of network intrusions that supports high traffic rates, ii) the use of the structured streaming library, and iii) two modes of operation: offline and online. The offline operation mode allows evaluating the performance of multiple machine learning algorithms over a given dataset, including metrics such as accuracy, F1-score, and area under the curve (AUC). The proposal uses dataframe structures, in online mode, the structured streaming library in continuous mode, which allows detection of threats in real-time and a quick reaction to attacks. To prevent or minimize the damage caused by security attacks, TeMIA-NT achieves flow-processing rates that reach 50 GB/s.**

## I. Introduction

Cybercrime is one of the major challenges introduced by the exponential growth of the Internet. According to Cybersecurity Ventures [1], damages related to cyber attacks are projected to reach US$6 trillion by 2021. Besides, the growth and popularization of areas such as Big Data and the Internet of Things pose even more significant challenges to cybersecurity. The introduction of billions of low power computing devices connected to the network increases the impact of possible attacks, as these devices can be easily hacked and compromised on a large scale [2], [3], [4]. The large volume of data to be analyzed in real-time also increases the complexity of classifying network traffic and detecting threats [5]. Finally, the average time to detect an attack is a crucial factor in the impact of cyber threats. More than a quarter of cyberattacks take a long time before being discovered, with this time often ranging from weeks to months [6]. The late detection of an attack exponentially increases the risk of financial losses and the risk of irreparable damage. The long time is due to the need for human intervention in these situations, significantly affecting the efficiency of dealing with threats

In this scenario in which security is a fundamental aspect, the need for systems capable of guaranteeing safe and reliable network use is increasing. Solutions based on Security Information and Event Management (SIEM) tools partially mitigate the problem by providing real-time network monitoring. This type of solution, however, is still highly dependent on the intervention of experts and is based on threat signature databases, therefore being inefficient in the detection of new attacks. Using machine learning algorithms for threat detection, on the other hand, automates the detection process and meets the required agility to prevent and mitigate network attacks. It is of utmost importance to select algorithms that perform well in the classification process, without harming accuracy and other evaluation metrics. Previously, our research group (GTA/UFRJ) proposed CATRACA [7], a tool that uses machine learning to detect threats in real-time.

This paper proposes TeMIA-NT: Intelligent Monitoring and Analysis of Network Traffic Threats, an intelligent threat monitoring and detection system based on machine learning and distributed processing in clusters. TeMIA-NT proposes and develops an entirely new distributed processing system with significant improvements in machine learning processing optimization. Our proposal focuses on the intelligence, scalability, and performance required to process large volumes of data while optimizing multiple machine learning algorithms to meet the diversity of new attacks. To increase performance, TeMIA-NT implements distributed processing entirely in Scala language and uses dataframe structures, instead of the standard Resilient Distributed Datasets (RDD) structure on the open-source Apache Spark platform. TeMIA-NT offers many options for machine learning algorithms and the possibility of optimizing hyperparameters, allowing testing, selecting, and adjusting the parameters of the best algorithm for each type of scenario. We implement the offline threat detection using the structured streaming library, which allows flow processing in micro-batches, with fault tolerance and reduced intervals. Online threat detection uses the continuous processing mode, which enables our proposal to perform similar to a native stream processing tool.

The rest of the article is organized as follows. Section II presents papers with themes related to the article. Section III introduces the Apache Spark platform, as well as its data structures and its machine learning library. Section IV presents the network traffic dataset used to test the proposed tool, while Section V presents the tool's architecture and features. Section VI presents and analyzes the performance tests and

their results, and Section VII presents the author's final considerations and concludes the work

## II. Related Works

New challenges in the intrusion detection area arise due to the high volume of traffic, a large number of IoT devices, distributed denial of service attacks, and zero-day attacks [8], [9], [10]. To meet these challenges, the use of machine learning techniques to classify flows in real-time became popular [7], [11], [12], [13]. The classification of large volumes of data at high speeds available employs three main distributed processing platforms: Apache Spark, Apache Storm, and Apache Flink. The fundamental difference between the platforms is that Spark performs batch processing while the Storm and Flink platforms perform native flow processing.

The Open Security Operations Center (OpenSOC) [14] is an analytical security framework for monitoring large amounts of data. OpenSOC originated a new project, Apache Metron [15], that is a tool that comprises the acquisition of different types of data, distributed processing, enrichment, storage, and visualization of results. Metron allows the correlation of security events from various sources, such as logs of applications and network packages. For this purpose, the framework uses distributed data sources, such as sensors on the network, logs of security element events, and enriched data called telemetry sources. The tool also provides a historical base of Cisco network threats.

Based on the Apache Spark Platform [16], there are the Apache Spot, Stream4Flow [17], and Hogzilla. Apache Spot is a project still in the incubation stage that uses telemetry and machine learning techniques for analyzing packages to detect threats. The Stream4Flow prototype uses the Elastic stack to view network parameters, however, it lacks the intelligence to perform anomaly detection. The Hogzilla tool[1] provides support for Snort, SFlows, GrayLog, Apache Spark, HBase, and libnDPI, offering network anomaly detection. Hogzilla also allows visualizing network traffic, using Snort to capture packets, and obtaining features through deep packet inspection. Stream4Flow captures packets using IPFIXcol and only considers header information. In our work, we use the flowtbag[2] software, which captures various flow statistics. In addition, our offline processing mode allows updates to the machine learning model, further promoting the detection of new threats.

We select the Apache Spark platform to develop the TeMIA-NT because it is the most adopted among the examined Big Data processing platforms. Spark offers more possibilities for machine learning algorithms and is the one with the largest active community. Nevertheless, to the best of our knowledge, TeMIA-NT is the only available system to use the recent structured streaming technology in batch and continuous modes in Apache Spark, allowing the selection among several machine learning algorithms, and operating in offline and online modes

[1]http://ids-hogzilla.org/, accessed in July 2020.

[2]https://github.com/DanielArndt/flowtbag, accessed in July 2020.

## III. The Apache Spark Platform

We use Apache Spark [16], a distributed processing platform that allows Big Data processing, providing an interface for programming in clusters with parallelism and fault tolerance, to develop the tool in this paper. We chose the Spark platform due to its efficiency, its great acceptance in the market, and because it has a wide library of machine learning algorithms. The platform also supports multiple programming languages, including Python, Scala, R, and Java.

The main feature of Apache Spark is how to process data: memory does all operations that involve reading and writing intermediate results. Spark is efficient for applications that perform multiple data transformation iterations in a distributed environment, avoiding time-consuming disk operations [18].

The Spark platform provides several libraries, such as Spark Streaming for real-time flow processing and GraphX for parallel graph computing. Also, Spark provides MLlib, a library that implements parallelizable and efficient machine learning algorithms in a distributed environment, making the platform an option for classifying network traffic. We use algorithms from the MLlib library to do performance analysis, which creates the machine learning models used for traffic classification.

### A. Data Structures

Because of the growing impact of Big Data, Zaharia *et al.* designed and developed Apache Spark to provide enterprise-level distributed processing for large datasets [16]. The data structures used by Spark play an essential role in the fast and efficient data processing, being responsible for its organization, management, and storage; they also provide functions and operations to make more efficient data processing.

*1) Resilient Distributed Datasets:* The first Spark data structure developed for distributed processing was resilient and distributed datasets (RDD). This structure is an immutable and, therefore, resilient dataset, partitioned in the cluster nodes. It can be operated by a low-level API, offering multiple transformations and functions. A crucial feature of this data structure is to provide computing resources in memory, providing the agility observed in Spark operations. Another essential feature is the use of lazy evaluation, which computes expressions or functions only when their results are needed, optimizing the execution time by avoiding unnecessary calculations. RDD also offers fault tolerance: each RDD can reconstruct lost data automatically, based on data within other nodes in the cluster. Since RDDs are immutable, they can be created or retrieved at any time, making data sharing and replication a simple process.

*2) DataFrame & Dataset:* DataFrames and Datasets are the other data structures implemented by Apache Spark. These structures differ from RDD in that they are structured as tables in a relational database: RDDs do not specify rows and columns, then queries in RDDs with a large number of records require longer periods to complete. On the other hand, DataFrames and Datasets follow a schema, which lists the

columns and the information they contain. As the data implemented through DataFrames and Datasets are structured, Spark implements performance optimizations in terms of processing time and memory consumption through the Tungsten [19] and Catalyst Optimizer [20] projects.

These data structures act similarly, differing only in terms of type handling: Datasets implement a strongly typed API, while DataFrames implement an untyped API. An untyped API allows parsing errors to go unnoticed during compilation time. Differently, a strongly-typed API detects these errors at compile-time, reducing the possibility of errors occurring during the execution of the program. Since Python and R do not have compile-time type security, these languages only implement DataFrames.

### *B. MLlib Library*

The purpose of the MLlib [21] library is to allow the use of machine learning techniques on the Apache Spark platform, implementing them in an efficient and scalable way through a high-level API. These techniques include standard classification, regression, clustering, and collaborative filtering machine learning algorithms, such as decision tree, linear regression, k-means, alternating least square, among others.

The library also offers featurization methods, allowing the Apache Spark platform to carry out the pre-processing of datasets before machine learning methods are applied. The application includes techniques that reduce dimensionality and rely on both the selection and extraction of features. These methods also allow the transformation of those features, such as normalization.

MLlib also provides multiple utilities to facilitate data processing, including statistical methods used to obtain results in terms of evaluation metrics, such as accuracy and AUC, and linear algebra methods. There are also methods responsible for optimizing the execution pipelines, allowing algorithms and models to be saved and loaded from memory as necessary.

Since they possess different logic and assume different characteristics of the input data, these machine learning algorithms present different results depending on the target problem. Therefore, it is important to evaluate which algorithms offer the best results for the treated problem that is the analysis and classification of network traffic. As such, the following algorithms made available through the MLlib library were implemented in the TeMIA-NT tool, so that the user can verify which offers the best solution to their dataset.

**Decision Tree**: The decision tree algorithm builds a tree in which each internal node evaluates a data feature. Each branch represents a decision around a possible value for the selected feature, and each final node in a branch indicates the class the element is most likely to belong to. Thus, the algorithm traverses the tree branches and evaluates the features of each node to estimate the sample probability to belong to a particular class. A great advantage of the decision tree algorithm is its ease of understanding and interpretation, being composed exclusively by rules in the "if-then-else" format.

**Naïve Bayes**: The naïve Bayes algorithm is a probabilistic classifier that works through the application of Bayes' theorem. This theorem, shown in the Equation 1, indicates the probability that an event $c$ will occur knowing that a given event $x$ has happened. The parameters used by the equation are the *a priori* probabilities of $c$ and $x$, as well as their likelihood.

$$P(c|x) = P(c)\frac{P(x|c)}{P(x)} \tag{1}$$

Since the algorithm performs the classification through a simple mathematical calculation, resulting in linear execution time, it is easily scalable for large datasets and several features. However, the accuracy obtained by this classifier may be lower than that obtained by other algorithms, since it assumes that analyzed elements are statistically independent, which may not be valid depending on the chosen dataset.

**Logistic Regression**: Logistic regression is a statistical model that approximates the *a posteriori* probability of the positive class, using the *sigmoid* function as a discriminant function, whose formula can be seen in the Equation 2. Binary classification problems usually employ this method.

$$f(x) = \frac{1}{1 + e^{-k(x-x_0)}} \tag{2}$$

**Multilayer Perceptron**: Multilayer perceptron is a neural network model that works by employing multiple perceptrons, which act as the network's "neurons", who are delegated the tasks of performing small calculations and forwarding their results to other perceptrons. The perceptrons are organized in layers, with each perceptron in one layer being fully connected to the perceptrons in the next layer. The first layer receives the input features from the dataset, while the last layer represents the classification results.

Each perceptron uses an activation function to connect with others, based on the results of the previous layers and the adjusted weights for each output connection. These activation functions are non-linear, allowing the acquisition of non-linear models, but increasing the time required to obtain the model. One of the most used activation functions is the logistic (or sigmoid) function, previously indicated in the Equation 2; others include the hyperbolic tangent function and ReLU.

Another technique used by the multilayer perceptron is backpropagation; this algorithm works by calculating the gradient of the loss function concerning each weight by the chain rule, iterating one layer at a time from the last layer to avoid redundant calculations of intermediate terms. In this way, it is possible to update the weights of each layer to minimize losses.

**Random Forest**: The random forest is an ensemble learning algorithm that works by creating multiple decision trees and whose classification result is defined as the mode of the classification results obtained by each tree. This method usually presents better results than those obtained by working with only one decision tree, while also offering a lower risk of overfitting; however, it has a considerably longer processing

time and is not scalable for large datasets without sacrificing the model's performance.

**Gradient-boosted Tree**: As with the random forest algorithm, the gradient-boosted tree is an ensemble learning algorithm based on decision tree models. Unlike random forest, where each tree is trained individually, trees in this algorithm are trained iteratively, with new trees using the prediction of previous trees to offer a more accurate model. This algorithm also has a high processing time and is not ideal for large datasets.

**Support Vector Machines**: The support vector machines (SVM) method initially works by mapping the input features into a larger space, called the feature space; this change allows a model that is difficult to separate in the original space to be separable by a hyperplane in the feature space. The method performs the separation between classes by defining this hyperplane, aiming to maximize the separation margin between the points closest to each of the classes.

An important aspect of this method is the definition of the chosen kernel function, responsible for the mapping done in the feature space. Multiple kernel functions exist, with some being: RBF, polynomial, hyperbolic tangent, among others.

### C. Structured Streaming

The real-time processing on the Apache Spark platform was initially implemented through the Spark Streaming library, which allows continuous processing of RDDs through the DStream API. With the introduction of DataFrame and Dataset as new data structures, the Structured Streaming library was developed to handle these structures in real-time while maintaining the optimizations they introduced.

Structured Streaming allows the programmer to program in a similar way to the one in batch data processing, with the platform dealing with the implementation of specific flow processing techniques through a high-level API. Structured Streaming implements the micro-batch technique, with data received within a certain time interval being added to a batch to be processed; after processing, the result is added to a table, and the elements of the processed batch are discarded. Other advantages of the library include "exactly once" fault tolerance, as well as end-to-end latency of up to 100 milliseconds.

Another processing method provided by the library is the continuous processing mode. This mode allows latency as low as one millisecond but does not offer all the functions of the main library, supporting only projection and selection operations. It also has "at least once" fault tolerance, leaving aside the advantages of tolerating exactly once of the other processing method.

## IV. Dataset and Schema

A crucial aspect of the development of an intrusion detection system (IDS) is the need to check its performance before it goes into operation. Thus, a dataset is used that contains both legitimate and malicious traffic. The most commonly used dataset in IDS development is the NSL-KDD [22], with other important datasets being the DARPA98 and the DARPA99. However, these and other datasets are often not recent, and in addition to using synthetic attack patterns and threats, may not portray the features of current network traffic.

The dataset used was obtained from traffic from a telecommunications operator [23], converted into flows using the flowtbag tool. Each flow is a sequence of packets, within a time window, which has certain features in common. The features used to group packets in flows were the 5-tuple (source IP, destination IP, source port, destination port, protocol), set commonly used in traffic analysis works. Grouping packages into flows, the flowtbag tool extracts 40 features for the construction of the data schema, including the number of packages sent and received, the minimum and maximum sizes of a package, among others.

The labeling of the dataset flows as legitimate or malicious, necessary for the creation of models in supervised machine learning algorithms, was done through IDS Suricata. The dataset was also balanced to avoid bias during the model training and test phases, therefore being composed of equal parts of legitimate and malicious traffic.

## V. The proposed architecture

The proposed tool has two operation modes: online and offline. The online mode performs classification in real time, whilst the offline mode allows to observe the performance of multiple classifiers for a given dataset, making the resulting metrics available in the visualization module. The proposed architecture, shown in Figure 1, is modular and consists of three main modules: data collection, processing and visualization.

The data collection module captures and abstracts network traffic flows. It also stores the datasets used in offline processing. The capture process reflects network traffic through the libpcap library. Then, the flowtbag tool abstracts the sequence of packets in the flows and their 40 features, including the flows length and the total number of packages for each flow. We use the five fields of the TCP/IP packet header, source IP address, destination IP address, source port, destination port, and protocol to abstract packets into flows. A channel on the Apache Kafka platform, which acts as a data buffer, receives the streams of data. We use the Hadoop Distributed File System (HDFS), a distributed database, to store the datasets used to train and test the classification models.

The processing module carries out the process of classifying these flows. The processing module is implemented in an Apache Spark cluster. This platform presents advantages to the development of the tool, as it has libraries aimed at the implementation of machine learning algorithms and the fast processing of data in real-time, using the micro-batch method with the structured streaming engine. The training module extracts the classification model using a dataset labeled from HDFS. In the online mode of operation, packages are collected and added to an Apache Kafka channel and flows are then classified as legitimate or malicious by the classification model obtained previously. To allow flows to be analyzed and classified in real-time in the online processing mode, we developed a classification module responsible for collecting

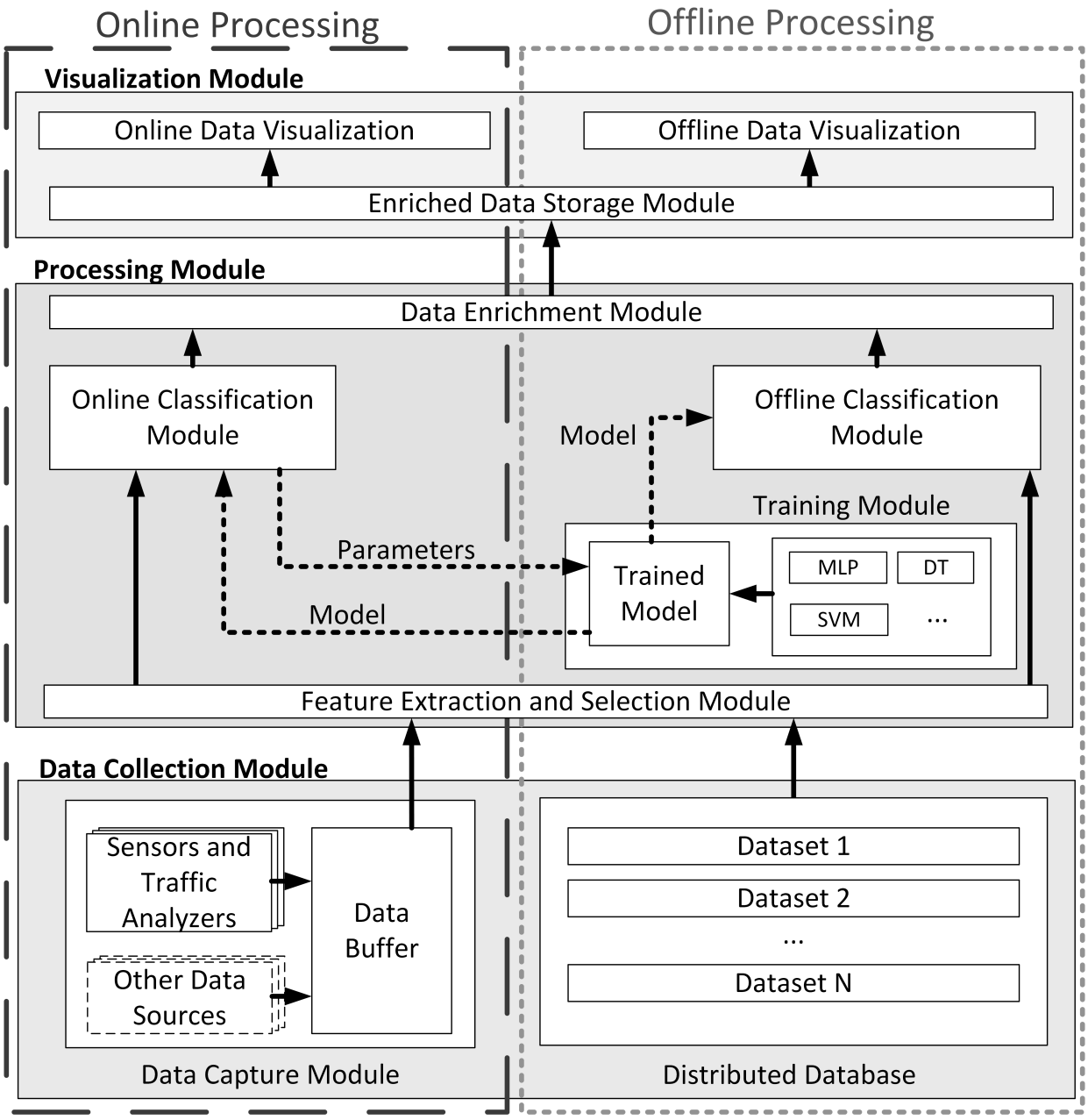


Fig. 1: TeMIA-NT modular architecture at online and offline modes.

this data as they are added to a certain Apache Kafka channel. The flows are then classified as legitimate or malicious by the classification model obtained previously. For execution in the offline mode, the classification module runs tests on various algorithms and datasets, obtaining performance metrics for each combination. The results of the classification, both online and offline, are then sent to an Elasticsearch server using the Apache Spark integration library.

The visualization module allows the network administrator to visualize the classifications history and the current state of the network, as well as the results of tested algorithms. We implement the visualization module using the Elasticsearch[3] and Kibana[4] software, both developed by Elastic. Elasticsearch implements a distributed and efficient search server, based on JSON documents. It receives and stores the data as the processing module sends it after the classification process is finished. Kibana is responsible for providing a user interface through dashboards, displaying to the network administrator the data received by Elasticsearch in real-time for both execution modes. It also allows consultation by historical data, using the search server features Elasticsearch.

[3]https://github.com/elastic/elasticsearch, accessed in July 2020.
[4]https://github.com/elastic/kibana, accessed in July 2020.

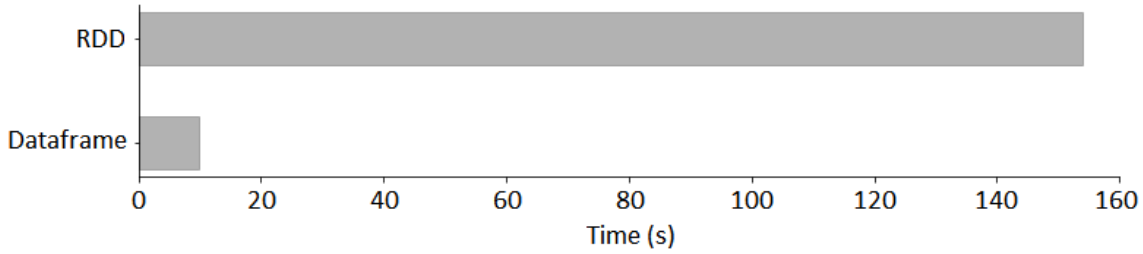


Fig. 2: Impact of the data structure on the training time of the decision tree.

## VI. Performance Analysis

This section presents the tests carried out and the analysis of their respective results. It introduces the chosen evaluation metrics to allow a more natural understanding of the terms used.

The test environment is a cluster composed up of 4 machines, of which 1 is the master and 3 are slaves. The master has 2 Xeon X5570 processors with 4 physical cores per processor, as well as 96 GB of DDR3 RAM. The slaves have 2 Xeon E5-2650 processors with 8 physical cores per processor and 32 GB of DDR3 RAM each. All machines use the Ubuntu 19.04 operating system.

### A. Evaluation Metrics

For comparison purposes, evaluation metrics were used to verify the results obtained by each studied algorithm. The metrics chosen were: accuracy, precision, sensitivity and F1-score. These metrics are presented in the equations below:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

$$F1\ score = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} \tag{6}$$

$$TP : True\ Positives;\ \ TN : True\ Negatives$$
$$FP : False\ Positives;\ \ FN : False\ Negatives$$

Accuracy calculates the proportion of flows classified correctly concerning the total number of flows examined. Precision calculates the ratio of positive flows correctly classified among all flows classified as positive. High accuracy means that positively rated flows are less likely to be negative. Sensitivity calculates the proportion of all positive flows correctly classified among the actual positive flows. A high sensitivity means that most of the real positive flows have been classified correctly. The F1-score is the harmonic mean of precision and sensitivity, being a metric that takes into account false negatives and positives.

### B. Results and Analysis

The model convergence and training time plus the processing speed must be considered in the context of real-time analysis. To verify the impact of the data structure used during model training, we compared the model training time of Dataframe-based TeMIA-NT with the RDD-based CATRACA, and IDS previously proposed by our research group. Figure 2 shows that the DataFrame data structure several performance optimizations have a significant impact on the latency. Training the model with DataFrame is ten times faster than the same operation made with RDD.

We split the dataset into 70% for the training set and the other 30% for the test set to perform the classification score and obtain the models in the offline processing mode. Also, we used K-fold cross-validation, with $k = 10$, to guarantee the model's generalizability. Finally, we use the grid search method to tune the hyperparameters in each algorithm. Figure 3 shows the results of each algorithm, with random forest, decision tree, and gradient-boosted tree offering the best performances.

The online mode of operation uses the model with the highest processing capacity and good accuracy. Table I shows that the decision tree algorithm presented the maximum flow volume rate of 50 GB/s. The random forest classification model has a lower performance due to the need to process multiple trees, and it is necessary to obtain the result for all trees to achieve the final classification result.

TABLE I: Models processing efficiency with the best results in terms of the number and volume of classified flows per second.

| | Flows/s | GB/s |
|---|---|---|
| Random Forest | 586.563,32 | 21,95 |
| Decision Tree | 1.330.732,59 | 49,80 |
| Gradient Boosted Tree | 1.206.962,94 | 45,17 |

As the decision tree model presents the best results both in accuracy and in classification capacity, this is the model used by default in the execution of the tool. However, other models can also be used according to the user's needs.

The last test observed the impact of parallelism on the tool's performance, observing how variations in the number of processing nodes affect the results. The processing time

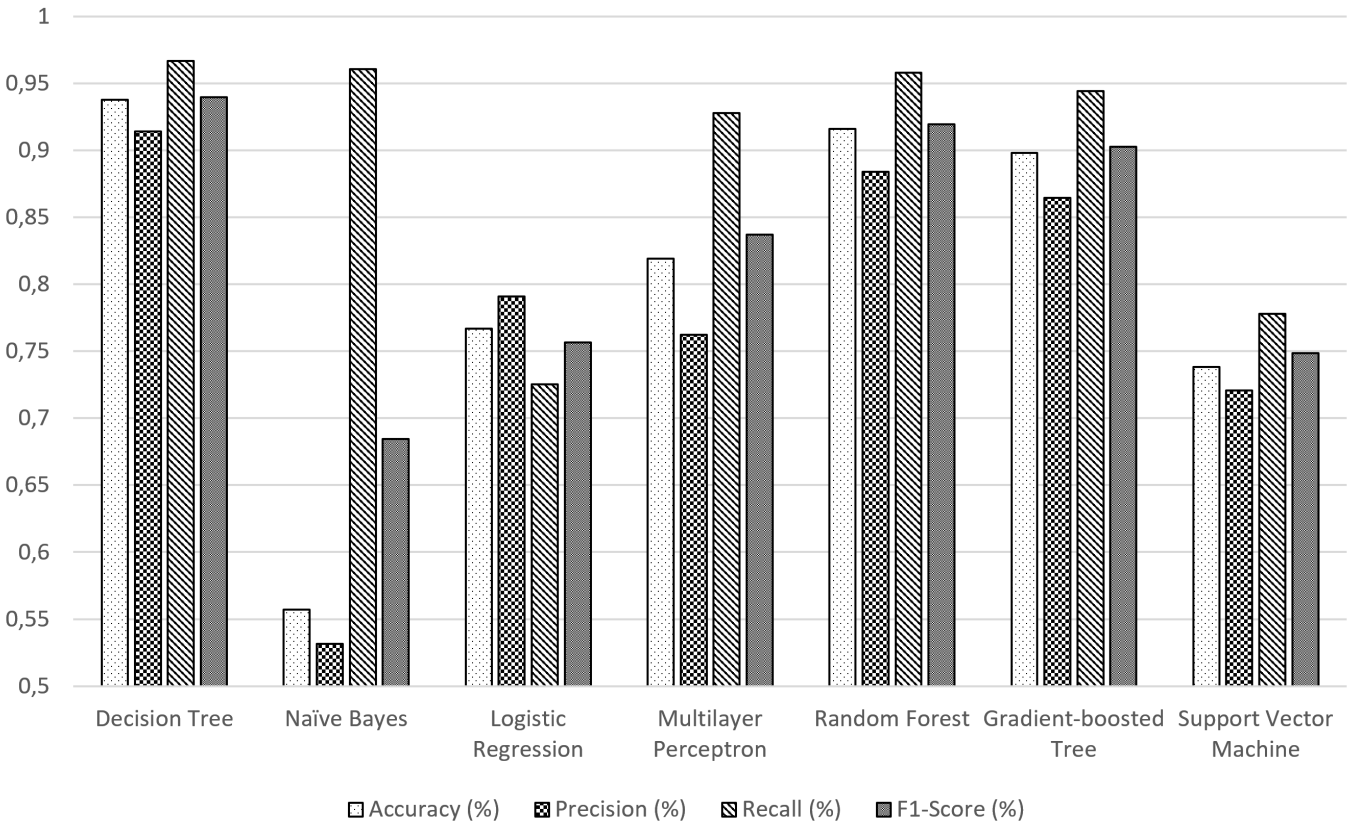


Fig. 3: Comparison of the evaluation metrics for the seven classifiers.

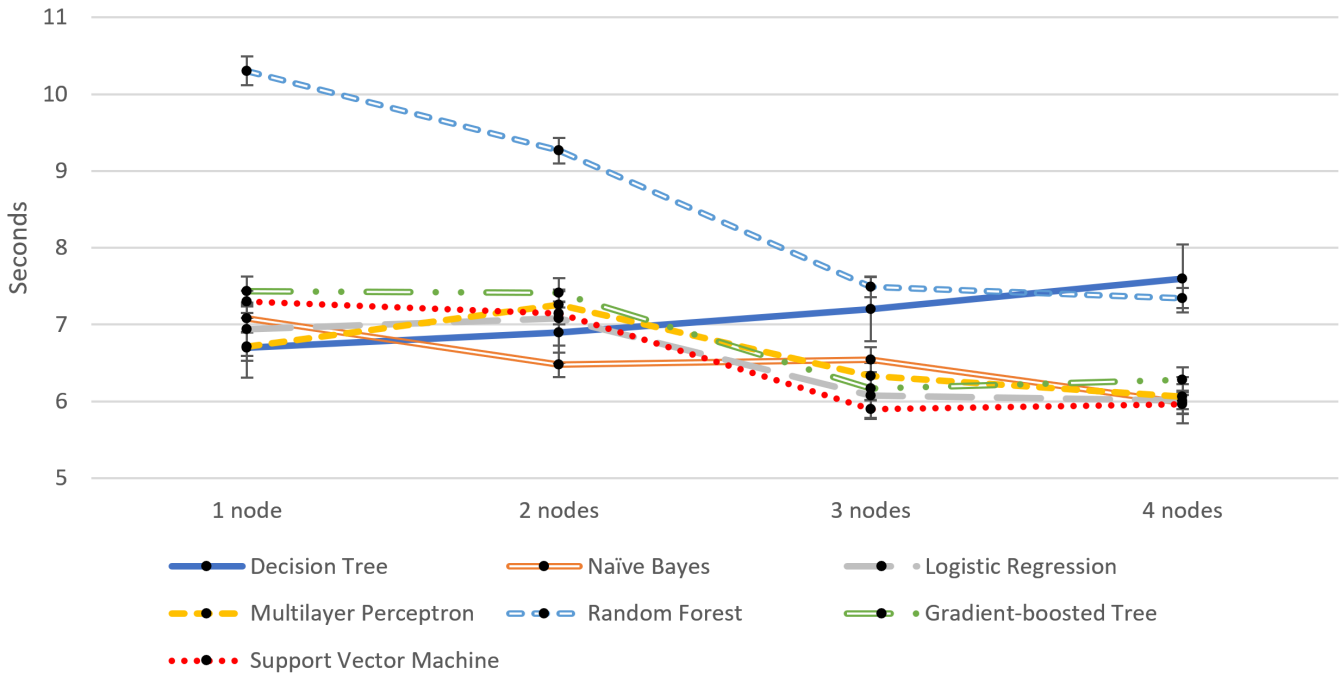


Fig. 4: Time necessary for classification based on number of processing nodes.

required to classify 10 million network flows was measured while varying the processing nodes number between 1 and 4 in the Spark environment; the results can be seen in Figure 4.

As can be seen from the Figure, increasing the number of processing nodes reduces the total time required to process flows for most algorithms. This impact is more significant in the case of the Random Forest, as this algorithm works by creating multiple models of trees that can be executed in parallel during the classification process. The results of the Decision Tree algorithm are negatively affected by the increase in the number of nodes.

## VII. Conclusion

This article presents the TeMIA-NT system[5], developed to monitor traffic using parallel flow processing. TeMIA-NT presents two modes of operation: online and offline. The online mode of operation allows the network manager to monitor and detect network security threats in real-time. The offline mode of operation allows the performance evaluation of multiple classification models obtained from different algorithms and datasets. TeMIA-NT also allows the selection from seven machine learning algorithms when obtaining classification models. The detection of threats in real-time with low latency is achieved thanks to the dataframe data structure and the continuous processing engine of the structured streaming library.

The obtained results from a dataset based on legitimate traffic demonstrate the high processing capacity in flows per second. The performance of each implemented machine learning algorithm is also observed, with the decision tree and random forest models presenting high values in metrics such as accuracy and f1-score. It was also shown how most algorithms scale with an increasing number of nodes on an Apache Spark cluster.

## References


[1] "Cybersecurity Market Report. Available at: *https://cybersecurityventures.com/*. Last access: 30 July 2020."

[2] E. Bertino and N. Islam, "Botnets and internet of things security," *Computer*, vol. 50, no. 2, pp. 76–79, 2017.

[3] A. Azmoodeh, A. Dehghantanha, and K.-K. R. Choo, "Big data and internet of things security and forensics: Challenges and opportunities," in *Handbook of Big Data and IoT Security*. Springer, 2019, pp. 1–4.

[4] Symantec, "Internet Security Threat Report. Available at: *https://docs.broadcom.com/doc/istr-24-2019-en*. Last access: 30 July 2020," 2019.

[5] R. A. A. Habeeb, F. Nasaruddin, A. Gani, I. A. T. Hashem, E. Ahmed, and M. Imran, "Real-time big data processing for anomaly detection: A survey," *International Journal of Information Management*, vol. 45, pp. 289–307, 2019.

[6] Verizon Enterprise, "Data breach investigations report. Available at: *https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf*. Last access: 30 July 2020," 2020.

[7] M. A. Lopez, D. M. F. Mattos, O. C. M. B. Duarte, and G. Pujolle, "Toward a monitoring and threat detection system based on stream processing as a virtual network function for big data," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 20, p. e5344, 2019.

[8] M. Pelloso, A. Vergutz, A. Santos, and M. Nogueira, "A self-adaptable system for DDoS attack prediction based on the metastability theory," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–6.


[5]The code, documentation, and license are available at: https://www.gta.ufrj.br/TeMIA-NT/.


[9] E. Viegas, A. Santin, A. Bessani, and N. Neves, "Bigflow: Real-time and reliable anomaly-based intrusion detection for high-speed networks," *Future Generation Computer Systems*, vol. 93, pp. 473–485, 2019.

[10] R. Campiolo, L. A. F. dos Santos, W. A. Monteverde, E. G. Suca, and D. M. Batista, "Uma arquitetura para detecção de ameaças cibernéticas baseada na análise de grandes volumes de dados," in *Anais do I Workshop de Segurança Cibernética em Dispositivos Conectados*. SBC, 2018.

[11] A. G. P. Lobato, M. A. Lopez, I. J. Sanz, A. A. Cardenas, O. C. M. Duarte, and G. Pujolle, "An adaptive real-time architecture for zero-day threat detection," in *2018 IEEE international conference on communications (ICC)*. IEEE, 2018, pp. 1–6.

[12] M. A. Lopez, D. M. F. Mattos, and O. C. M. B. Duarte, "An elastic intrusion detection system for software networks," *Annals of Telecommunications*, vol. 71, no. 11-12, pp. 595–605, 2016.

[13] M. A. Lopez, D. M. F. Mattos, O. C. M. B. Duarte, and G. Pujolle, "A fast unsupervised preprocessing method for network monitoring," *Annals of Telecommunications*, vol. 74, no. 3-4, pp. 139–155, 2019.

[14] Cisco Systems, "OpenSOC: The Open Security Operations Center. Available at: *https://opensoc.github.io/*. Last access: 30 July 2020," 2014.

[15] Apache Software Foundation, "Apache Metron. *https://metron.apache.org/*. Last access: 30 July 2020," 2017.

[16] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[17] T. Jirsik, M. Cermak, D. Tovarnak, and P. Celeda, "Toward Stream-Based IP Flow Analysis," *IEEE Communications Magazine*, vol. 55, no. 7, pp. 70–76, 2017.

[18] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache Spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[19] R. Xin and J. Rosen, "Project Tungsten: Bringing Apache Spark Closer to Bare Metal. *Available at: https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html*. Last access: 30 July 2020," 2015.

[20] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark SQL: Relational data processing in Spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1383–1394.

[21] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[22] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *2009 IEEE Symposium on CISDA*, July 2009, pp. 1–6.

[23] M. A. Lopez, R. S. Silva, I. D. Alvarenga, G. A. F. Rebello, I. J. Sanz, A. G. Lobato, D. M. F. Mattos, O. C. Duarte, and G. Pujolle, "Collecting and characterizing a real broadband access network traffic dataset," in *2017 1st Cyber Security in Networking Conference (CSNet)*, Oct 2017, pp. 1–8.