Toward Feasible Machine Learning Model Updates in Network-based Intrusion Detection

Pedro Horchulhack, Eduardo K. Viegas, Altair O. Santin
Graduate Program in Computer Science (Programa de Pós-Graduação em Informática - PPGIa)
Pontifical Catholic University of Parana (Pontíficia Universidade Católica do Paraná), Brazil
{pedro.horchulhack, eduardo.viegas, santin}@ppgia.pucpr.br

***Abstract*—Over the last years, several works have proposed highly accurate machine learning (ML) techniques for network-based intrusion detection systems (NIDS), that are hardly used in production environments. In practice, current intrusion detection schemes cannot easily handle network traffic's changing behavior over time, requiring frequent and complex model updates to be periodically performed. As a result, unfeasible amounts of labeled training data must be provided for model updates as time passes, making such proposals unfeasible for the real world. This paper proposes a new intrusion detection model based on stream learning with delayed model updates to make the model update task feasible with a twofold implementation. First, our model maintains the intrusion detection accuracy through a classification assessment approach, even with outdated underlying ML models. The classification with a reject option rationale also allows suppressing potential misclassifications caused by new network traffic behavior. Second, the rejected instances are stored for long periods and used for incremental model updates. As an insight, old rejected instances can be easily labeled through publicly available attack repositories without human assistance. Experiments conducted in a novel dataset containing a year of real network traffic with over 2.6 TB of data have shown that current techniques for intrusion detection cannot cope with the network traffic's evolving behavior, significantly degrading their accuracy over time if no model updates are performed. In contrast, the proposed model can maintain its classification accuracy for long periods without model updates, even improving the false-positive rates by up to 12% while rejecting only 8% of the instances. If periodic model updates are conducted, our proposal can improve the detection accuracy by up to 6% while rejecting only 2% of network events. In addition, the proposed model can perform model updates without human assistance, waiting up to 3 months for the proper event label to be provided without impact on accuracy, while demanding only 3.2% of the computational time and 2% of new instances to be labeled as time passes, making model updates in NIDS a feasible task.**

***Index Terms*—Intrusion Detection, Stream Learning, Reject Option**

I. Introduction

According to a Kaspersky [1] report, over 800 million network attacks targeting over 10% of all Internet users were identified in 2020 alone. In general, administrators use a network-based intrusion detection system (NIDS) to detect this growing number of network attacks through one of two approaches [2]: *misuse-based* approaches that search for well-known attack patterns in the input data, and as a result are only able to detect previously known threats, and *behavior-based* approaches that build a behavioral model according to the expected production environment behavior, thus identifying attacks according to deviations of the known and expected normal behavior [3]. Consequently, it is assumed that *behavior-based* techniques are able to detect new attacks as long as they behave significantly differently from benign events [4].

Over the last decades, due to the ever-increasing number of network attacks, several highly accurate *behavior-based* techniques have been proposed for intrusion detection, wherein pattern recognition through machine learning (ML) techniques are typically used [5]. An ML model is built through the evaluation of a training dataset that must be made of up to millions of labeled network events from both *normal* and *attack* behaviors. The obtained model can then be deployed in production environments for the classification of other events, a task that is achieved by finding similarities in the behavior of its input data to those modeled during the training phase [6]. As a result, if the network behavior of the production environment changes, a new ML model must be built, considering that the behavior present in the used training dataset does not reflect those currently experienced in production [7]. An outdated ML model may result in a higher error rate than those measured during the test phase. Thus, network operators may need to suppress signaled NIDS alerts as soon as the expected ratio of false alarms increases [4]. In practice, even the identification of a higher error rate is a challenging task, taking into account that the label of network events in production environments is unknown [3].

The behavior of network traffic is highly variable while also changing as time passes, a situation caused either due to the discovery of new attacks or due to the provision of new services [7]. Therefore, to maintain its classification accuracies and address the behavior changes of the production environment, ML-based NIDS must be updated regularly [8]. However, the model update is not easily achieved, considering that the new network traffic must be collected and correctly labeled [9], i.e., marked as either *normal* or *attack*. Nonetheless, the ML training task is also a computationally expensive process. Thus, model updates may demand days or even weeks of human assistance before the availability of an updated ML model [10]. As a result, systems deployed in production environments may remain unprotected against new kinds of attacks for several days due to the high cost related to model updates.

In practice, the labeling of network events is not readily feasible, considering it typically demands human assistance,

which is hardly available or demands a high financial cost [11]. Nonetheless, the manual labeling of all network events is not feasible, considering the vast amount of data that must be manually labeled. Due to the difficulties related to labeling of events, operators often make use of traditional *misuse-based* techniques for the labeling of network events for ML update purposes [3]. In such a case, the label of new attacks typically becomes publicly known after a significant period has passed since its initial exploitation, e.g., after weeks or even months after the attack occurred in production [12]. Thus, a feasible ML-based NIDS update can only be performed considering delayed model updates, e.g., after a month has passed, and traditional *misuse-based* can be used for the event labeling. Meanwhile, the outdated NIDS deployed in production must be able to maintain the expected system classification accuracies, despite the underlying ML model being outdated [3], [4].

Current ML model update approaches discard the outdated model and build a new model over the newly collected data [2]. As a result, the model update task requires a higher number of labeled network events to be provided, demanding additional storage and computational costs, considering that the previously modeled network behavior is discarded along with the outdated model. Despite being a known challenge, the model update task in ML-based NIDS remains overlooked in the literature, wherein authors assume that periodic model updates can be easily applied without considering the challenge it produces in their proposed schemes.

In general, the literature resort to stream learning techniques in scenarios wherein the environment behavior changes as time passes [13]. In contrast to traditional ML approaches, stream learning enables the incorporation of new instances into the currently deployed model in an incremental-based approach, further easing the model update challenge. However, despite its benefits on the model update task, it assumes the availability of the proper event label [14], making their application on networked environments unfeasible due to the vast amount of network traffic that must be labeled as time passes.

This paper proposes a novel intrusion detection scheme based on stream learning that can apply delayed model updates with old rejected instances without an impact on the accuracy of the classification, which is implemented in a threefold way. First, to ease the model update procedure, our proposal uses a stream learning classifier pool that enables incremental model updates to be conducted. As a result, model updates can be applied incrementally, with lower computational costs than traditional approaches. Second, to maintain the classification accuracy, even when outdated models are used, our proposal assesses the classification quality through a classification based on a reject option rationale. Thus, the classification quality of our proposed scheme is evaluated, and only highly confident classifications are accepted. With our proposal, although an updated ML model is not yet available, the confidence values of the classification can attest to the correctness of the classification, even with outdated models, maintaining the accuracy of the system classification. Third, model updates are incrementally applied on the stream learning classifier using old rejected instances. As the rationale of such a scheme, a correct event label will be publicly available after a while has passed, for example, after a month, further easing the model update challenge. In addition, model updates can be applied using only rejected instances, thus further decreasing the computational costs. As a result, our proposed model can maintain its classification accuracy over time, even with outdated models, and uses an easy-to-apply model update task through incremental model updates on previously rejected instances, without human assistance.

In summary, the main contributions of this paper are as follows:

- Widely used stream learning and batch learning classifiers are evaluated concerning their classification accuracies over time (Section IV). Experiments conducted on a dataset spanning a year of real network traffic show that current approaches are unable to achieve intrusion detection under evolving network traffic behaviors, demanding an infeasible periodicity of the model updates, as well as significant amounts of labeled training data.
- A new stream learning for intrusion detection model that conducts a classification with a rejection option and can maintain its accuracy for long periods without model updates is proposed (Section V). The proposed scheme without updates can provide accuracy rates similar to those of traditional monthly updating techniques.
- A feasible model update approach can achieve incremental model updates on intrusion detection schemes deployed during production. Model updates are conducted on a small subset of previously rejected instances, significantly decreasing the computational costs while not demanding human intervention for labeling, which can be applied autonomously through traditional *misuse-based* techniques. The proposed scheme can store rejected instances for long periods before using them for model update purposes without a significant impact on accuracy.

The remainder of this paper is organized as follows. Section II further describes the challenges of applying ML techniques for NIDS. Section III presents related studies on NIDS model updates. Section IV evaluates the traditional ML-based NIDS concerning their classification accuracies over time. Section V describes our proposed model. Section VI evaluates our proposed scheme, and Section VII provides some concluding remarks regarding this research .

II. Preliminaries

In this section, we describe the aspects that make network intrusion detection challenging to ML-based techniques.

A. Network-based Intrusion Detection

A typical NIDS can be described using four sequential modules [2]: *data acquisition*, *feature extraction*, *classification*, and *alert*. *Data acquisition* is responsible for data collection from the monitored environment, which is typically achieved through the collection of network packets from a network

TABLE I: Example of a feature set that can be extracted at network-level, considering each feature grouping in a 15s window interval.

Network Grouping	#	Collected Features
Hosts Communication, Source to Destination, Destination to Source	1	First Quartile Inter Arrival Time
	2	Median Inter Arrival Time
	3	Average Inter Arrival Time
	4	Third Quartile Inter Arrival Time
	5	Maximum Inter Arrival Time
	6	Variance Inter Arrival Time
	7	Minimum Inter Arrival Time
	8	Minimum Packet Length
	9	First Quartile of Packet Length
	10	Median Packet Length
	11	Average Packet Length
	12	Third Quartile of Packet Length
	13	Maximum Packet Length
	14	Variance of Packet Length
	15	Total Network Packets
	16	Total Network Packets With TCP ACK Flag Set
	17	Total Network Packet With Only TCP ACK Flag Set
	18	Total Network Packet With TCP SYN Flag Set
	19	Total Network Packet With TCP FIN Flag Set
	20	Total Network Packet With TCP PSH Flag Set
	21	Total Network Packet With TCP URG Flag Set
	22	Network Throughput

interface card (NIC) for further analysis. The collected data are then used as input by the *feature extraction* module, which extracts a set of behavioral features from the collected data. In a NIDS, network data are typically represented by a network flow, which summarizes the communication between the hosts and services within a given time window.

Table I shows an example of network-related features that can be used for a classification task [15]. In such a case, network packets exchanged between hosts, considering the *origin*, *destination*, and *hosts* as key values are summarized in a 15-s time window, and 66 features are extracted (22 features for each feature group). Therefore, network flows depict the communication behavior between entities on the network, measured by the exchanged network packets over time. The feature vector built is then classified as either *normal* or *attack* traffic using the *classification* module, applying an ML model, as an example. When a network flow is classified as an *attack*, the *Alert* module properly reports it.

Several techniques have been proposed for the classification of network flows, wherein the authors often resort to ML-based schemes, typically through pattern recognition (*batch-based*) approaches [2]. The proposed approaches rely on a three-phase process, namely *training*, *validation*, and *testing* [16]. In the *training* phase, an ML model is built through an evaluation of the network data available in the training dataset. In practice, the ML model training phase typically aims to find a behavioral model capable of optimizing the separation between its input classes, that is, *normal* and *attack* samples [17]. Thus, the training dataset must consist of a significant number of labeled network events from both *normal* and *attack* events. The *validation* phase aims toward model improvements, such as the fine-tuning of model parameters and the feature selection. Therefore, the ML model is built using the *training* dataset, and the model parameters are selected through a *validation* dataset, each composed of different network-related samples. Finally, the accuracy of the improved built model is estimated during the *testing* phase using the *test* dataset. The obtained model can then be deployed during production for the classification of new network events [3].

In recent years, several studies have applied *stream* learning techniques to scenarios in which the behavior changes over time [18]. In contrast to *batch* learning approaches, *stream* learning enables incremental model updates to be applied, wherein each new instance is used as an input for model updates. As a result, the computational cost of the model updates can be significantly decreased, considering that the outdated model can be leveraged during model updates. Stream learning has been extensively used in several fields, such as finance, telemetry, and industrial applications [19].

B. Challenges of Machine Learning for NIDS

Networked environments present many challenges compared to fields where ML techniques have been successfully applied [3], [4], [16]. The behavior of network traffic is highly variable; constructing a realistic training dataset becomes a challenging task.This is because the training dataset must provide a realistic number of network events, which can portray the expected production environment behavior to build a reliable ML model. As the behavior of network traffic can change drastically within a small interval, the construction of a realistic training dataset containing all expected network traffic variations is not easily achieved [16]. In addition, even if a realistic training dataset is built, it would fail as it would consider a static network traffic behavior [4]. The behavior of network traffic evolves as time passes, a situation that can be caused either by the occurrence of new attacks or owing to the provisioning of new services or new service content being requested [7], [16]. As a result, even a *"perfect"* ML model, built through a realistic training dataset, will increase its error rates over time-demanding periodic model updates to be applied to maintain its classification accuracy [7].

Model updates in ML-based NIDS are not easily feasible, considering that a new training dataset must be built and the newly collected network events properly labeled [3]. A procedure that can be executed manually by a human, which is often infeasible considering the vast amount of data that must be inspected or executed by *misuse-based* techniques [2]. Network events can be autonomously labeled in such a context as long as the collected attacks are publicly known. A related attack signature is made publicly available (e.g., reported in a *Common Vulnerabilities and Exposures* (CVE) database). However, public attack disclosures often occur after a significant period has passed [4], with some studies suggesting up to 300 days before their disclosure [20]. To apply *misuse-based* techniques, the collected network events must be stored for long periods before they can be adequately and autonomously labeled, leaving the systems unprotected against new attacks

because the outdated ML-based NIDS will not be promptly updated.

C. Desired Properties for Easiness on Model Updates

The desired properties of a reliable ML-based intrusion detection system have been discussed in several works in the literature [16]. In general, it is desired that proposed schemes can generalize the behavior of the events of a training dataset. As a result, proposed approaches should detect similar/new attacks and services while operating regardless of the current environment. However, regardless of the built model generalization capacity, the environment behavior will change as time passes, demanding model updates to be performed.

The model update task in an ML-based NIDS remains an overlooked problem in the literature, and authors often assume that it can be quickly conducted when needed [3], [4]. In practice, the model update task, particularly in networked environments, poses a significant challenge to network operators [16]. However, even identifying outdated ML models is not a common concern and is often overlooked by authors. An ML-based intrusion detection model must provide feasible model update procedures. Thus, it can withstand changes in network traffic behavior over time. Ideally, an ML-based NIDS should provide the following characteristics related to model updates:

- ***Small sample of new network events***. A model update task should be executed using only a few updated network events. In such a case, the number of instances used for model update tasks should be decreased, considering the challenges related to the collection of new events, and that previous knowledge of the network traffic behavior is available, for example, the data that was used for the initial model training task.
- ***Easy-to-use event labeling procedure***. The labeling task of newly collected network events must be conducted in an easy-to-use manner because human intervention is typically required for such a process, rendering a periodic execution infeasible, particularly when huge amounts of network events must be labeled. A labeling task should ideally make use of traditional *misuse-based* techniques that enable autonomous labeling to be conducted.
- ***Low computational resource requirement***. Model updates are typically executed offline. Thus, computational resources can be used as needed. However, because of the highly variable nature of network traffic behavior, model updates must be executed frequently, making the computational costs of the model update a possible burden on the network operator.
- ***Performed within a short time interval***. Model updates typically demand several days or even weeks to be achieved. In the meantime, an outdated model is deployed in a production environment, leaving systems unprotected against new types of attacks. Thus, the model update task should provide an updated model in as few time windows as possible.
- ***Long model lifespan***. Network traffic behavior varies significantly over time. Consequently, the ML model must deal with changes in the network traffic behavior as time passes without demanding frequent model updates to be applied. More specifically, model updates must not be required to be conducted often, considering the challenges related to such tasks.
- ***Maintain its accuracy rates even if outdated***. Updated ML models cannot be frequently provided, considering the time needed for model updates and how frequently a network operator can execute them. The ML model must ensure that it can maintain its accuracy rate for a while, even when facing new network traffic behavior that was not experienced during the model training task.

III. Related Works

Machine learning techniques have been extensively and successfully applied in several fields for classification purposes [3], [7]. In general, proposed schemes in the literature aims at higher classification accuracies, often considering unrealistic production environment settings. For instance, Kilincer *et al.* [6] compared the accuracy rates of several classical batch classifiers, such as Support Vector Machines (SVM), k-Nearest Neighbors (kNN), and Decision Trees (DT), on five distinct intrusion datasets. The classifiers reached high accuracy rates, regardless of the evaluation data used. Another batch learning approach was proposed by Sangkatsanee *et al.* [21], which conducts real-time intrusion detection using decision trees. Despite the low training time, their proposed model achieved high accuracy rates but did not consider periodic model updates.

Another popular approach used to increase accuracy relies on using a pool of classifiers in an ensemble-based approach. For instance, Gao *et al.* [8] showed that intrusion detection through an ensemble of classifiers and a majority voting scheme could increase the system accuracy. Their proposed model does not consider periodic model update needs. Fatemeh *et al.* [22] proposed a multiple classifier system based on AdaBoost and neural networks. The authors provided higher accuracy on a single dataset through feature selection and multiple classifiers, while the challenge of network traffic behavior change was not evaluated. In Jie Gu *et al.* [23], the authors relied on an ensemble technique through a feature augmentation approach for improved accuracy. Their proposed approach does not consider the non-stationary nature of network traffic or the challenges that periodic model updates pose to their technique. Otoum *et al.* [24] proposed an ensemble-based intrusion detection in a wireless field that also provides lower error rates. However, it leaves the system unprotected from changes in the behavior of the network traffic. Periodic model updates are considered by Alebachew *et al.* [25], wherein the training dataset is continuously updated with new instances. At each model update, the number of used training events is increased, demanding additional computational training time as time passes.

To address the computational burden of a model update in a dynamic environment, researchers often resort to stream learning techniques [26]. These approaches can incrementally update the underlying classification model with newly labeled instances. Despite being widely applied in several fields, stream-learning-based intrusion detection techniques are still at their inception. For instance, Adhikari *et al.* [27] proposed a stream learning scheme to address evolving intrusion attempts. Although their technique can ease model updates, their scheme assumes that the correct event label is always available. Another stream learning approach for intrusion detection was proposed by Martindale *et al.* [28], where the use of an ensemble of classifiers can increase the accuracy rates. Similarly, the authors assume that event labels can be provided as needed, which is unrealistic in networked production environments. In addition, Pu *et al.* [29] introduced a technique to identify anomalies by combining a One-Class Support Vector Machines (OCSVM) and Sub-Space Clustering (SSC), where each subspace is used as an input to an instance of OSCVM and finally in the detection of anomalies. Although their proposed model can achieve the autonomous labeling of events, it is computationally expensive to apply in real-time.

Owing to the difficulties related to the execution of model updates, in recent years, several works have been proposed in such context. For instance, federated learning performs the model training task in a collaborative and distributively manner, wherein training data can be kept on end devices for model training purposes [30]. Although such a technique can overcome privacy issues related to sending private data to a centralized entity, the main challenges related to model updates in intrusion detection remain unaddressed. In such a context, proposed schemes must keep their reliability for long periods, despite being outdated, while also having to deal with the main challenges related to model updates in NIDS (see Section II-C. In general, to ensure the reliability of classification, different authors often assess the classification confidence values of the classifiers [31]. For instance, Lin *et al.* [32] evaluated the classifier confidence values to reject potential misclassifications in biomedical image analysis. In addition, Marinho *et al.* [33] also relied on the same rejection strategy for map image classification purposes. Eduardo *et al.* [34] makes use of the classification confidence values to assess the classification reliability in intrusion detection. The proposed model can maintain system accuracy. However, the authors overlook the feasibility of model updates in their scheme as time passes, assuming the availability of the event label when needed. Classification with a reject option has yielded promising results in several fields where a misclassification incurs a high level of risk. However, it is infrequently used for intrusion detection purposes, despite the damage that a false negative may cause a computational system.

Due to the difficulties related to the lack of reliability of proposed schemes, often caused by the network traffic behavior changes over time, proposed approaches are rarely used in production. However, related works rarely consider a proper evaluation of their proposals, wherein traditional ML-based evaluation often takes place, discarding the properties of production environments. Roberto *et al.* [35] proposed a framework for reliable evaluation of intrusion detection approaches that includes a structured methodology to replicate all necessary steps ranging from the feature engineering process to the obtained performance metrics. As a result, the authors provided a well-defined evaluation approach that related works can replicate. The main focus of their proposed approach is to enable the replication of reported results in the literature without challenging the common assumptions used by related works. In a prior work [16], we have introduced the expected properties that a reliable ML-based intrusion detection model must provide while also presenting a dataset that enables such an evaluation. The desired properties include detecting similar/new attacks or services while maintaining the measurement accuracy when deployed on new environments. The impact caused by the changes in the network traffic behavior over time remains unaddressed in related works that focus on the reliability of intrusion detection schemes [16], [35].

IV. Detection of a Moving Target

This section investigates how changes in network traffic may affect ML-based intrusion detection approaches and how model updates can be used to address such a challenge.

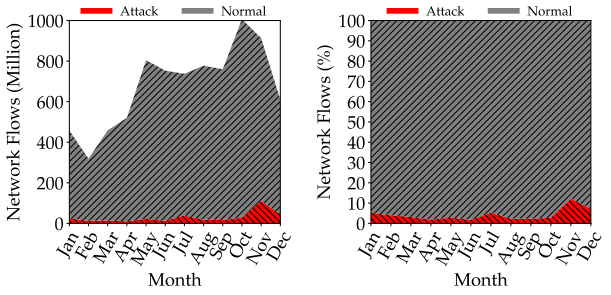
A. A Realistic Intrusion Dataset with Extended Time Interval

Intrusion detection techniques built over these datasets have not been evaluated when considering their long-term detection accuracies in the face of the underlying network traffic behavior changes. A realistic intrusion detection dataset [16] must provide ***real*** network traffic that can be observed in a production environment. The network data must contain ***valid*** and ***highly diverse*** network traffic with proper client/server communication. The collected events must be ***previously labeled*** as either *normal* or *attack* events through either expert assistance or autonomous approaches, such as through *misuse-based* tools or unsupervised ML algorithms. The network traffic must be collected for ***extended intervals***, thus enabling an evaluation of the impact of network traffic behavior changes over time. Finally, the dataset built should be ***publicly available*** for a proper benchmark of the results obtained.

Providing the characteristics above in an intrusion dataset is a challenging task that can be achieved in two ways [16]. First, the authors may collect real network traffic from the production environment. As a result, although the collected data are realistic, highly diverse, and valid, sharing is often impossible due to privacy concerns [36]. Nonetheless, the collection of real network traffic for extended intervals can pose a significant challenge because of the difficulties related to the labeling task [37]. Second, the authors may set up a controlled testbed that can ease the labeling task, as well as the collection of data for extended intervals [16]. However, the collected network traffic is often unrealistic, presenting a low diversity in client/server communication [38].

TABLE II: Used intrusion dataset statistics.

Property	*Value*
Average Daily Network Packets	110 Millions
Average Daily Network Flows	22 Millions
Average Daily Anomalous Flows	0.9 Millions
Average Daily Throughput	420 Mbps
Average Daily Dataset Size	7.1 GB
Total Network Packets	40.1 Billions
Total Network Flows	8.1 Billions
Total Dataset Size	2.6TB



(a) Number of network flows over time.

(b) Distribution of network flows over time.

Fig. 1: Dataset network flow distribution.

Our proposal is evaluated using the *MAWIFlow* [7] dataset. The data are made from network traffic that passes through the Samplepoint-F from the MAWI [39] archive, i.e., it is composed of real, valid, and highly diverse network traffic. The data are collected daily in 15-min intervals from a transit link between Japan and the USA. For our evaluation, was used the network traffic that occurred throughout the year of 2014. The built dataset comprises more than 2.6TB of data, compounding approximately 40 billion network packets. Because of the enormous amount of data, to enable the previous labeling of events, the data used are labeled using an unsupervised ML technique from MAWILab [11], which enables the autonomous labeling of the input records as either *normal* or *attack* events. To find anomalies in the MAWI archive, it uses several unsupervised ML algorithms, thus without human assistance for the event labeling task. The identified anomalies are labeled as an *attack*, whereas the remaining data are assumed to be *normal* events. For the feature extraction task, BigFlow [7] was used, which grouped events in 15-s intervals while extracting the 66 flow-based features from Moore's approach [15], as shown in Table I. A summary of the dataset statistics is shown in Table II.

B. Addressing Network Behavior Changes

The evaluations performed are aimed at answering the following research questions (RQ): (***RQ1***) *How do changes in network traffic behaviors impact traditional ML-based techniques*? (***RQ2***) *How do periodic model updates affect the accuracy of the evaluated schemes over time*? (***RQ3***) *What are the computational costs of periodic model updates*?

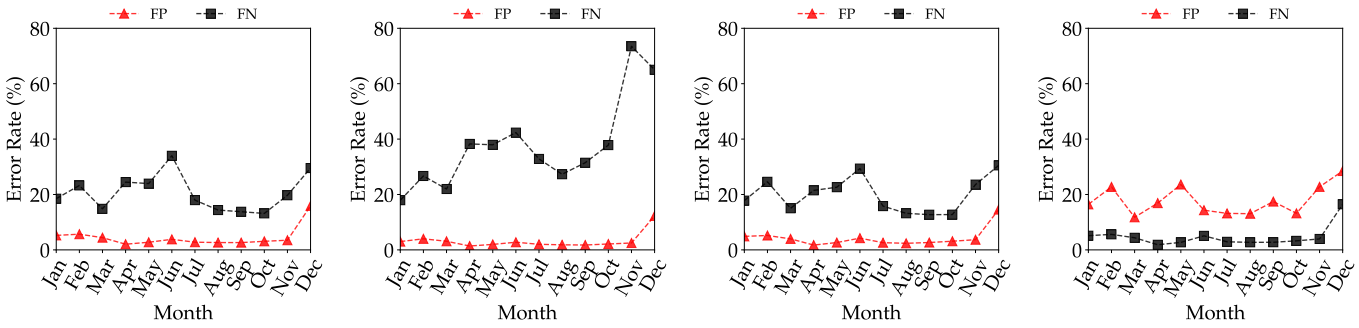
For evaluation purposes, two widely used ML-based techniques were considered: *batch* and *stream* learning approaches. *Batch* learning refers to traditional pattern recognition techniques in which the retraining of the model discards the outdated model at each model update. By contrast, *stream* learning refers to stream learning approaches that apply incremental model updates according to the outdated models.

Batch Learning. Four widely used batch learning algorithms for intrusion detection purposes were evaluated [2], namely, Random Forest (RF), Gradient Boosting (GBT), AdaBoosting (Ada), and Ensemble. The RF was evaluated using 100 decision trees as its base learner, each built using $gini$ as a node quality measure without a maximum tree depth value. The GBT was evaluated using 100 decision trees as base learners, with a learning rate of 0.1, and *friedman mse* as the split quality measure. Ada was evaluated using the boosting algorithm with 100 decision trees as the base learners and a learning rate of 1.0. Finally, the ensemble was implemented through a majority voting procedure for the three previously described classifiers. The batch learning classifiers were implemented on top of the scikit-learn API v. 0.24. ***Stream Learning.*** Similarly, four widely used stream learning classification algorithms were evaluated in our dataset: Hoeffding Tree (HT), Leveraging Bag (Bag), OzaBagging (Oza), and Ensemble. The HT was evaluated using information gain as the node split criterion, a grace period of 200, and naive Bayes adaptive as the leaf node prediction. The bagBag was evaluated with 3 HT as the base estimator and ADWIN as the leveraging algorithm with 0.002 as the delta parameter for the change detector. Similarly, the Oza was evaluated with 3 HT as base estimators. The ensemble was implemented through a majority voting procedure from the three single-evaluated classifiers HT, Bag, and Oza. The stream learning classifiers were implemented on top of scikit-multiflow API v. 0.5.3. Owing to the highly unbalanced nature of the dataset, because the majority of instances are *normal* (Figure 1), a random undersampling without a replacement technique is applied on each dataset day.

The classifiers were evaluated with respect to their false-negative rates (FN), false-positive rates (FP), and F1 scores. To achieve such a goal, the following classification performance metrics were used:

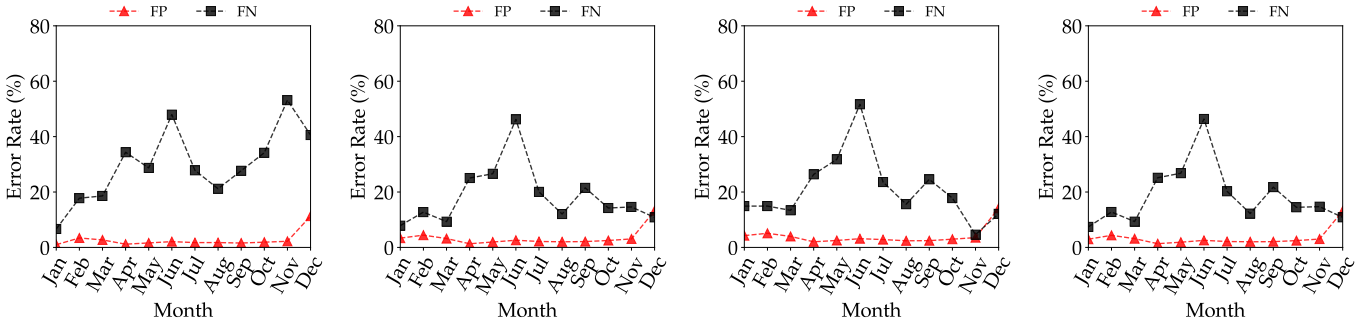
- *True-Positive* (TP): number of attack samples correctly classified as attack.
- *True-Negative* (TN): number of normal samples correctly classified as normal.
- *False-Positive* (FP): number of normal samples incorrectly classified as attack.
- *False-Negative* (FN): number of attack samples incorrectly classified as normal.

The F1 score was computed as the harmonic mean of precision and recall values while considering attack as positive samples and normal as negative samples [40], as shown in



(a) Hoeffding Tree (HT) (b) Leveraging Bag (Bag) (c) OzaBagging (Oza) (d) Ensemble

Fig. 2: Accuracy performance of the evaluated *stream* learning classifiers without performing periodic model updates.



(a) Random Forest (RF) (b) Gradient Boosting (GBT) (c) AdaBoosting (Ada) (d) Ensemble

Fig. 3: Accuracy performance of the evaluated *batch* learning classifiers without performing periodic model updates.

Eq. 3.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

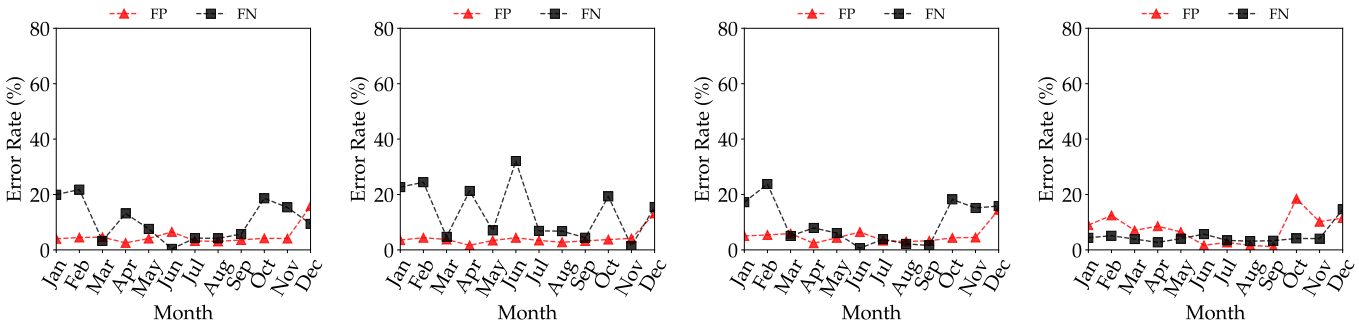
$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

The first experiment aims at answering $RQ1$ and evaluates the performance accuracy of both batch and stream learning classifiers when no model updates are applied. The evaluated classifiers were trained using the first 30 days of the *January* dataset data and evaluated through the remaining dataset period, without model updates being applied. The goal is to measure how the network traffic behavior changes over time, which affects the classification accuracy of traditional ML-based schemes.

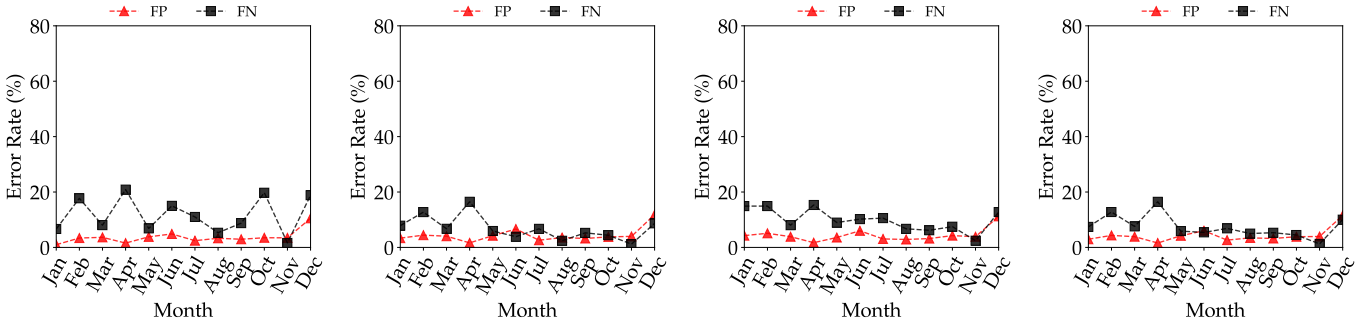
Figures 2 and 3 show the performance accuracy of the evaluated stream learning and batch learning classifiers, respectively. It is possible to note a significant accuracy impact as time passes and the model lifespan increases for all evaluated classifiers. For instance, the HT stream learning classifier (Fig. 2a) increases its FN rate by 4.7% in February, only a month after its training period, while presenting the worst performance in June, reaching a 34% FN rate, which is an increase of 15.4% when compared to the rates obtained in January. Similarly, batch learning classifiers are also affected by changes in network traffic behavior. In this case, considering the RF batch learning classifier (Fig. 3a), it increases its FN rate in February by 11%, while presenting the worst classification accuracy in November, reaching 53% of the FN rate. By contrast, the FP rates are not significantly affected as time passes; however, higher FN rates are experienced, demanding model updates to be nonetheless applied. Regardless of the classification scheme used, changes in network traffic significantly affect the classification accuracy if no model updates are conducted, even when an ensemble of classifiers are used (Figures 2d and 3d).

The second experiment aims to answer $RQ2$ and evaluate performance accuracy when applying periodic model updates. Monthly model updates are conducted on each evaluated classifier, using the data one month before the model update period. For instance, on February 1st, the underlying ML model is updated with data that occurred during the previous 30 days (January 1 through 31). Both batch and stream learning classifiers are trained from scratch through model updates. Hence, stream learning classifiers are not incrementally updated, as commonly assumed in related studies. In other words, the evaluation addresses changes in network traffic behavior through monthly model updates, a common assumption that has yet to be evaluated in the literature.



(a) Hoeffding Tree (HT) (b) Leveraging Bag (Leve) (c) OzaBagging (Oza) (d) Ensemble

Fig. 4: Accuracy performance of the evaluated *stream* learning classifiers with monthly model updates.



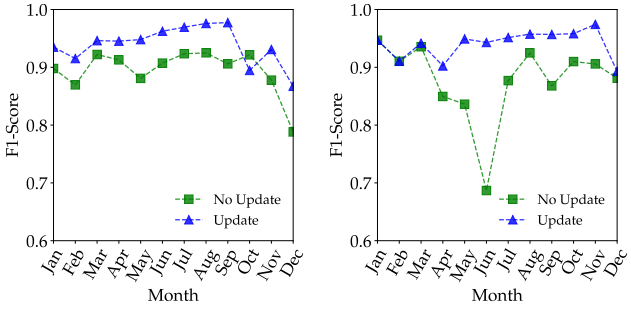
(a) Random Forest (RF) (b) Gradient Boosting (GBT) (c) AdaBoosting (Ada) (d) Ensemble

Fig. 5: Accuracy performance of the evaluated *batch* learning classifiers with monthly model updates.

Figures 4 and 5 show the performance accuracy throughout time when monthly model updates are applied on the stream and batch learning classifiers, respectively. In such a case, the majority of the evaluated classifiers were able to provide high accuracy rates throughout time, showing that periodic model updates can be used to address changes in network traffic behavior. For instance, the stream learning ensemble approach (Fig. 4d) presented an average FP rate of only 7.5%, whereas its no updated counterpart presented an average of 18% (Fig. 2d). Similarly, the worst accuracy rate does not significantly vary when periodic model updates are applied, showing that changes in network traffic behavior can be addressed through frequent model updates.

Figure 6 compares the F1 scores (Eq. 3) with and without periodic model updates, being performed of both batch and stream learning ensemble techniques. It is possible to note a significant improvement on F1 scores when model updates are being performed, increasing up to 0.09 and 0.24 for stream and batch learning techniques, respectively. As a result, reliable deployment of ML-based NIDS demands periodic model updates be performed to keep the system accuracies high as time passes.

We further investigate how challenging it is for the evaluated techniques to achieve periodic model updates. Figure 7 shows the cumulative number of instances demanded by the evaluated schemes during monthly model updates. It is possible to note



(a) Stream Learning (b) Batch Learning

Fig. 6: Comparison of F1-Score for batch and stream learning ensemble classifiers, with and without monthly model updates.

that, as time passes and further rounds of model updates are executed, the number of instances that should be labeled also increases. In practice, considering monthly updates with the last 30 days of data as the training dataset, all network events must be labeled over time. However, the task of labeling network events is challenging and infeasible to perform for all collected data, particularly when considering a high-speed network setting, which can produce huge amounts of events within a small window of time. As a result, despite periodic model updates enabling ML-based techniques to maintain their

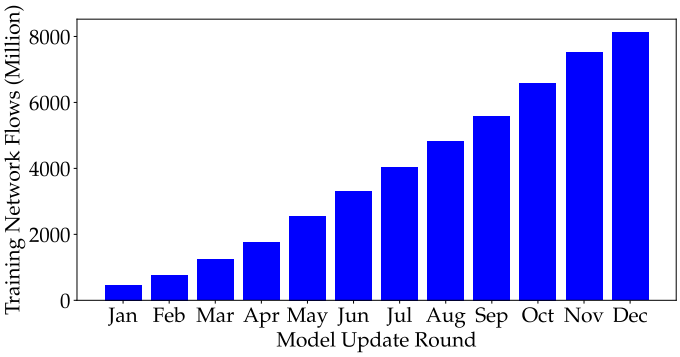
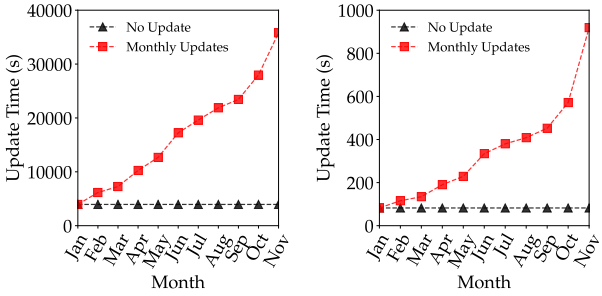


Fig. 7: Cumulative number of instances that should be labeled throughout a time when monthly model updates are applied. The large number of events that must be labeled during model updates poses a significant challenge to traditional approaches. Although *misuse-based* labeling approaches can be used for a subset of network events.



(a) Stream Learning (b) Batch Learning

Fig. 8: Cumulative computational costs of evaluated ensemble classification during model updates.

classification accuracies over time, the number of instances that should be labeled makes such a task infeasible in a real-world setting.

Finally, to answer $RQ3$, we investigate the computational costs of the model updates on traditional ML-based techniques. To achieve this goal, we compared the cumulative computational costs of the ensemble-based classifiers with model monthly model updates (Figs. 4d and 5d) versus their counterparts without model updates (Figs. 2d and 3d). The experiments were conducted on commodity hardware, and the computational costs were computed to the CPU usage time for all available cores, using multithreading ML API.

Figure 8 shows the cumulative computational costs of the ensemble-based classifiers. Periodic model updates demand an average additional computational processing time of 83 and $3,254$ s for the batch and stream learning classifiers, respectively. Recall that each evaluated ML technique is implemented on specific APIs and that the required processing time relies on the underlying implementation of the ML library. In addition, due to the outdated model's discarding, the batch learning classifiers will require more extended processing over time, and more network flows will be used in the model training task.

C. Discussion

In this section, experiments have shown that the natural changes in network traffic behavior over time affect the classification accuracies of ML-based approaches if no model updates are conducted (Figures 2 and 3). To address this evolving behavior of network traffic, we assume that periodic model updates will be applied. The experiments showed that periodic model updates can indeed be used to maintain the classification accuracies of intrusion detection schemes (Figures 4 and 5). However, the challenge related to the model update task required to maintain the reliability of the system makes periodic model updates unfeasible under production settings. More specifically, the number of instances that should be labeled during model updates (Figure 7) cannot be provided through the production settings. Nonetheless, although it can be executed offline, the increase in the computational costs owing to the frequent execution of the model update task (Figure 8) also further increases the difficulties related to the model updates. We further investigate how the changes affect the accuracy of the proposed intrusion detection schemes and how periodic model updates can address such behavior.

V. A Stream Learning Intrusion Detection Model With Delayed Model Updates

We propose an approach aiming to maintain the system accuracy as time passes while also easing the model update task through a two-step process, namely, *Stream Learning Intrusion Detection* and *Offline Model Update*. The *Stream Learning Intrusion Detection* aims to maintain the accuracy of the system as time passes, even if no model updates are applied. The proposal considers that model updates can be achieved with significant delays or even not applied at all, as occurs in a production environment. Consequently, the classification algorithm deployed must cope with new network traffic behavior even when outdated ML models are used. To avoid affecting the system accuracy caused by outdated models, we evaluate the classification confidence values during a classification based on a reject option rationale. As a result, only highly confident and most likely correct classifications are accepted by our model. Thus, our proposed scheme can maintain the accuracy of the system classification over time, even with outdated classifiers.

The goal of the *Offline Model Update* is to ease the model update task. Our scheme applies incremental model updates using the instances rejected by our classifiers deployed during production, thus decreasing the number of instances used during model updates. Nonetheless, to ease the labeling task, our scheme stores rejected instances for an extended period before using them for the model updates. As a result, the network operator can easily label them through traditional *misuse-based* intrusion detection techniques, considering that by the time the system is updated with such rejected instances, the proper event label will be publicly available, for example,

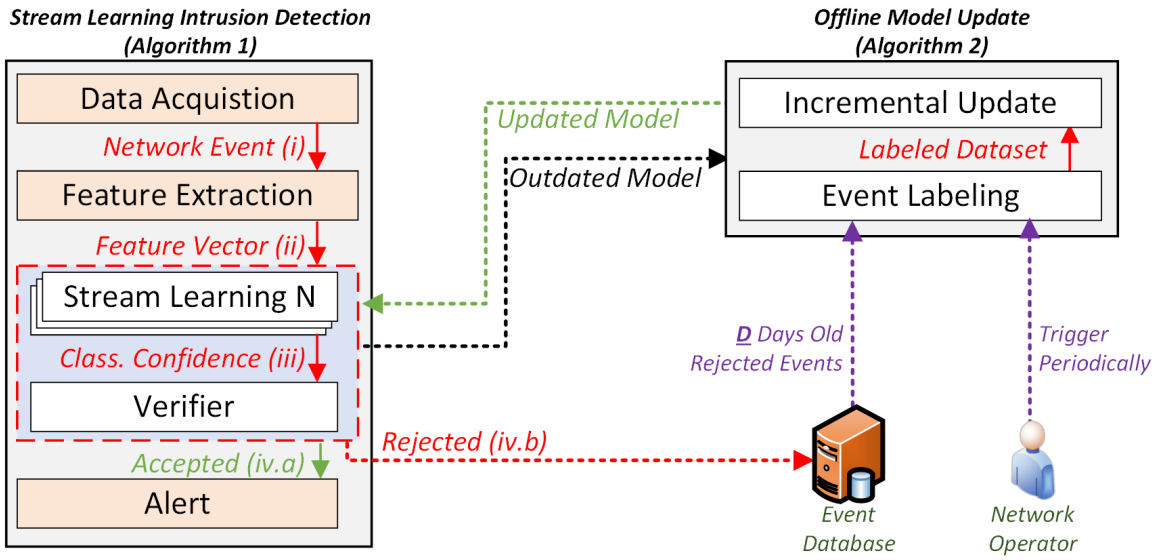


Fig. 9: Proposed intrusion detection based on streaming learning with delayed model updates. The classification procedure is executed continuously through a stream learning classifier pool. Model updates are executed offline and periodically with old rejected instances.

in a Common Vulnerability and Exposure (CVE) database. The proposal insight is that the model update task can be easily conducted using instances that could not be reliably classified by our underlying ML model during production, as evaluated through classification with a reject option approach.

Figure 9 shows an overview of our proposed model. The following subsections describe the modules implemented through our proposal.

A. Stream Learning Intrusion Detection

The behavior of network traffic changes regardless of the deployed ML-based NIDS must maintain its classification accuracies measured during the test phase under production usage even if the underlying ML model is outdated (see Section II-C). However, traditional ML-based NIDS applies a decision on all evaluated instances, even if they are unknown to the ML model, which may increase the error rate as time passes. As a result, current detection schemes often increase their error rate measured during the test phase over time, caused by changes in the underlying network traffic behavior.

To address such shortcomings the classification is conducted in two ways in our proposal. First, to ease the model updates, classification is conducted through a pool of stream learning classifiers (Figure 9, *Stream Learning N*). As a result, model updates can be applied incrementally, leveraging the current outdated model deployed during production, thus significantly decreasing the computational costs of the model update. Second, to maintain its classification accuracy for more extended periods, even if no periodic model updates are applied, events are classified with a reject option (Figure 9, *Verifier*). To achieve this goal, we evaluate the classification confidence values from each classifier and accept only those classifications that surpass a predefined classification acceptance threshold. Finally, the network event label is assigned through a majority voting rationale from all accepted classifications from every single classifier. On the contrary, if none of the deployed classifiers can accept the applied classification, the event is rejected, and its alert is suppressed. Classification confidence values are classifier agnostic, e.g., the Random Forest classifier outputs its confidence values according to the ratio of individual decision trees that assign a given label to the evaluated event. It is important to note that the rejection threshold must be defined according to the operator's discretion because a higher threshold will produce higher reliability but with different events being rejected. In contrast, a lower threshold will reject fewer instances but produce higher error rates during classification as time passes.

The overall classification procedure is shown in Figure 9 (*Stream Learning Intrusion Detection*). It starts with a to-be-classified network event collected by a *data acquisition* module (Fig. 9, *Event i*). The behavior of the collected data is then extracted using the *feature extraction* module, compounding a feature vector (Fig. 9, *Event ii*). The extracted vector is classified by a pool of stream learning classifiers (Figure 9, *Stream Learning N*), wherein each model outputs a related classification confidence value (Fig. 9, *Event iii*). The confidence values are evaluated using a *Verifier* module, which assesses the confidence values of each classifier according to their classification acceptance threshold. Each accepted classification is used to establish the final event label through a majority voting procedure. If all evaluated classifiers reject the evaluated event, its alert is suppressed, and the event is stored for later model updates (Fig. 9, *Event iv.b*). Otherwise, accepted events are forwarded to the *Alert* module, which properly reports them to the network operator (Fig. 9, *Event*

Algorithm 1 Proposal Network Flow Classification

Require:
Instance $inst = \{x_1, ..., x_N\}$
Classifiers $pool = \{c_1, ..., c_N\}$
Thresholds $threshold = \{(t_a^1, t_n^1), ..., (t_a^N, t_n^N)\}$
procedure CLASSIFICATION($inst$, $pool$, $threshold$)
  for each $classifier, t \in \{pool, threshold\}$ **do**
    $class, conf \leftarrow classify(classifier, inst)$
    if $class = normal$ **and** $conf \geq t_n$ **then**
      $vote(inst, normal)$
    else if $class = attack$ **and** $conf \geq t_a$ **then**
      $vote(inst, attack)$
    end if
  end for
  if $getNumberOfVotes(inst) = 0$ **then**
    $reject(inst)$
  else
    $alert(getMajorityVote(inst))$
  end if
end procedure

iv.a).

Algorithm 1 describes the classification procedure of our proposal. The algorithm receives as input an instance (*inst*) to be classified, composed of several flow-based features ($x_1, ..., x_N$), a classifier *pool* with several stream learning classifiers, and a corresponding classification acceptance *threshold* for both normal and attack classes for each classifier. Every single classifier conducts the classification, and the corresponding output confidence value (*conf*) is evaluated to accept or reject a decision performed by the single classifier. Accepted classifications are used by a majority voting process (*vote*). Finally, if none of the classifiers are able to accept the classification, the event is rejected (*reject*); otherwise, an alert is signaled (*alert*).

B. Offline Model Updates

Model updates are conducted offline, considering that the outdated ML model will still be deployed under the production environment. To decrease the number of events that must be labeled as time passes (see Section II-C), the model update task is achieved through only those events that were previously rejected by our proposal. As the rationale of such a scheme, model updates can be made only through instances that are not reliably classified by the outdated model deployed during production. This is because the underlying ML models will be improved according to only those instances it was unable to classify previously properly. As a result, our proposal can further decrease the number of instances labeled, stored, and used for the model updates. In addition, model updates are conducted only when a predefined time window has passed since the rejection of a given event (Figure 9, ***D*** *Days Old Rejected Events*) because, if older events are used in the model updates, they can be autonomously labeled using traditional *misuse-based* tools. Consequently, by the time the rejected event is used in the model update, the proper event label will be publicly available, for example, in publicly available cybersecurity attack databases, making the labeling task feasible in a production environment. Thus, model updates can be applied autonomously without human intervention.

Algorithm 2 Proposal Offline Model Update

Require:
Storage Interval $d = daysToStoreRejectedInstances()$
Dataset $dataset = getOldRejectedInstances(d)$
Classifiers $pool = \{c_1, ..., c_N\}$
Label Provider $l = getLabelProvider()$
procedure MODELUPDATE($dataset$, $pool$, l)
  for each $instance \in dataset$ **do**
    $label = provideLabel(l, instance)$
    for each $learner \in pool$ **do**
      $incrementalUpdate(learner, instance, label)$
    end for
  end for
end procedure

The model update procedure is shown in Figure 9. It is triggered periodically by a network operator, e.g., every month. In such a case, stored rejected events older than a predefined time window (e.g., 30 days after its rejection) are collected from an event database that stores rejected events from the production environment. The selected events are then labeled through an *Event Labeling* module, which may either be accomplished through expert assistance or make use of *misuse-based* techniques for the autonomous labeling of events. The labeled events are then used for incremental model updates of the pool of stream learning classifiers deployed within the production environment.

Algorithm 2 describes the model update task of our proposal. It receives as input a storage interval d that represents the number of days a rejected event should be stored before it can be used for model updates, a $dataset$ comprising rejected events older than d, a classifier $pool$, and a label provider l. The label provider is used for labeling a networking event properly. It may be made of expert assistance, *misuse-based* tools, or even *unsupervised* ML algorithms. The model update is achieved by requesting the event label for each instance in the dataset. With its corresponding event label, the network event is used for the incremental model update for each classifier used in the classification pool. Finally, the updated classifier pool is forwarded for the classification module to update the underlying ML algorithms (Figure 9, *Updated Model*).

As a result, the model update task is significantly eased. Fewer instances must be used for model updates, considering that only rejected ones are used for an update. Selected instances can be autonomously labeled through traditional *misuse-based* techniques, considering that they can be stored for an extended period and that their label will be publicly known by the time they are used for model update purposes. Nonetheless, the required computational processing during model updates is significantly decreased because our proposed

model leverages stream learning classifiers, enabling incremental model updates to be performed. This is also a result of the reduced number of instances applied for the model updates because we only use those previously rejected instances by our scheme.

C. Discussion

The number of required updated network events (see Section II-C) decreases as our proposed scheme is able to select which instances should be used for model update purposes through classification with a reject option approach. Given, they are stored for some time before being used for model update purposes, newly collected network events can be easily labeled as they are stored for a while before they are used for model updates. Computational costs are decreased considering that only a subset of instances is used for model updates, and outdated models are updated over time. Finally, considering that our model accepts only highly confident classifications, the accuracy rates and model lifespan are improved during production usage even with outdated models, considering that only highly confident classifications are accepted by our scheme. As a result, our proposed model can address the main challenges related to model updates on ML-based NIDS while also providing reliability in intrusion detection over time.

VI. Evaluation

Our evaluation aims at answering the following research questions: (***RQ4***) *Is the proposed classification evaluation approach able to improve the accuracy of the classification?* (***RQ5***) *How does our proposed model perform without periodic updates?* (***RQ6***) *How does our proposed model perform with periodic model updates?* (***RQ7***) *How does the delay on model update task affects the classification accuracy over time?* (***RQ8***) *What are the computational costs of our proposed model?*

The following subsections further describe our proposed model building procedure and the evaluations conducted

A. Model Building

The proposed scheme was implemented, making use of the same set of stream learning classifiers evaluated previously (see Section IV). Therefore, our proposed model relies on a pool of stream learning classifiers that includes the HT, Bag, and Oza, similarly as performed by the Ensemble approach (Figure 2d), also evaluated in Section IV. Similarly, the stream learning classifiers were implemented on top of scikit-multiflow API v. 0.5.3, and the same set of parameters were used. Due to the highly unbalanced nature of the dataset, as the majority of instances are *normal* (Figure 1), a random undersampling without replacement technique is applied on each dataset day as a dataset preprocessing procedure.

B. Stream Learning Intrusion Detection

Our first experiment aims at answering $RQ4$ and evaluates whether the proposed approach for a classification evaluation aids in improving the accuracy of the proposed scheme. The stream learning ensemble of our proposed model was trained using January data, and the tradeoff between the error and rejection rates on the test dataset of February was evaluated. The error-reject tradeoff is established through the class-related-threshold approach [41]. Therefore, each considered event class, *normal* and *attack*, holds a specific acceptance threshold for each single classifier (see Algorithm 1). The classification confidence values are classifier agnostic and were obtained through the *predict_proba* function in the scikit-multiflow API. The goal is to measure whether a classification assessment can be used to improve classification accuracy, even under an outdated ML model setting.

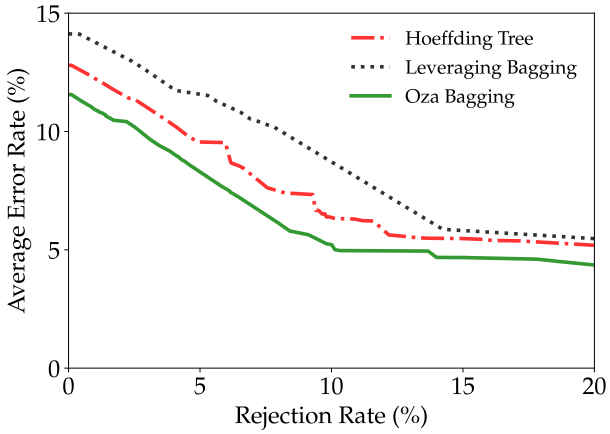


Fig. 10: Rejection and average error rate tradeoff of each single stream learning classifier, used by our stream learning pool on February data.

Figure 10 shows the Pareto curve of the error rejection tradeoff for every single classifier of February, considering only the optimal set of operation points for every single classifier. In such a case, the average error rate is measured as the FP and FN rates average. The rejection rate was measured according to the rate of instances in which our model suppressed the classification outcome. It is possible to note that the classification evaluation through the classification confidence values can be used to improve the system's accuracy, even when outdated ML models are used. For instance, the OzaBagging classifier decreased the error rate from up to 12% to only 5% when rejecting up to only 10% of instances. Therefore, the classification evaluation approach can be used to maintain the system's accuracy while applying a model update task.

To answer $RQ5$, we evaluated how our proposed model performs when no periodic model updates are conducted, meaning rejecting only potential misclassifications as established by our verifier scheme. The evaluation goal is to measure how our model can be applied, even if outdated, by assessing only the classification reliability of the classified instances. The acceptance threshold of every single classifier is set at a 10% error rate (Figure 10), and the intrusion detection task is conducted without model updates throughout the year. It is

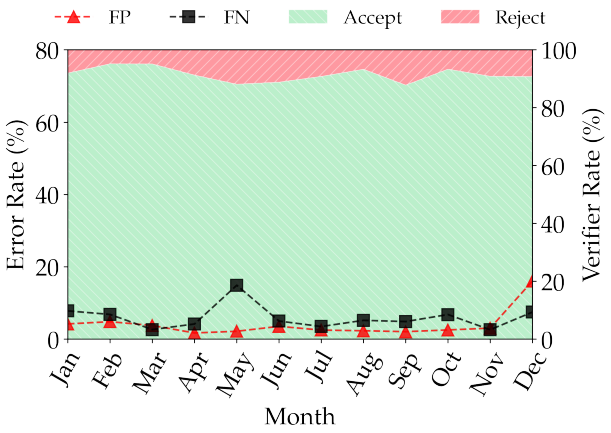


Fig. 11: Proposed scheme accuracy and rejection rate over time without periodic model updates.

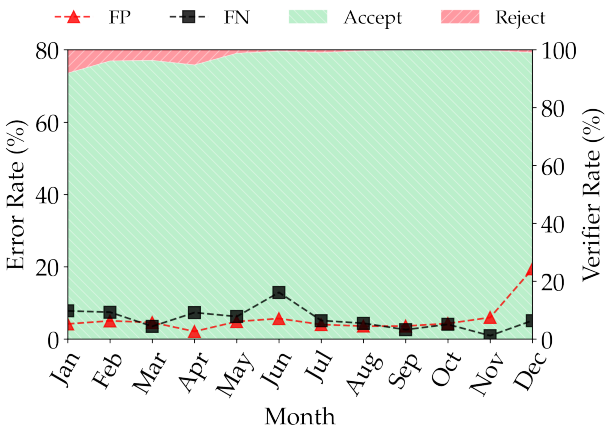
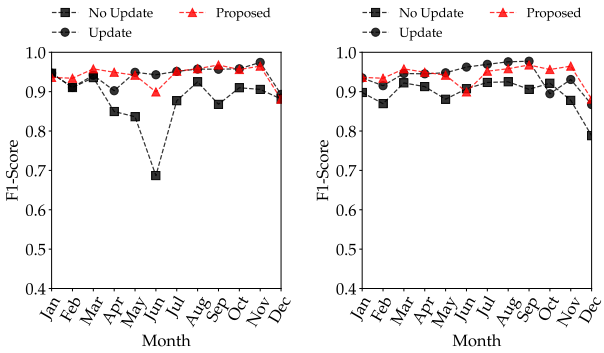


Fig. 12: Proposed scheme accuracy and rejection rate over time with monthly model updates being performed using rejected instances with 30 days old for model updates.

important to note that the operation point must be established according to the operator's needs. Although a higher rejection rate is able to provide higher accuracies over time, a higher number of network events are rejected, whereas a lower rejection rate accepts more network events but is susceptible to producing higher error rates as time passes. Recall that our proposed model (Figure 9) relies on an ensemble of classifiers implemented using the HT, Bag, and Oza, and the detection is applied through a majority voting rationale. Thus, our model rejects a given instance if all single classifiers also reject it, and only the accepted classification by every single classifier is used in majority voting (Algorithm 1).

Figure 11 shows the accuracy and rejection performance of our model without periodic model updates being applied. It is possible to note that our model can maintain the system accuracy rate over time compared to January, even with outdated underlying classifiers. More specifically, our scheme maintains the system accuracy for most of the dataset months, despite the rejection rate, even if no model updates are conducted. On average, our scheme without model updates, and using only the classification evaluation technique, improved the FP rate by 12% when compared to the traditional technique without a rejection (Figure 11 versus Figure 2d), while rejecting an average of only 8.5% of events. Consequently, the classification evaluation approach can be used to maintain the system accuracy over time while an updated model remains unavailable or even improve the model lifespan of the intrusion detection scheme if no model updates are planned to be performed.

To answer $RQ6$, we evaluate the performance of our proposed model with periodic model updates being applied using the old rejected instances. The same previously used classification acceptance operation points were applied to reject the instances by our model. The rejected instances were stored and used in incremental model updates after 30 days had passed since rejection (Algorithm 2, the parameter d), while the model update task is executed in a monthly periodicity. For



(a) Batch Learning (b) Stream Learning

Fig. 13: F1-Score comparison of our proposed model $vs.$ the ensemble approach of both traditional stream and batch learning techniques.

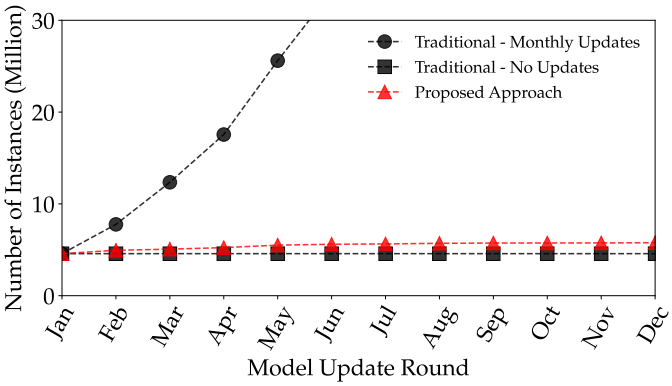
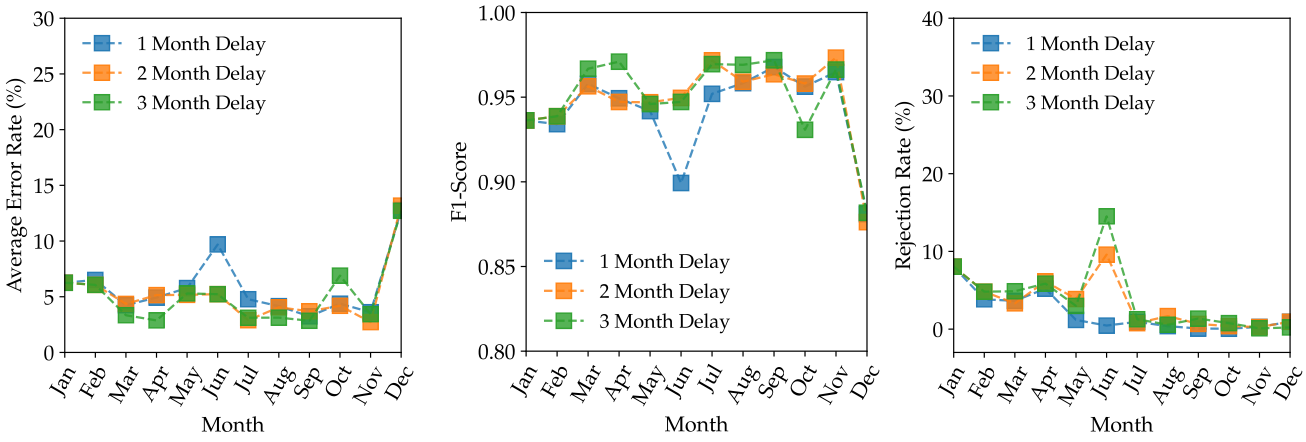


Fig. 14: Cumulative number of instances demanded during model updates of our proposed scheme $vs.$ traditional techniques.

instance, on February 28th, our stream learning classifier pool is updated with the instances that were rejected by our model



(a) Accuracy Rate (b) F1 Score (c) Rejection Rate

Fig. 15: Average error, F1 score, and rejection rate over time by varying the storage interval of rejected instances for the use of model updating in our proposal.

from January 1st to January 31th. Instances were rejected a month before the model update task execution. The storage window time should be defined according to the operator's needs, considering the label provider technique used by the network operator.

Figure 12 shows the accuracy and rejection rates of our proposed scheme when monthly model updates are applied with rejected instances that were stored for at least 30 days. In such a case, our proposed scheme can significantly decrease the error rates while significantly rejecting fewer instances over time compared to its no-update counterpart. More specifically, on average, model updates improved the FP rate by 2% while rejecting only 2% of instances, a decrease in the rejection rate by 6.5% when compared to its no-update counterpart (Figure 11 versus Figure 12). Therefore, our proposed scheme can maintain the accuracy rates of intrusion detection over time while significantly easing the model update process when needed.

We further investigate the accuracy performance of our model when periodic model updates are conducted using the 30-day-old rejected instances. Figure 13 compares the performance accuracy of our model when compared to the traditional ensemble approach of both stream and batch learning techniques. Our scheme, with monthly updates, was able to provide lower error rates when compared to the evaluated techniques without updates and with monthly updates. More specifically, when compared to the batch ensemble approach, our scheme improved the F1 score by an average of 0.08 and 0.03 compared to the no-updates and monthly updates, respectively (Figure 12 versus Figs. 3d and 5d), which represents an improvement of 5.7% and 0.6% on the average error rate.

When compared to the stream approach, our scheme improved the F1 Score by an average of 0.06 and 0.03 compared to the no-updates and monthly updates, respectively (Figure 12 versus Figs. 2d and 4d), thus, also improving the average error rate by up 6.1% and 1% respectively. Recalling that an improvement in F1 Score, was achieved while rejecting only an average of 2% of the instances.

Nonetheless, despite the improved accuracy of our model, we were also able to decrease the challenge regarding the model update task significantly. Figure 14 shows the number of instances demanded by our model when compared to both no-update and monthly update techniques. Our proposed scheme increased the accuracy rates while demanding an average of only 2.2% of the labeled instances demanded by the traditional monthly update techniques. In addition, when compared to the no-update schemes, our proposed model incurred only 2% of additional instances to be provided over time. Therefore, despite the improvements in the accuracy of our proposal, as the main contribution, the model update task is significantly eased when considering the low requirements on the number of labeled instances to be provided as time passes. Furthermore, our proposal's instances used in the model updates can be easily labeled when considering the high storage interval applied during the model updates.

To answer question $RQ7$ we further investigate how the delay of model update tasks through rejected instances (Algorithm 2, *Storage Interval* d) can be increased without impact on accuracy or rejection. As evaluated previously, our proposed scheme can last long periods without periodic model updates without affecting the system accuracy (see Figure 11). Thus, we further investigate the effect of increasing the storage intervals on both accuracy and rejection rate. Figure 15 shows the average error rate, F1 Score, and rejection rate when rejected instances are stored for more extended periods before being used for model updates. It is possible to note that higher storage intervals do not significantly affect accuracy or rejection rate over time. For instance, increasing the storage interval of rejected instances from 30 days to 90 days increases the average rejection rate by only 1.7%, decreasing the average error rate by only 0.6%. As a result, network operators can store rejected instances for more extended periods if their label

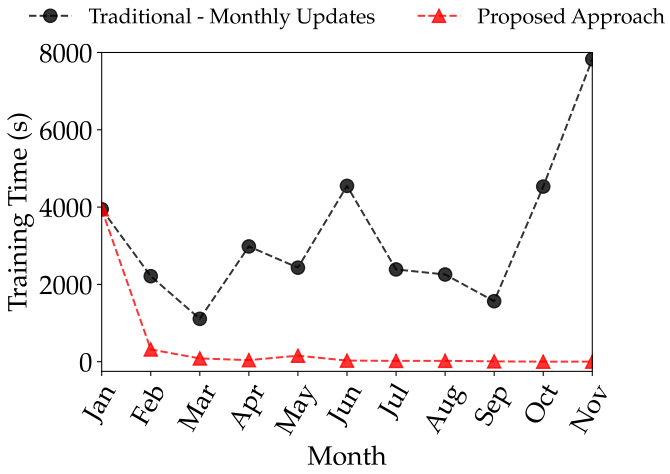


Fig. 16: Computational costs of model updates of our proposed scheme vs traditional monthly updated stream learning ensemble technique.

provider technique demands it, further easing the model update task without significant tradeoffs on accuracy and rejection rates.

Finally, we answer $RQ8$ and evaluate the computational costs of our proposal when compared to the traditional ensemble stream learning technique with monthly model updates (Figure 12 versus Figure 4d). Similarly, the experiments were conducted on commodity hardware, and the computational costs were computed according to the sum of the CPU usage time from all available cores. Figure 16 shows the computational costs of our proposal when periodic model updates are applied through the 30-day-old rejected instances (as performed in Figure 12). It should be noted that our proposed model, when compared to the traditional ensemble approach with monthly model updates, demands an average of only 3.2% of the computational costs.

As a result, the proposed model can significantly ease the model update task by increasing the model lifespan, reducing the number of labeled instances provided while also demanding significantly lower computational costs during model updates. For instance, consider an ML-based NIDS deployed in production for the classification of the *MAWIFlow* dataset. Throughout a 12-month period, with monthly model updates, our proposed scheme would demand a total of $4,6$ thousand seconds of training computational time, while the traditional technique would demand a total of $35,7$ thousand of seconds of training computational time, a 7.67 fold increase. In addition, if we assume that each instance demands 264 bytes for its storage (Table I, 66 features with 4 bytes per feature), our model would demand monthly average storage capacities of 133 GB, while the traditional approach would demand each month an additional of 958 GB, a 7.2 fold increase per month. Therefore, our proposed model makes the model update task more feasible, considering the storage rejection period until the label becomes publicly available, and significantly decreases the computational and storage costs for model updates.

VII. Conclusion

In the literature in general, the authors overlook the challenges that the evolving network behavior may cause on their proposed schemes, which, in practice, will require frequent model updates to be conducted. However, the model update is a challenging task in NIDS due to the huge amounts of network traffic that must be evaluated, labeled, and used during the model training phase. This paper makes the model update feasible through a stream learning classifier pool, a classification evaluation approach, and delayed model updates. The stream learning algorithms have enabled the easy incorporation of new network traffic behavior, whereas the classification assessment provides a reliable classification even when the underlying models are outdated. As a result, even without periodic model updates, our model can keep its accuracy over time and even improve its FP rates by up to 12% compared to the traditional stream learning classification approach. The delayed model updates enable the labeling task of new network behavior to be efficiently conducted after a proper attack disclosure in a public repository. Therefore, the proposed model can perform model updates without human assistance, waiting up to 3 months for the proper event label to be provided without impact on the system accuracy, while demanding only 3.2% of the computational time and 2% of new instances to be labeled as time passes, making model updates in NIDS a feasible task.

The future work will extend the proposed model into a federated learning architecture, further easing the model update process.

The dataset used in the experiments described throughout the present paper is publicly available for download at *https://secplab.ppgia.pucpr.br/idsovertime*.

Acknowledgment

This work was partially sponsored by Brazilian National Council for Scientific and Technological Development (CNPq), grant nº 430972/2018-0).

References

[1] *Kaspersky Security Bulletin 2020. Statistics*, 2020. [Online]. Available: https://securelist.com/kaspersky-security-bulletin-2020-statistics/99804/

[2] B. Molina-Coronado, U. Mori, A. Mendiburu, and J. Miguel-Alonso, "Survey of network intrusion detection methods from the perspective of the knowledge discovery in databases process," *IEEE Trans. on Network and Service Management*, vol. 17, no. 4, pp. 2451–2479, Dec. 2020.

[3] C. Gates and C. Taylor, "Challenging the anomaly detection paradigm: A provocative discussion," in *Proc. of the Workshop on New Security Paradigms (NSPW)*, 2006, pp. 21–29.

[4] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010.

[5] G. W. Cassales, H. Senger, E. R. de Faria, and A. Bifet, "IDSA-IoT: An intrusion detection system architecture for IoT networks," in *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, Jun. 2019. [Online]. Available: https://doi.org/10.1109/iscc47284.2019.8969609

[6] I. F. Kilincer, F. Ertam, and A. Sengur, "Machine learning methods for cyber security intrusion detection: Datasets and comparative study," *Computer Networks*, vol. 188, p. 107840, Apr. 2021. [Online]. Available: https://doi.org/10.1016/j.comnet.2021.107840

[7] 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, Apr. 2019.

[8] X. Gao, C. Shan, C. Hu, Z. Niu, and Z. Liu, "An adaptive ensemble machine learning model for intrusion detection," *IEEE Access*, vol. 7, pp. 82 512–82 521, 2019. [Online]. Available: https://doi.org/10.1109/access.2019.2923640

[9] A. Nisioti, A. Mylonas, P. D. Yoo, and V. Katos, "From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3369–3388, 2018. [Online]. Available: https://doi.org/10.1109/comst.2018.2854724

[10] M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, "Multi-stage optimized machine learning framework for network intrusion detection," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1803–1816, Jun. 2021. [Online]. Available: https://doi.org/10.1109/tnsm.2020.3014929

[11] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking," in *Proc. of the 6th Int. Conf. on emerging Networking EXperiments and Technologies (CoNEXT)*, 2010.

[12] A. Blaise, M. Bouet, V. Conan, and S. Secci, "Detection of zero-day attacks: An unsupervised port-based approach," *Computer Networks*, vol. 180, p. 107391, Oct. 2020. [Online]. Available: https://doi.org/10.1016/j.comnet.2020.107391

[13] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: A survey," vol. 37, pp. 132–156, Sep. 2017. [Online]. Available: https://doi.org/10.1016/j.inffus.2017.02.004

[14] S. U. Din, J. Shao, J. Kumar, W. Ali, J. Liu, and Y. Ye, "Online reliable semi-supervised learning on evolving data streams," vol. 525, pp. 153–171, Jul. 2020. [Online]. Available: https://doi.org/10.1016/j.ins.2020.03.052

[15] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems - SIGMETRICS '05*. ACM Press, 2005. [Online]. Available: https://doi.org/10.1145/1064212.1064220

[16] E. K. Viegas, A. O. Santin, and L. S. Oliveira, "Toward a reliable anomaly-based intrusion detection in real-world environments," *Computer Networks*, vol. 127, pp. 200–216, Nov. 2017.

[17] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A detailed investigation and analysis of using machine learning techniques for intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 686–728, 2019. [Online]. Available: https://doi.org/10.1109/comst.2018.2847722

[18] Y. Zhong, W. Chen, Z. Wang, Y. Chen, K. Wang, Y. Li, X. Yin, X. Shi, J. Yang, and K. Li, "HELAD: A novel network anomaly detection model based on heterogeneous ensemble learning," *Computer Networks*, vol. 169, p. 107049, Mar. 2020. [Online]. Available: https://doi.org/10.1016/j.comnet.2019.107049

[19] H.-L. Nguyen, Y.-K. Woon, and W.-K. Ng, "A survey on data stream clustering and classification," *Knowledge and Information Systems*, vol. 45, no. 3, pp. 535–569, Dec. 2014. [Online]. Available: https://doi.org/10.1007/s10115-014-0808-1

[20] L. Bilge and T. Dumitras, "Before we knew it," in *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*. ACM Press, 2012. [Online]. Available: https://doi.org/10.1145/2382196.2382284

[21] P. Sangkatsanee, N. Wattanapongsakorn, and C. Charnsripinyo, "Practical real-time intrusion detection using machine learning approaches," *Computer Communications*, vol. 34, no. 18, pp. 2227–2235, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S014036641100209X

[22] F. Safara, A. Souri, and M. Serrizadeh, "Improved intrusion detection method for communication networks using association rule mining and artificial neural networks," vol. 14, no. 7, pp. 1192–1197, Apr. 2020.

[23] J. Gu, L. Wang, H. Wang, and S. Wang, "A novel approach to intrusion detection using SVM ensemble with feature augmentation," *Computers & Security*, vol. 86, pp. 53–62, Sep. 2019. [Online]. Available: https://doi.org/10.1016/j.cose.2019.05.022

[24] S. Otoum, B. Kantarci, and H. T. Mouftah, "A novel ensemble method for advanced intrusion detection in wireless sensor networks," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. IEEE, Jun. 2020. [Online]. Available: https://doi.org/10.1109/icc40277.2020.9149413

[25] A. Chiche and M. Meshesha, "Towards a scalable and adaptive learning approach for network intrusion detection," vol. 2021, pp. 1–9, Jan. 2021. [Online]. Available: https://doi.org/10.1155/2021/8845540

[26] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: A survey," *Information Fusion*, vol. 37, pp. 132–156, Sep. 2017. [Online]. Available: https://doi.org/10.1016/j.inffus.2017.02.004

[27] U. Adhikari, T. H. Morris, and S. Pan, "Applying hoeffding adaptive trees for real-time cyber-power event and intrusion classification," *IEEE Transactions on Smart Grid*, vol. 9, no. 5, pp. 4049–4060, Sep. 2018. [Online]. Available: https://doi.org/10.1109/tsg.2017.2647778

[28] N. Martindale, M. Ismail, and D. A. Talbert, "Ensemble-based online machine learning algorithms for network intrusion detection systems using streaming data," *Information*, vol. 11, no. 6, p. 315, Jun. 2020. [Online]. Available: https://doi.org/10.3390/info11060315

[29] G. Pu, L. Wang, J. Shen, and F. Dong, "A hybrid unsupervised clustering-based anomaly detection method," *Tsinghua Science and Technology*, vol. 26, no. 2, pp. 146–153, 2021.

[30] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," vol. 37, no. 3, pp. 50–60, May 2020. [Online]. Available: https://doi.org/10.1109/msp.2020.2975749

[31] B. Hanczar, "Performance visualization spaces for classification with rejection option," *Pattern Recognition*, vol. 96, p. 106984, Dec. 2019. [Online]. Available: https://doi.org/10.1016/j.patcog.2019.106984

[32] D. Lin, L. Sun, K.-A. Toh, J. B. Zhang, and Z. Lin, "Biomedical image classification based on a cascade of an SVM with a reject option and subspace analysis," *Computers in Biology and Medicine*, vol. 96, pp. 128–140, May 2018. [Online]. Available: https://doi.org/10.1016/j.compbiomed.2018.03.005

[33] L. B. Marinho, J. S. Almeida, J. W. M. Souza, V. H. C. Albuquerque, and P. P. R. Filho, "A novel mobile robot localization approach based on topological maps using classification with reject option in omnidirectional images," *Expert Systems with Applications*, vol. 72, pp. 1–17, Apr. 2017. [Online]. Available: https://doi.org/10.1016/j.eswa.2016.12.007

[34] E. K. Viegas, A. O. Santin, V. V. Cogo, and V. Abreu, "Facing the unknown: A stream learning intrusion detection system for reliable model updates." Springer International Publishing, 2020, pp. 898–909. [Online]. Available: https://doi.org/10.1007/978-3-030-44041-1_78

[35] R. Magán-Carrión, D. Urda, I. Díaz-Cano, and B. Dorronsoro, "Towards a reliable comparison and evaluation of network intrusion detection systems based on machine learning approaches," vol. 10, no. 5, p. 1775, Mar. 2020.

[36] H. Wu, Z. Yu, G. Cheng, and S. Guo, "Identification of encrypted video streaming based on differential fingerprints," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, Jul. 2020. [Online]. Available: https://doi.org/10.1109/infocomwkshps50562.2020.9162914

[37] H. Wang, M. J. Bah, and M. Hammad, "Progress in outlier detection techniques: A survey," *IEEE Access*, vol. 7, pp. 107 964–108 000, 2019. [Online]. Available: https://doi.org/10.1109/access.2019.2932769

[38] M. Tavallaee, N. Stakhanova, and A. A. Ghorbani, "Toward credible evaluation of anomaly-based intrusion-detection methods," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 40, no. 5, pp. 516–524, 2010.

[39] MAWI, "MAWI Working Group Traffic Archive - Samplepoint F," 2021. [Online]. Available: https://mawi.wide.ad.jp/mawi/

[40] R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti, "Similarity-based android malware detection using hamming distance of static binary features," vol. 105, pp. 230–247, Apr. 2020. [Online]. Available: https://doi.org/10.1016/j.future.2019.11.034

[41] G. Fumera, F. Roli, and G. Giacinto, "Reject option with multiple thresholds," *Pattern Recognition*, vol. 33, no. 12, pp. 2099–2101, Dec. 2000. [Online]. Available: https://doi.org/10.1016/s0031-3203(00)00059-5