Machine Learning Intrusion Detection in Big Data Era: A Multi-Objective Approach for Longer Model Lifespans

Eduardo Viegas, Altair Olivo Santin, and Vilmar Abreu Jr

Abstract—**Despite highly accurate intrusion detection schemes based on machine learning (ML) reported in the literature, changes in network traffic behavior quickly yield low accuracy rates. An intrusion detection model update is not easily feasible due to the enormous amount of network traffic to be processed in near real-time for high-speed networks, in particular, under big data settings. In this paper, we propose a new scalable long-lasting intrusion detection architecture for the processing of network content and the building of a reliable ML-based intrusion detection model. Experiments performed through the analysis of five years of network traffic, about 20 TB of data, have shown that our approach extends the lifespan of our model by up to six weeks. That occurs because the average accuracy rate of our proposal lasted eight weeks after the training phase, and traditional ones reached only two weeks after the model building. Additionally, our proposal achieves up to 10 Gbps of detection throughput in a 20-core big data processing cluster.**

Index Terms—**Machine Learning, Intrusion Detection, Big Data, High-Speed Networks.**

I. Introduction

In recent years, network devices have been significantly increasing their bandwidth capability. Therefore, the average broadband speeds are expected to double within only five years, growing from 39 Mbps in 2017 to 75.4 Mbps in 2022 [1].

Cyberattacks have also significantly increased their capabilities, e.g., in 2017, a Distributed-Denial-of-Service (DDoS) attack peaked at 600 Gbps – a year after it reached over 1.7 Tbps, which is a 183 percent throughput increase. Thus, when DDoS attacks are occurring, they might represent up to 25 percent of a country's total current Internet traffic. Hence, current and future deployment of Intrusion Detection System (IDS) mechanisms must be able to perform at such high-speed network bandwidths.

Manuscript received June 1, 2020; revised October 19, 2020; accepted November 11, 2020. Date of publication November 17, 2020; date of current version March 17, 2021. This work was partially sponsored by Coordination for the Improvement of Higher Education Personnel (CAPES), grant 99999.008512/2014-0, and the Brazilian National Council for Scientific and Technological Development (CNPq), grant 430972/2018-0. Recommended for acceptance by Dr. Ka-Cheong Leung. *(Correspondence Author: Altair Olivo Santin).*

Eduardo Viegas and Altair Olivo Santin are with the Graduate Program in Computer Science, Pontifical Catholic University of Parana, Curitiba, Parana 80215-901, Brazil (e-mail: eduardo.viegas@ppgia.pucpr.br; santin@ppgia.pucpr.br).

Vilmar Abreu Jr is with the Polytechnic School, Pontifical Catholic University of Parana, Curitiba, Parana 80215-901, Brazil (e-mail: vilmar.abreu@pucpr.br).

Digital Object Identifier 10.1109/TNSE.2020.3038618

Traditionally, IDS techniques were built using signature-based approaches, meaning that the cyberattacks are detected by matching a signature (e.g., well-known streaming of bits or a sequence of events). Therefore, only known attacks can be detected through such an approach. In addition, as new attacks are discovered over time, new signatures must be built demanding human intervention, and detection throughput is further decreased as more signatures must be evaluated [2]. Therefore, the detection can only be performed after the cyberattack occurrence.

In a machine learning (ML)-based IDS, the intrusion is represented through a feature set that comprises the appropriate event behavior to enable the classification between benign and malicious events. Consequently, an ML model is expected to properly classify other events as long as they present the same behavior learned in a training phase.

When the network traffic behavior changes, a new intrusion model must be built as the current one has passed its lifespan. Building an intrusion model involves the storage of the new network data content, labeling of data content events, extraction and selection of intrusion features, the ML algorithm parameter optimization, detection model training, and testing of the model.

Current approaches in the literature, in general, assume that the intrusion model is updated regularly without taking into account the demanded human intervention and time spent to perform such a task [4]. For instance, a single 1 Gbps node can produce more than 10 TB of content in a single day. The impact of network traffic changes over time to proposed ML-based intrusion detection approaches is not evaluated in the literature. Consequently, the model update periodicity, and the accuracy degradation over time are not taken into account. In contrast, authors often assume periodic model updates while disregarding the challenges it poses to proposed ML-based techniques. As a result, the accuracy impact and the model lifespan of current literature approaches are unknown.

In the production (real-world) environment, the network traffic behavior changes daily, either due to new attack discoveries or the offering of new services [3]. Nonetheless, in the current big data era, network data arrives in an unstructured manner from a variety of sources.

Beyond having to deal with model updates, an intrusion detection architecture has to be deployed as a distributed system. Thus, it must provide a near real-time detection of network attacks from a variety of sources while also being able to update the ML model before its lifespan has passed.

Otherwise, its accuracy rates quickly decrease, rendering the system unreliable.

Current proposals for network traffic measurement and analysis in the big data context often rely on batch-based Hadoop clusters [5]. In general, current techniques store network packets as raw data (e.g., pcap format) in a distributed file system (e.g., the Hadoop distributed file system (HDFS) [6]) for later processing. However, although such approaches offer significant improvements in scalability [7], they lack applicability in a real-world environment because, in such settings, the network traffic must be analyzed at network speed for delay-free intrusion detection.

In this study, we propose a novel ML-based intrusion detection architecture suited for near real-time intrusion detection in high-speed networks. First, we propose a novel multi-objective feature selection technique. Our technique aims at providing a higher model lifespan with no significant impact on model accuracy. Consequently, the built model can survive for longer periods of time, while maintaining the accuracy rates obtained at training time. Thus, model updates are required less frequently, enabling model deployment in high-speed networks, which is advantageous since model updates are not always feasible.

Second, we proposed a novel architecture suited for near real-time intrusion detection. The proposed architecture is twofold. The first, stream-based, is responsible for performing the near real-time intrusion detection, executing the feature extraction and classification. The second, batch-based, is responsible for updating the underlying ML models through the proposed multi-objective feature selection technique.

In summary, the main contributions of this paper are: 1) A network traffic dataset spanning five years of real and labeled network traffic is provided. The dataset, the first of its kind, which comprises over 20 TB of data and $\sim$300 billion of network packets, is the first that enables the evaluation of the reliability over time of ML-based techniques; 2) We experimentally show how network traffic behavior changes affect the current ML-based approaches for intrusion detection along the time. The evaluation results, not experimentally performed in the literature, show that current approaches are unfeasible for near real-time detection of intrusion attempts in high-speed networks (big data content processing) due to their low model lifespan; 3) We propose a novel multi-objective feature selection technique aimed at increasing the ML model lifespan. The proposed approach is able to increase the period in which the ML model is reliable, in other words, it maintains the accuracy rates obtained in the test phase for longer periods of time; 4) We propose and evaluate an architecture comprising near real-time detection and a building technique feasible for intrusion detection in big data settings. Using ML models with longer lifespans, our technique can achieve higher detection throughputs while maintaining accuracy for longer periods of time.

The remainder of this paper is organized as follows. Section II presents the challenges that big data settings pose to ML-based intrusion detection techniques. Section III addresses related works. Section IV describes the novel intrusion detection dataset and evaluates traditional ML-based techniques regarding the model lifespan. Section V describes our proposed intrusion detection architecture and the novel multi-objective model-building technique. Section VI evaluates our proposal. Finally, Section VII concludes our work.

TABLE I
EXTRACTED FEATURE SET, FOR EACH FEATURE GROUPING IN TIME-WINDOW INTERVALS FROM RAW NETWORK DATA

Features Grouping	#	Network Features
Source IP Address, Source IP and Destination IP Addresses, Destination to Source IP Address, Source to Destination IP Address	1	Number of Packets
	2	Number of Bytes
	3	Average Packet Size
	4	Percentage of Packets (PSH Flag)
	5	Percentage of Packets (SYN and FIN Flags)
	6	Percentage of Packets (FIN Flag)
	7	Percentage of Packets (SYN Flag)
	8	Percentage of Packets (ACK Flag)
	9	Percentage of Packets (RST Flag)
	10	Percentage of Packets (ICMP Redirect Flag)
	11	Percentage of Packets (ICMP Time Exceeded Flag)
	12	Percentage of Packets (ICMP Unreachable Flag)
	13	Percentage of Packets (ICMP Other Types Flag)
	14	Average Packet Size
	15	Throughput in Bytes

II. INTRUSION DETECTION AND BIG DATA

A. *Extraction of Intrusion Behavior from Network Data*

Network-based intrusion detection systems (NIDSs) perform detection according to the intruder behavior gathered from the network data content. For instance, network data content can be made of packets or network logs, such as NetFlow records, among others. In general, a huge amount of network packets (big data settings) arrives in a disorderly manner. In other words, the network packets must be preprocessed before being handled by a NIDS engine.

During the preprocessing, fields of interest must be selected and parsed; only then can they be provided to a feature extraction module. The goal of the feature extraction module is to extract features; in fact, a vector of network behaviors is called a features vector. The networking event is aiming to describe behavior by a feature vector, which can then be evaluated and classified by an ML algorithm.

Usually, a single event (e.g., a network packet) does not enable a proper behavior characterization required for feature extraction because several packets typically comprise a network communication (message flow). In general, network-based attacks only differ from normal behavior when several packets are analyzed together. For instance, a single packet that occurred during a flood-based DDoS attack, if evaluated in isolation, might be either a normal client opening a connection or an attack.

A common practice for a feature's network extraction is to summarize a set of packets into a time-window to allow NIDS detection [8]. In this case, each network event is grouped according to a selected criterion, e.g., source IP address.

Table I shows a typical feature set extracted from the network data split into time-windows, building a network flow. In the table, four distinct feature groupings establish how the network packets are summarized in the time-window for the feature extraction task. For instance, the *Destination to Source IP Address* grouping scheme extracts the listed network features by summarizing the network packets that occurred in a time-window according to the destination to source IP addresses. As a result, for each grouping, 15 features are extracted; hence, the final feature vector comprises 60 features (15 for each feature grouping).

B. Machine Learning for Network-Based Intrusion Detection

In general, intrusion detection through ML-based techniques is performed employing pattern recognition approaches [9], which have a goal of classifying a given input into a set of classes. In NIDS, pattern recognition can be performed by classifying given network content as either normal or attack. The classifier development task is divided into a three-phase process: *training*, *validation*, and *test*.

In the training phase, a dataset containing a set of previously labeled events (e.g., network data content labeled as either normal or attack), is provided to an ML algorithm (classifier) to obtain a model. Additionally, a validation dataset can be used to evaluate the produced model and perform improvements (if needed) in its input, e.g., select the best subset of features and adjust the model parameters.

The evaluation of the resulting model through a test dataset is performed, and the accuracy rates, such as the rates of *true-positive*, *true-negative, false-positive,* and *false-negative* instances, are measured. The accuracy rate is measured as the ratio of instances correctly classified from the total number of evaluated instances.

A *true-positive* (TP) rate denotes the ratio of attack events correctly classified, whereas a *true-negative* (TN) rate denotes the ratio of normal events correctly classified. In contrast, a *false-positive* (FP) rate denotes the ratio of normal events misclassified as attacks, whereas a *false-negative* (FN) rate denotes the ratio of attack events misclassified as benign events.

In general, in the literature, authors often aim at decreasing their system FP rates, considering that a high FP rate renders the system unreliable to the administrator, which disregards further system alerts.

A popular approach to improve the obtained system accuracy is to perform a feature selection technique, for instance, through a wrapper-based approach. In this case, the best subset of features is selected according to a given feature selection goal, e.g., the test dataset's obtained accuracy.

Several techniques have been proposed for the feature selection task, ranging from random subset selection [16] to genetic search algorithms [17], which have yielded promising results. The genetic search feature selection approach leverages the notion of gene selection to find the best subset of features.

The idea behind genetic search algorithms is to use a ranking selection method to emphasize good points and a niche method to maintain stable subpopulations of good evolutionary features. Hence, during the feature selection task, the classifier is used to find the obtained accuracy according to a given subset of features. At the same time, the genetic search algorithm emphasizes good subsets for subsequent feature selection.

C. Network-Based Intrusion Detection in Big Data Settings

Big data scenarios are often characterized in 5 main aspects, namely 5Vs, which includes Volume, Velocity, Variety, Veracity, and Value [35]. For instance, consider a monitored high-speed network environment. In such a case, network data is generated at high velocity, which produces a vast amount of volume. The monitored network data may arrive in a variety of formats, which includes network packets, netflow records, or even application logs. Finally, the analysis of such data provides value, for instance, through the identification of an intrusion, if its veracity is assured.

In such settings, traditional computing architectures are unable to cope with the processing demands [36]. Hence, big data environments require novel and distributed processing architectures, such as those provided on Hadoop ecosystem [5].

For instance, to overcome network-based intrusion detection challenges in high-speed environments, several works have proposed distributed and highly scalable intrusion detection mechanisms [6]. In such a context, the data capturing mechanism must be able to read the network packets in an unstructured format from several sources (big data settings). The feature extraction mechanism must be able to structure the captured data and extract features in a distributed fashion. Hence, the usage of time-window intervals for the feature extraction task becomes challenging [10], because the data to be processed must be distributed between several nodes and analyzed according to its common properties, e.g., the source IP address (Table I, *Features Grouping*). Therefore, the data summarization for network content classification poses a significant challenge for feature extraction.

In NIDS, an ML intrusion model is only reliable for a small period of time, considering that the network traffic constantly changes even in high-speed networks. However, after the model lifespan has passed, the classifier continues to classify events, even with an obsolete model, producing an unreliable classification. Thus, it becomes necessary to detect the accuracy decrease to perform the model update task.

The problem is that the administrator does not know when accuracy decreases. As a result, model updates remain a great challenge in ML-based NIDS, since they are needed to perceive the accuracy decrease and repeat the classifier development process.

Surprisingly, in related works, the focus is on distributing the ML model into several nodes for throughput increase purposes [11]. However, the model update task remains to be addressed as a big challenge for NIDS deployment in big data settings.

III. Related Works

A. Intrusion Detection Through Machine Learning Techniques

Network traffic behavior changes over time are often neglected in the literature in which authors often assume that periodic model updates are performed, without considering the cost achieving such a task. For instance, Ambusaidi *et al.* [20] propose an ML-based NIDS based on support vector machine (SVM) and filter-based feature selection. In their work, the authors show that feature selection can improve detection accuracy.

Pajouh *et al.* [21] apply a feature reduction technique to decrease the feature set dimensionality, then naïve Bayes and k-nearest neighbor classifiers are applied in conjunction with the classification task. Wang *et al.* [22] apply a feature augmentation technique to increase the feature set dimensionality. In their evaluation, through an SVM, the authors can improve detection accuracy. A feature selection approach was also performed by Shitharth *et al.* [23]. In their work, the authors rely on a wrapper-based feature selection through a cuckoo search algorithm and a neural network-based classifier for intrusion detection. Similarly, the authors were able to improve detection accuracy.

In our prior work [24], a feature selection technique through a multi-objective genetic search algorithm was applied in the context of network-based intrusion detection, aiming for accuracy improvement and energy consumption decrease. The evaluation results have shown that a multi-objective feature selection technique aids in improving conflicting objectives.

Feature selection techniques are widely used, aiming for accuracy improvement. However, their applicability to improving model lifespan is yet to be known. In addition, related works generally do not consider the challenges that model updates introduce to their technique. As a consequence, the difficulties introduced by performing model updates render their proposed approach unable to cope with the evolving behavior of network traffic. To the best of our knowledge, we are the first to address the model lifespan at the model building stage.

B. Intrusion Detection for High-Speed Networks

Approaches for flow measurement and classification of massive network activities, in general, rely on pre-stored data. For instance, in Lee and Lee [25], a Hadoop-based network traffic monitoring and analysis system is proposed. The authors performed the network flow extraction by mapping the network packet files in HDFS. Their proposed approach was able to reach 14 Gbps in a 200-node cluster. However, their approach demanded the storage of the network packet files. In their work, the classification was performed using a connection threshold established through a set of Hive queries. Consequently, the queries must be periodically updated due to the network traffic changes over time.

Fortugne *et al.* [26] integrated several anomaly detectors in a big data processing architecture for network traffic classification. The authors applied a hash function to split the network data into groups. Each group held an anomaly detection model that classified network data according to the obtained anomaly score, using a previously established threshold. Consequently, their proposed technique did not address the classification scheme update challenge, and the model update task was neglected. In addition, their proposal detection throughput is unfeasible for high-speed network monitoring in a big data environment.

Some authors have proposed stream processing techniques for the measurement of massive network data. For instance, in Baer *et al.* [27], a data stream warehouse for network traffic classification is proposed. The authors relied on time windows for incremental and continuous execution of their data queries. In addition, their proposed technique combined an ML framework for the classification of their exported network features at each time window. However, their technique relied on a supervised dataset while failing to consider the scalability challenge of their ML algorithms and the challenge of improving the ML model lifespan. A similar approach to the proposal was taken by Apache Metron [28]. The tool relies on the Apache Storm [29] processing framework to perform feature extraction at each time window. However, the tool demanded the storage of the occurred network activities in the HBase for post-classification, consequently significantly decreasing their proposal throughput.

To the best of our knowledge, our work is the first that does not require the storage of network events, neither for feature extraction nor for classification. In addition, we deal with the evolving behavior of networks in the case of high-speed networks through a multi-objective feature selection technique that takes into account the model lifespan at the model building phase, thus, decreasing the ML model update periodicity.

IV. Lifespan of Traditional Machine Learning Detection Techniques

Although the need for model updates for NIDS is a known requirement, the lifespan of current detection models remains unknown. This section evaluates the accuracy of degradation and the model lifespan of traditional ML detection techniques.

A. Data Description

An important issue to be considered in intrusion modeling is to have a properly built training and testing dataset. A dataset used for such a purpose must be made of network data with real, valid, variable, publicly available, and correctly labeled events (network packets). However, in general, to provide such an enriched dataset, one must record real data, making data sharing unfeasible due to privacy concerns. Nonetheless, the evaluation of a model lifespan is even more difficult since data must be recorded for long periods, increasing the amount of data to be labeled and stored.

Our work has leveraged the measurement and analysis of the WIDE Internet (MAWI) network traffic archive [12]. More specifically, it has used the MAWI Samplepoint-F, from the MAWI archive, collected daily for a 15-min-long interval from a transit link between Japan and the USA, therefore made of real network traffic.

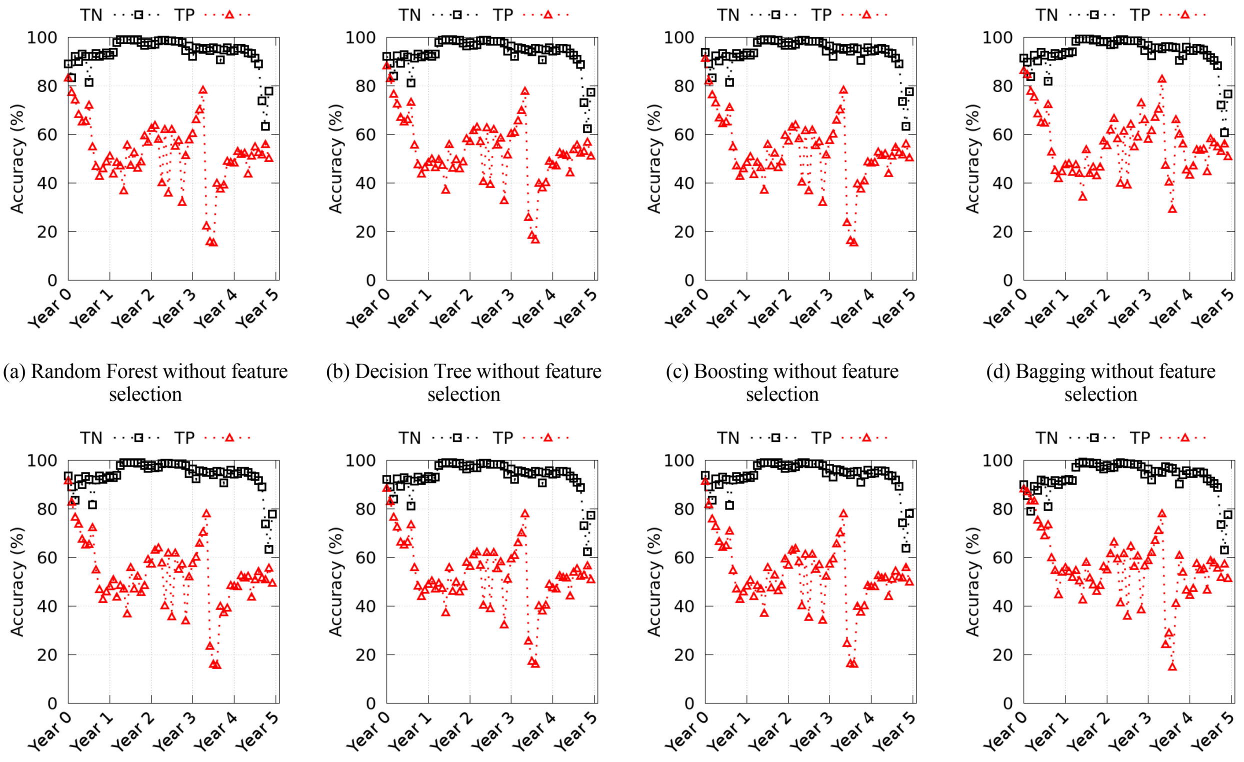
The used archive enables proper evaluation of model lifespan and accuracy degradation, using five years of network traffic ranging from 2012 to 2016. The built dataset comprises over 20 TB of data from $\sim$300 billion network packets. To automatically label the input records, i.e., tag events as either normal or attack, an unsupervised ML technique from MAWILab [13] was employed.

MAWILab employs several unsupervised machine learning algorithms to find anomalies in MAWI data, which do not demand individual event labels. The anomalies found are tagged as an attack, whereas the remaining data is assumed to be normal. The feature extraction algorithm (explained in Section V) groups events in a 15-second time-window interval, while extracting the feature set shown in Table I.

B. Accuracy Behavior Over Time

The first evaluation aims at assessing the ML model accuracy over time through the built dataset. Due to the imbalanced nature of the dataset (only $\sim$2% of instances are samples of attack), a random under-sampling without replacement was performed in the training data. Hence, the data distribution used for training purposes is equally distributed between the classes.

The total amount of data, $\sim$20 TB, is used for evaluation purposes. To properly evaluate the network traffic behavior and the



(a) Random Forest without feature selection (b) Decision Tree without feature selection (c) Boosting without feature selection (d) Bagging without feature selection

(e) Random Forest with feature selection (f) Decision Tree with feature selection (g) Boosting with feature selection (h) Bagging with feature selection

Fig. 1. Accuracy behavior of several ML algorithms with and without feature selection over a 5-year range.

change impact on the ML model, the classifier is trained through the first dataset month (January 2012) and then evaluated throughout the remaining five years without model updates.

Four widely used ML algorithms in big data settings were evaluated: random forest (RF), decision tree (DT), boosting and bagging. The selected set of ML algorithms were defined due to the vast amount of data in the used dataset, which renders more complex training and classification ML techniques unfeasible for real-time classification of network traffic.

The RF, boosting, and bagging algorithms were each built with 100 decision trees as their base learners. The DT classifier was implemented through the C4.5 algorithm, with a confidence factor of 0.25. The classifiers were built with and without feature selection.

The wrapper-based genetic search algorithm was used as a feature selection method, with a mutation probability of 3.3% and a crossover probability of 60%, executed with 100 generations with 100 populations each. The feature selection algorithm used the accuracy as a fitting parameter, as measured through the validation dataset, a common approach in the literature. The classifiers were built on top of Weka API version 3.8 [34] and were evaluated in a distributed manner through Apache Flink framework version 1.8.1 [31]. The classifiers were evaluated based on their TP and TN rates, being the ratios of attacks and normal events correctly classified, respectively.

It is important to note that as the system considers a two-class scenario, the FP and FN rates can be measured as the opposite of the TN and TP rates, respectively. Consequently, a low TN rate denotes a high FP rate, while a low TP rate denotes a high FN rate.

Figure 1 shows the accuracy behavior of the evaluated classifiers with and without feature selection. It is notable that when model updates are not performed, the model accuracy decreases for all the evaluated classifiers, with or without feature selection being made.

Surprisingly, significant decreases can be found within a few months after training. For instance, the RF classifier TP rate decreased by 11% in the first month following the training phase. Be that as it may, the accuracy continues to decrease as time passes, reaching only 19% for the RF, in the worst TP rate in November 2015, i.e., a 61% decrease from the rate obtained at the test phase, in January 2012.

The TN rate does not significantly decrease over time; in contrast, it even improves in May 2013 for the majority of the evaluated classifiers. Such accuracy behavior variation, a significant decrease in the TP rate, and a slight decrease or improvement in the TN rate shows that the built model is no longer reliable. The accuracy rates obtained at the test phase (January 2012) can no longer rely on only months after a training period.

It is worth noting that feature selection does not positively impact ML model accuracy over time (Figure 1-e to 1-h) compared to its counterpart using all features (no feature selection). We further investigate the accuracy tradeoff in Figure 2, which shows the RF accuracy over time with and without the traditional feature selection being made. It is possible to note that traditional feature selection has little or no impact on the final ML model lifespan, which, in general, improves the TP rate by only 0.1% over the evaluated five years of our built dataset.

Finally, we also investigate the relation between model lifespan and average accuracy to better understand the impact of

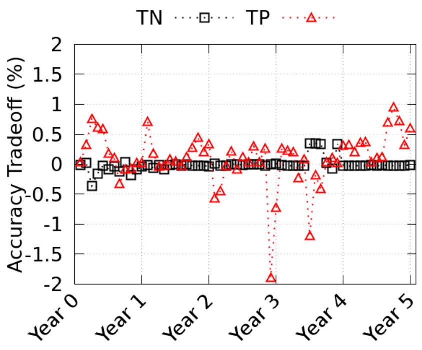


Fig. 2. RF accuracy tradeoff over time with and without feature selection.

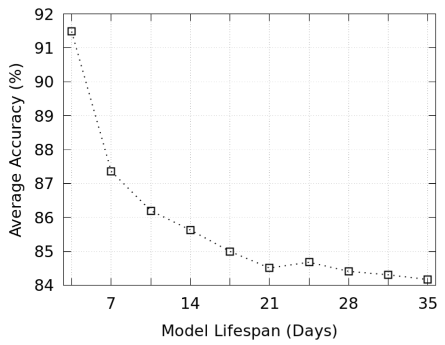


Fig. 3. Model lifespan (update periodicity) and average accuracy for an RF Classifier in 2016.

model update periodicity in the ML model. We evaluated the accuracy behavior according to the model update periodicity in 2016, as obtained through the RF classifier with all features.

The goal of the evaluation is to confirm if the accuracy decrease over time (shown in Figure 1) is related to the model's age. In other words, to confirm if the network traffic behavior is causing the accuracy decrease. Figure 3 shows the relation between the model update periodicity (model lifespan) and the average accuracy in 2016 for the RF classifier. A direct relation is notable between model update periodicity and model accuracy for up to 21 days of the model lifespan. That means it is possible to increase the system accuracy if model updates are performed regularly every 3 weeks or more. In addition, if a higher accuracy rate is desired, one must update the ML model more often. For instance, a daily updated model can reach ∼91% accuracy, almost a 5% accuracy increase when compared to a weekly updated model.

C. Discussion

Over the last few years, proposed ML-based intrusion detection schemes have disregarded the challenge of network traffic changes over time. This built dataset is a breakthrough toward the proper evaluation of ML-based intrusion detection schemes. To the best of our knowledge, it is the first dataset made of real network traffic, previously labeled, publicly available, and comprising many years of network traffic behavior.

The evaluation performed through our built dataset has shown that current ML-based intrusion detection schemes are unable to cope with the evolving behavior of network traffic, even when feature selection is made.

Current approaches significantly decrease their accuracy within weeks after the training period (Figure 1). Therefore, ML-based schemes must be updated regularly, making their applicability in real-world environments more challenging.

The periodic model has a direct impact on model accuracy over time (Figure 3). However, the model update task is a challenging process, particularly in big data settings. The data to be used must be stored and labeled, and only then can the training be performed. The training task is, in general, a highly computationally expensive process, which can demand days or even weeks of analysis of TB-scale data.

To enable the reliable usage of proposed intrusion detection schemes, the building of ML models with longer lifespans becomes imperative. Therefore, ML-based techniques must be able to withstand long periods without model updates. In other words, techniques must be able to remain reliable, even after a long period since the last model retraining.

It is important to note that such characteristics are often not given proper care. In general, proposed techniques are evaluated concerning their attained accuracy rate during the test phase, without taking into account the challenges that the changes in network traffic behavior over time pose to their proposed scheme.

V. Long-Lasting Intrusion Detection Architecture for Big Data Environments

In order to address the evolving behavior of network traffic over time while also performing the intrusion in near real-time in big data environments, we propose a *Long-Lasting Intrusion Detection Model*. The proposed model is implemented in a twofold manner.

First, we address the ML model lifespan through a multi-objective feature selection technique. The proposal insight is to take into account the model lifespan at the feature selection stage, aiming to increase the model lifespan with no significant side-effect on accuracy.

The proposal leverages a multi-objective feature selection technique with two main goals: accuracy and model lifespan. The accuracy objective aims to improve the model accuracy in a test dataset, which is the common approach used in the literature. In contrast, the model lifespan objective is to decrease the model update periodicity, and it takes into account the impact of the selected subset of features on model lifespan. As a consequence, the proposed model building technique can increase the model lifespan while maintaining or even improving the system accuracy.

Second, we propose a distributed intrusion detection architecture comprising both *batch* and *stream* processing environments. The goal is to perform near real-time intrusion detection through *stream* processing techniques while also provide updated ML models through the *batch* processing environment. In other words, the *stream* processing enables our architecture to detect intrusion attempts in big data environments in near real-time, whereas the *batch* processing environment periodically provides an updated ML model for the *stream* processing side.

The main aspects addressed by the proposed long-lasting intrusion detection architecture for big data settings are shown in Figure 4. The next subsections describe each stage in detail.

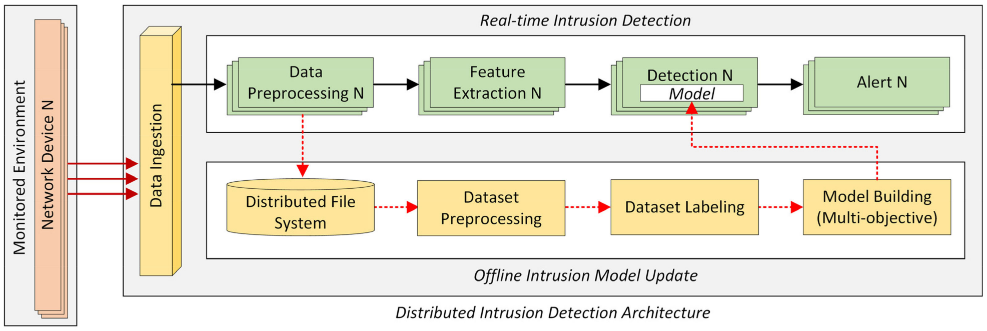


Fig. 4. Long-lasting intrusion detection architecture for big data settings.

A. *Architecture Data Flow*

The proposed architecture acquires network data from monitored network devices (Figure 4, *Network Device*), such as *routers*, *switches*, *servers*, and other network hardware or devices exposed to network attacks. The network data comprises network events of interest, such as *network packets*, *netflow records*, or *network logs*. Afterward, the *Data Ingestion* module (Figure 4) acts as a middleware between the proposed architecture and the network devices. Its goal is to handle the communication between the various monitored devices and ensure the proper handling of messages, providing a single interface between the unstructured data and the proposed architecture.

As the data is acquired, the *Data Preprocessing* module (Figure 4, *Data Preprocessing*) reads it in near real-time for both the *Real-time Intrusion Detection* and *Offline Intrusion Model Update* modules. In the *Real-time Intrusion Detection* module, the ingested data is forwarded for further processing in near real-time. The data is sequentially distributed and processed by the *Feature Extraction*, *Detection*, and *Alert* modules (Figure 4). The ingested data can also be used for model update purposes and is forwarded to the *Offline Intrusion Model Update* module. In this case, the collected data is stored in the *Distributed File System* until the model update periodicity has passed, triggering the model update procedure. In turn, the procedure is run by the *Data Preprocessing*, *Dataset Labeling*, and *Model Building* modules (Figure 4), which, when a new model is built, update the *Detection* module.

B. *Real-Time Intrusion Detection*

Despite the need for an ML model that properly classifies network events, the alarms must be generated on time, aiming to enable the proper handling of intrusion attempts while also helping reduce the damage when an attack is occurring. Therefore, ingested data must be classified as soon as possible by the *Real-time Intrusion Detection* module, which aims to detect network attacks in near real-time. The module is deployed as a stream processing big data architecture. The processing flow is performed as an acyclic graph flow. Each module is deployed in several processing worker nodes. Thus, the processing flow becomes scalable, providing a higher detection throughput.

During the processing flow, data arrives for preprocessing and is processed until it can be properly handled. The processing flow consists of the following tasks: *Data Preprocessing*, *Feature Extraction, Detection,* and *Alert* modules, each replicated in several workers' nodes (Figure 4).

The *Data Preprocessing* module receives unstructured input data from monitored devices and properly structures them for further processing. In other words, the module receives data in a variety of formats (e.g., network packets, logs, NetFlow records) and structures them into a single, known format.

As soon as data is structured, it is forwarded to both the *Feature Extraction* and *Offline Intrusion Model Update* modules. In the former, the real-time processing flow is continued for proper classification and reporting. In contrast, in the latter, data is stored until the model lifespan has passed and a model update is required.

For the classification, feature extraction must be performed. The structured data is forwarded to the *Feature Extraction* module grouped according to its properties. For instance, the data with the same source IP address is forwarded to the same feature extraction worker node for proper data value summarization.

Data summarization is performed according to the features grouping (Table I, *Features Grouping*). Nevertheless, data must be grouped in time-intervals. Therefore, the feature extraction exports the extracted feature set following a time-interval rule.

Finally, after the proper feature extraction, the classification can be performed in near real-time. Each *Detection* module holds a separate ML model. The goal is to increase detection throughput, considering that, in general, detection is orders of magnitude faster than training.

By replicating the model across multiple workers, it is possible to achieve a higher detection throughput than when using a single distributed model. After the proper detection of a network intrusion attempt, the alarm is forwarded to an *Alert* module. That module, in turn, sends the event to a reporting engine that provides the detailed detection report to the end-user. In this case, it is typically made of a graphical dashboard, data analytics engines, and other tools for better comprehension of the found intrusion attempt.

C. *Offline Intrusion Model Update*

In general, the ML training phase demands a great amount of time to properly learn behavior from the input event samples. In addition, in big data settings, the demanded time and required processing infrastructure for training significantly increase. The processing infrastructure for training involves ML algorithms, parameter optimization, feature selection, and model testing. Besides the execution of the IDS in near real-time, the ML model must be periodically updated offline. Consequently, the *Offline Intrusion Model Update* module is deployed as a batch processing architecture. The processing flow is executed sequentially and divided among several processing worker nodes.

Initially, the preprocessed data produced in near real-time by the stream processing side is stored in the *Distributed File System* to enable further processing. Afterward, at the end of each model lifespan, the model update process is performed.

The model lifespan must be established according to the administrator's discretion. For instance, one may use a lower model lifespan as an attempt to increase system accuracy. Others may use a higher model lifespan to decrease the processing costs. Therefore, the model lifespan must be established considering such tradeoff, as shown in Figure 3, through a traditional model building technique.

When the model update process takes place, the first performed task is in the *Dataset Preprocessing* module, which performs the feature set extraction and addresses incomplete data and outliers [29]. The preprocessed data is then handled by the *Dataset Labeling* module since, due to the vast amount of data, labeling cannot be performed by an expert; therefore, an automatic process must be performed. For instance, an unsupervised ML algorithm [30] can be used to identify anomalies.

As soon as data is preprocessed and correctly labeled, it is handled by the *Model Building* module that selects the appropriate feature set, customizes parameters, and trains and tests the model. Finally, at the end of the training phase, the new and updated model is forwarded to the *Real-Time Intrusion Detection* module, more specifically, the *Detection* module.

The model is then replicated in several worker nodes, improving the system throughput, i.e., maintaining the obtained system accuracy during the test phase in production deployment.

D. Building Models with Longer Lifespans

In the state-of-art, current model building techniques do not take into account the model lifespan at the training phase. That happens mainly because proposals in the literature do not have a properly built training dataset that spans for long periods (see Section III). In contrast, our architecture evaluates the model lifespan and takes it into account during the feature selection task. However, to provide a high detection accuracy, during feature selection, the model lifespan is coupled with detection accuracy in a multi-objective feature selection process.

The model building process is performed through a multi-objective feature selection technique with two objectives, namely *accuracy* and *model lifespan*. The objectives are computed through Eq. 1 and Eq. 2, in which N denotes the number of evaluated events, TP the number of true positives, and TN the number of true negatives.

$$accuracy = \frac{TP^{period\ i} + TN^{period\ i}}{N} \tag{1}$$

$$model\ lifespan = \frac{TP^{period\ i+1} + TN^{period\ i+1}}{N} \tag{2}$$

The *accuracy* goal is computed using Eq. 1. In this case, the goal is to improve the detection accuracy in a specific period (*period i*). For instance, improve detection accuracy in the period from the 1st day to the 15th day of January 2012. On the other hand, the *model lifespan* goal is computed through Eq. 2. It aims at improving detection accuracy in the period following the accuracy computation period (*period i + 1*). For instance, improve detection accuracy in the period between the 15th to 30th day of January 2012. In other words, the accuracy goal is to improve detection in a certain period, while the lifespan goal is to improve the accuracy in the period after the training time, aiming at the lifespan.

Our model building technique trains the model using the first half period of the training dataset and evaluates the model using the second half of that period. For instance, in a dataset comprising 30 days of network data, the first 15 days are used for training and *accuracy* computation (Eq. 1), whereas the remaining 15 days are used for *model lifespan* evaluation (Eq. 2).

The accuracy obtained in the last days is used as a model lifespan measure. However, to measure accuracy, the model is built and evaluated using five years. Consequently, the produced model contains information over the whole training period and has a higher expected model lifespan because we considered both model accuracy and expected lifespan during the feature selection task.

The built model is expected to last longer while also providing a higher detection accuracy. One must note that detection accuracy may present a conflicting objective to model lifespan. For instance, one may increase model accuracy by overfitting it for a specific period, and as a consequence, decreasing the model accuracy after such period, i.e., model lifespan. Therefore, the selected accuracy and model lifespan tradeoff must be selected according to the administrator's needs.

E. Discussion

Current approaches for network-based IDSs are not able to cope with the network traffic behavior changes over time (Section III) and high-speed networks. In fact, in general, the model lifespan is not even evaluated. As a consequence, when ML-based techniques are deployed in production environments, the ML model quickly becomes outdated, and a new model must be built, which demands expert intervention and wastes time.

The proposal takes into account both model accuracy and model lifespan. It achieves its goal by considering a specific period for evaluating the current model accuracy and the following period for evaluating the model lifespan. As a result, by considering the model lifespan during the model building stage, we can improve the expected model lifetime, decreasing the needed periodicity for model updates.

To overcome the challenge of ML-based IDSs in high-speed networks, we propose a twofold architecture, composed of *real-time intrusion detection* and *offline intrusion model updates*. The former handles IDSs in near real-time through a distributed stream processing architecture. The latter handles the periodic model updates, employing the aforementioned multi-objective feature selection approach to provide ML models with longer lifespans. Therefore, in summary, the proposed architecture performs intrusion detection in near real-time while also handling model updates aiming for a higher model lifespan.

VI. Prototype

A proposal prototype was implemented and deployed in a distributed environment, as shown in Figure 5. The prototype

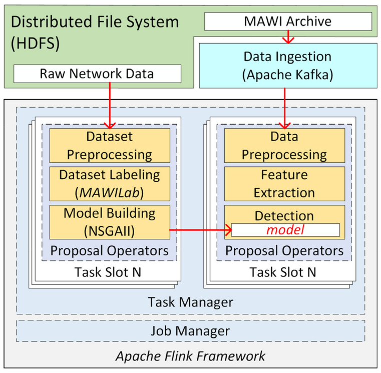


Fig. 5. Proposal prototype architecture.

is implemented on top of Apache Flink [31] processing framework, version 1.8.1, due to its capability to operate in both batch and stream processing conditions.

On the stream processing side, the prototype takes as input network packet headers from the MAWI archive [12], those stored in HDFS [32] version 3.2.1. The network packet headers are continuously read and sent to an Apache Kafka [33] topic, version 2.3.1, which acts as the proposal data ingestion mechanism (Figure 3). Meanwhile, the *Data Preprocessing* module reads the Apache Kafka topic and further processes it, forwarding it to the *Feature Extraction*, and then the *Detection* modules.

The proposal time-window mechanism was implemented for the feature extraction task using the native Apache Flink windowing mechanism (default: 15 seconds). The feature grouping (Table I) was realized using the Apache Flink *KeySelector* interface. Finally, the *Detection* module executes the underlying machine learning algorithm through the Weka API [34].

On the batch processing side, a batch job is periodically invoked, according to a given model lifespan periodicity. In such a situation, the *Data Preprocessing* module reads the raw network data stored on the HDFS, performs the feature extraction, and forwards the feature vector to the *Dataset Labeling* module. That module, in turn, labels the feature vectors according to the *MA WILab* [26] label files, which are provided in an XML format. Finally, the labeled feature vectors are used to build a new model.

The proposed multi-objective feature selection is performed through the NSGA-II [15] genetic search algorithm that computes the desired objectives, according to Eq. 1 and Eq. 2, and outputs an updated model. The final built model is then written to a file and read by the *Detection* module, which is performed in near real-time. The proposal's parallelism operators were set according to the number of worker nodes used in our experimental evaluation.

VII. Evaluation

The evaluation of our proposal was performed in two steps. First, we evaluated the technique for multi-objective feature selection that aims for a higher model lifespan. Second, we evaluated the scalability.

The performed evaluations aim at answering four research questions: *i*) *What is the accuracy and model lifespan tradeoff? ii) How good is our long-lasting model building technique*

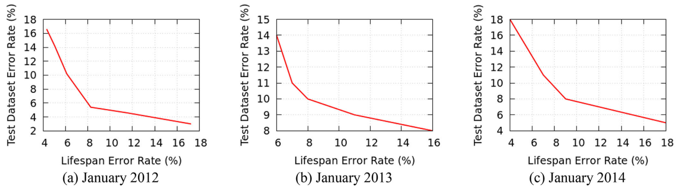


Fig. 6. Multi-objective feature selection tradeoff between Model Lifespan and model Accuracy for the three first years in the built dataset.

at improving model lifespan? iii) *How often must the model be updated to provide reliability? iv*) *Is the proposed architecture processing throughput scalable?*

A. Accuracy and Model Lifespan

The first evaluation comprises the accuracy and model lifespan improvement. To evaluate our proposed model, only the RF classifier was used, through the dataset introduced in Section III, as the other evaluated classifiers presented similar results. Similarly, the same set of parameters from Section III were used. Thus, the RF used 100 decision trees as its base-learner.

For building models with longer lifespans, the multi-objective feature selection algorithm NSGA-II was used [15]; at each model update, 100 generations with 100 populations, a mutation probability of 3.3%, and a crossover probability of 60% are used to find the best subset of features. A C++ program was implemented on top of the NSGA-II algorithm. The batch program is invoked by our prototype (detailed in Section VI) periodically. At each execution, the program selects a subset of features, as computed by the NSGA-II algorithm, builds an ML model through Weka API [34], and computes both accuracy (Eq. 1) and model lifespan (Eq. 2).

During model updates, the last past 30 days are used for model building. To evaluate the model accuracy (Eq. 1), only the first 15 days are used for training, while the remaining are used for the evaluation of the model lifespan (Eq. 2). For instance, if the model is updated in the last day of January 2012, the data that occurred between the 1st and 15th days are used for training, and data from the 16th to 30th are used for feature selection/testing.

For comparison purposes, the NSGA-II was also executed using a single-objective, in this case, accuracy, which is the traditional method used in the literature, using only the computation process from Eq. 1.

The evaluations were performed using an RF classifier with 100 decision trees as its base-learner. At each model update, a random undersampling without replacement procedure is performed due to the imbalanced nature of the dataset.

To answer the question (*i*), the first evaluation aimed at establishing the tradeoff between model lifespan and accuracy. Figure 6 shows the multi-objective tradeoff, for monthly model updates, between both feature selection objectives taking into account three periods (January 2012, January 2013, and January 2014); similar results were found at each model update.

It is possible to note a clear tradeoff between accuracy and model lifespan. For instance, in January 2012, one is able to increase model lifespan by 10% with a 10% accuracy tradeoff (Figure 5-a, Lifespan Error Rate versus Test Dataset Error

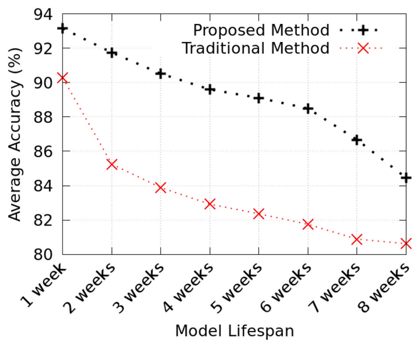


Fig. 7. Model lifespan and accuracy throughout time.

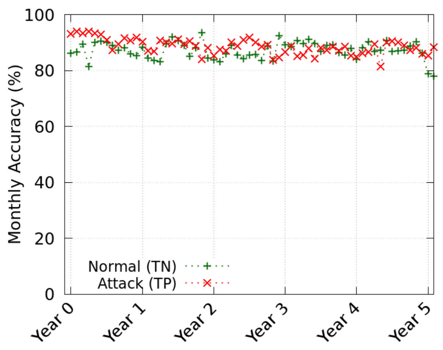


Fig. 8. RF accuracy throughout 5 years with monthly updates and multi-objective feature selection.

Rate). Therefore, to increase the model lifespan, although not significant, an increase in error rate must be tolerated.

When used in production, one must evaluate whether a higher frequency of model updates is needed or if the accuracy may be decreased but with a lower model update frequency.

To answer the question (*ii*), we evaluate the relation between model update frequency and accuracy over five years. To achieve such a goal at each model update, we select the average operation point when both objectives are close to zero, obtained from the NSGA-II, and build a new RF model.

The comparison between our proposed technique and the traditional feature selection approach is shown in Figure 7. It is notable that our multi-objective feature selection technique significantly improves the model lifespan and accuracy tradeoff when compared to traditional approaches (single objective, using only accuracy as the objective). For example, our model building technique was able to provide 90% accuracy with a model lifespan of three weeks, whereas the single selection approach only maintained the same accuracy with a model lifespan of one week, illustrating a model lifespan improvement of two weeks.

Our approach was able to build models with improved lifespans when compared to a traditional building technique. For instance, considering a two-week model lifespan, our technique experiences only a 1% accuracy decrease, while the accuracy of the traditional approach drops by 5%. Thus, the accuracy drops when model updates are performed less frequently is not as significant as in the traditional approach. That behavior can be observed for up to six weeks of model lifespan, which only degrades the system accuracy by 4% when compared to a weekly updated model.

It is notable that current approaches, which only consider accuracy at the test period, significantly decrease the model lifespan impacting the system accuracy. In contrast, when considering the model lifespan during feature selection, one can increase the model lifespan for up to six weeks (Figure 7, proposed technique model lifespan of eight weeks versus traditional technique model lifespan of two weeks).

We have also evaluated the obtained accuracy through the proposed multi-objective technique throughout the five years. Figure 8 shows the accuracy of the proposed approach when monthly model updates are performed (Figure 6, model lifespan of four weeks). It is notable that our proposed approach can maintain its accuracy over time ($\sim$90%), a significant improvement from the traditional approach that experiences a significant decrease in accuracy only weeks after the training period.

To answer the question (*iii*), one must consider the desired accuracy rates, given the model accuracy decreases over time for network attack classification. Therefore, one must always take into account the '*age*' of the classification model when performing IDSs. In other words, higher accuracy over time demands higher model update frequency. For instance, to reach 90% accuracy, one must perform model updates every three weeks, whereas to reach 84%, the model can be updated once every eight weeks (Figure 7).

The improvement in the relation between model lifespan and accuracy provided by our multi-objective feature selection technique significantly improves the ML-based IDS deployment in the production environment. That is because the system administrator is now able to properly establish the model lifespan and improve it when model updates cannot be performed frequently.

B. Scalability

To answer question (*iv*), the architecture throughput was evaluated during deployment using our 11-node cluster. The classification throughput was evaluated according to the *Real-time Intrusion Detection* module, considering that the model update task is typically performed offline. The architecture was deployed using 11 nodes, each with a 2-core CPU, 8 GB of memory, and an Ubuntu 18.04 OS. Out of the 11 nodes, ten were used as worker nodes (Figure 5, *Task Manager*) while the remaining node was responsible for the infrastructure management, acting as the master node (Figure 5, *Job Manager*). For each evaluation, the architecture was executed for 30 minutes, while its throughput was measured according to the network packet ingestion rate (Figure 5, *Data Ingestion* to *Data Preprocessing*).

Figure 9 shows the proposal scalability when varying the number of used worker nodes to perform near real-time intrusion detection. It is notable that our proposed approach can increase its throughput according to the number of available worker nodes. For instance, when deployed in a 10-worker node cluster, our proposal can reach up to 10.75 Gbps detection throughput, as opposed to a 5.67 Gbps detection throughput when deployed in a 5-worker node cluster.

In summary, each added worker node increases our architecture detection throughput by 1.07 Gbps on average, showing that when deployed in a distributed processing environment, our approach can scale-up its detection throughput, thus being feasible for high-speed networks.

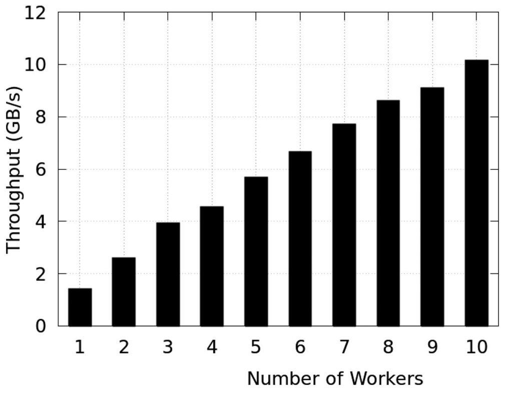


Fig. 9. Proposal scalability according to the number of used worker nodes.

VIII. Conclusion and Future Work

In recent years, the state-of-the-art of the model update task in ML-based IDSs has been neglected by the research community. In this work, we have tackled the problem of ML model's lifespan in big data environments. We have experimentally shown, through five years of real network data, that current and widely used techniques available in the literature significantly lose accuracy over time, within weeks following the training time. As a consequence, ML-based intrusion detection techniques presented models with short lifespans, unable to cope with the inherent changes of network traffic behavior over time. Our work proposed and evaluated a novel long-lasting intrusion detection architecture for big data environments. The proposed architecture, implemented in a twofold processing model: *batch* and *streaming*, was able to provide near real-time scalable intrusion detection while also solving the problem of short lifespans of intrusion detection models. Concerning proposal scalability, we showed that our architecture could be deployed in big data settings while demanding significantly fewer model updates over time. As future work, we will consider the improvement of the intrusion detection models' lifespan by combining hybrid detection architecture, coupling batch techniques, and stream learning techniques. The used dataset throughout the paper experiments, namely MAWIFlow, is publicly available for download in https://secplab.ppgia.pucpr.br/mawiflow.

References

[1] Cisco, "Cisco visual networking idsex: Forecast and trends, 2017–2022," in *Proc. VNI Glob. Fixed Mobile Internet Traffic Forecasts*, 2019.

[2] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. IEEE Symp. Secur. Priv.*, 2010, pp. 305–316.

[3] C. Gates and C. Taylor, "Challenging the anomaly detection paradigm: A provocative discussion," in *Proc. 2006 Work. New Secur. Paradig.*, 2007, pp. 21–29.

[4] S. Xu, Y. Qian, and R. Q. Hu, "Data-driven network intelligence for anomaly detection," *IEEE Netw*, vol. 33, no. 3, pp. 88–95, 2019.

[5] E. Viegas, A. Santin, A. Bessani, and N. Neves, "BigFlow: Real-time and reliable anomaly-based intrusion detection for high-speed networks," *Futur. Gener. Comput. Syst.*, vol. 93, pp. 473–485, 2019.

[6] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion detection and big heterogeneous data: A survey," *J. Big Data*, vol. 2, no. 1, pp. 1–41, 2015.

[7] R. A. Ariyaluran Habeeb, F. Nasaruddin, A. Gani, I. A. Targio Hashem, E. Ahmed, and M. Imran, "Real-time big data processing for anomaly detection: A survey," *Int. J. Inf. Manage.*, vol. 45, pp. 289–307, Feb. 2019.

[8] A. Baer *et al.*, "DBStream: A holistic approach to large-scale network traffic monitoring and analysis," *Comput. Netw.*, vol. 107, pp. 5–19, 2016.

[9] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Commun. Surv. Tut.*, vol. 18, no. 2, pp. 1153–1176, Secondquarter 2016.

[10] M. A. Faisal, Z. Aung, J. R. Williams, and A. Sanchez, "Data stream-based intrusion detection system for advanced metering infrastructure in smart grid: A feasibility study," *IEEE Syst. J.*, vol. 9, no. 1, pp. 31–44, Mar. 2015.

[11] X. Meng *et al.*, "MLlib: Machine learning in apache spark," *J. Mach. Learn. Res.*, vol. 17, pp. 1–7, 2015.

[12] "MAWI working group traffic archive." [Online]. Available: http://mawi.wide.ad.jp/mawi/samplepoint-F/. Last Access: June 1, 2020.

[13] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking," in *Proc. 6th Int. Conf.*, pp. 1–12, 2010.

[14] P. Carbone *et al.*, "Apache flink: Stream and batch processing in a single engine," *Bull. IEEE*, vol. 36, 2015.

[15] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[16] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Comput. Electr. Eng.*, vol. 40, no. 1, pp. 16–28, 2014.

[17] P. Hajela and C. Y. Lin, "Structural optimization genetic search strategies in multicriterion optimal design," *Struct. Optim.*, vol. 4, pp. 99–107, 1992.

[18] J. Peng, K. K. R. Choo, and H. Ashman, "User profiling in intrusion detection: A review," *J. Netw. Comput. Appl.*, vol. 72, pp. 14–27, 2016.

[19] C. F. Tsai, Y. F. Hsu, C. Y. Lin, and W. Y. Lin, "Intrusion detection by machine learning: A review," *Expert Syst. Appl*, vol. 36, no. 10, pp. 11994–12000, 2009.

[20] M. A. Ambusaidi, X. He, P. Nanda, and Z. Tan, "Building an intrusion detection system using a filter-based feature selection algorithm," *IEEE Trans. Comput*, vol. 65, no. 10, pp. 2986–2998, 1 Oct. 2016.

[21] H. H. Pajouh, R. Javidan, R. Khayami, A. Dehghantanha, and K. K. R. Choo, "A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in IOT backbone networks," *IEEE Trans. Emerg. Top. Comput.*, vol. 7, no. 2, pp. 314–323, 1 Apr. Jun. 2019.

[22] H. Wang, J. Gu, and S. Wang, "An effective intrusion detection framework based on SVM with feature augmentation," *Knowl.-Based Syst.*, vol. 136, pp. 130–139, 2017.

[23] S. Shitharth and D. P. Winston, "An enhanced optimization based algorithm for intrusion detection in SCADA network," *Comput. Secur.*, vol. 70, pp. 16–26, 2017.

[24] E. Viegas, A. O. Santin, A. Franca, R. Jasinski, V. A. Pedroni, and L. S. Oliveira, "Towards an energy-efficient anomaly-based intrusion detection engine for embedded systems," *IEEE Trans. Comput.*, vol. 66, no. 1, 1 Jan. 2017.

[25] Y. Y. Lee and Y. Y. Lee, "Toward scalable internet traffic measurement and analysis with hadoop," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 1, pp. 5–13, 2012.

[26] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking," in *Proc. 6th Int. Conf.*, pp. 1–12, 2010.

[27] A. Bar, A. Finamore, P. Casas, L. Golab, and M. Mellia, "Large-scale network traffic monitoring with DBStream, a system for rolling big data analysis," in *Proc. IEEE Int. Conf. Big Data*, pp. 165–170, 2014.

[28] A. Metron. [Online]. Available: http://metron.apache.org/.

[29] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 9, pp. 2250–2267, Sep. 2014.

[30] T. T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *Commun. Surv. Tut., IEEE*, vol. 10, no. 4, pp. 56–76, Fourth Quarter 2008.

[31] Apache Flink. [Online]. Available: https://flink.apache.org/

[32] HDFS Architecture Guide. Accessed: June 1, 2020. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[33] A. Kafka. Accessed: June 1, 2020 [Online]. Available: https://kafka.apache.org/

[34] WEKA. Accessed: June 1, 2020 [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/

[35] H. Hu, Y. Wen, T. S. Chua, and X. Li, "Toward scalable systems for big data analytics: A technology tutorial," *IEEE Access*, vol. 2, pp. 652–687, 2014.

[36] W. L. Chang and N. Grady. "NIST big data interoperability framework: Volume 1, definitions" in *Proc. NIST Special Publication (NIST-SP)*. Report n. 1500-1r2, pp. 1–53, 2019.