BigFlow: Real-time and Reliable Anomaly-based Intrusion Detection for High-Speed Networks

Eduardo Viegas[1,2], Altair Santin[1], Alysson Bessani[2], Nuno Neves[2]
[1]Graduate Program in Computer Science / Pontifical Catholic University of Parana, Curitiba, Parana, Brazil
[2]LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Lisboa, Portugal
{eduardo.viegas, santin, vilmar.abreu}@ppgia.pucpr.br, { anbessani, nfneves}@ciencias.ulisboa.pt

Abstract— **Existing machine learning solutions for network-based intrusion detection cannot maintain their reliability over time when facing high-speed networks and evolving attacks. In this paper, we propose BigFlow, an approach capable of processing evolving network traffic while being scalable to large packet rates. BigFlow employs a verification method that checks if the classifier outcome is valid in order to provide reliability. If a suspicious packet is found, an expert may help BigFlow to incrementally change the classification model. Experiments with BigFlow, over a network traffic dataset spanning a full year, demonstrate that it can maintain high accuracy over time. It requires as little as 4% of storage and between 0.05% and 4% of training time, compared with other approaches. BigFlow is scalable, coping with a 10-Gbps network bandwidth in a 40-core cluster commodity hardware.**

Keywords—Data Stream, Stream Learning, Classification Reliability, Anomaly-based Intrusion Detection

I. INTRODUCTION

According to the CISCO network forecast report, the worldwide network traffic in 2016 was 96 EB/month and is expected to reach 278 EB/month in 2021 [1]. Current network devices can reach a bandwidth of 100 Gbps, and there are plans to support a bandwidth of 400 Gbps in the near future [2]. Moreover, recent network-based cyber-attacks also take advantage of this scenario to hide themselves—given that they deceive detection engines—by taking advantage of the large amount of data that should be inspected in a very short time. For instance, in October 2016, a DDoS attack with 100 thousand malicious endpoints surpassed a bandwidth of 1.2 Tbps in a domain name server infrastructure. Attacks of this kind can potentially bring down several sites in US and Europe, including Twitter, Netflix, and CNN [3]. Nonetheless, reports of attacks reaching more than 100 Gbps of traffic are becoming surprisingly common nowadays [3, 4]. Therefore, operators need access to solutions for real-time analysis of such malicious content over those massive network attacks.

Current approaches for network traffic measurement and analysis in the Big Data context often rely on Hadoop-based clusters [5, 6]. In general, they store packets as raw data (pcap) to a distributed filesystem (e.g., HDFS [7]) and process them later. Although such approaches offer significant improvements in scalability [5], they lack applicability to real-world environments because in such settings, the network traffic must be analyzed at line speed for a delay-free intrusion detection.

Current methods for discovering new network attacks mostly use unsupervised machine learning (ML) techniques, which typically require storing the network traffic over a certain time for identifying unknown anomalies [8]. However, owing to the massive amount of network packets, their storage for further analysis is not feasible in most scenarios [9]. Thereby, to enable the near real-time (as close as possible to the network throughput) detection of threats, supervised ML techniques should be considered [10]. When using these methods, the traffic behavior is in general represented as a model, resulting from a computationally expensive process (the training stage). Afterward, the classifier uses the obtained model, to categorize the input events as either normal or attacks.

However, as the behavior of traffic changes, either due to new types of malicious actions or alterations in the transmitted content (e.g., due to the offering of new services [9, 11]), the attack models require constant revision. Consequently, the model's accuracy observed on the training dataset might not be evidenced on unseen data. In such a case, the intrusion detection engine will no longer be trusted by the operator given that the alarms are not generated as expected [9]. In this paper, we assess this accuracy loss experimentally, using a real network traffic dataset spanning a year and four ML classifiers. Our experiments show that the accuracy of classifiers trained in the beginning of the year can decrease by up to 23% during the year.

The identification of changes in the network behavior is a challenging task, which often requires human intervention for the reevaluation of the current model's error rate. Thus, to achieve reliability, the model must be periodically tested and updated (e.g., every month). This requires human intervention not only for rebuilding the model (which takes time and storage) but also for ensuring that the production model is operational, with acceptable error rates.

This paper proposes *BigFlow*, a system for reliable real-time network traffic classification in high-speed networks. Our proposal is based on two main insights. First, *BigFlow* determines whether the classification outcome should be accepted or not, in contrast to traditional approaches, which always classify events as normal or attack. The purpose is to make the administrator aware that a possible change has occurred in the network traffic behavior. In this sense, when an event is rejected, there is a high probability that a new network traffic behavior is taking place. Although classification rejection has been used in other areas (e.g., for optical character recognition (OCR) [12] or medical diagnosis [13]), in these areas contextual information can help to identify pattern deviations; however, in the high-speed network traffic field, such a task is challenging. The main challenge that is not present in other areas [12, 13] relates to rejections based on the classifier confidence. This is because a classifier may become unreliable when facing unseen network traffic behavior, thereby committing classification mistakes with high confidence [52]. The second insight relates to the fact that *BigFlow* employs stream learning techniques [20] to analyze traffic in near real time. Such techniques support incremental model updates based on the rejected instances. The expectation is that after a period (e.g., within

one week), the rejected event is properly classified by an expert or a tool (e.g. signature-based network-based intrusion detection system - NIDS) based on public information (e.g., new indicators of compromise). A major advantage of this approach is that the incremental model updates, that incorporates new knowledge into the model, is based only on correctly classified events. This decreases the risk of inaccurate detections, which may lead to a high rate of false positives when processing further packets. Moreover, incremental model updates significantly decrease training time because the current model is not discarded, which is advantageous for high-speed networks.

Rejecting low-confidence classifications in an NIDS – the key idea of this work – leads to two important benefits: better detection accuracy (i.e., fewer misclassifications) and the identification of new characteristics of the evolving traffic, which are then used to incrementally update the classifier model. These benefits improve *BigFlow* reliability over time, even if the network's traffic behavior changes, at the same time significantly decreasing the amount of computational and storage resources needed to operate the system.

In combination, these techniques make *BigFlow* scalable with the number of nodes employed in the system (with a network traffic processing capacity of up to 10 Gbps in our experiments), without losing accuracy over time.

In summary, the paper includes four contributions:

1. We provide the first publicly available dataset for benchmarking intrusion detection engines over a long period, called *MAWIFlow*. This dataset contains real and labeled network traffic records with 158 features each, extracted from 15-min-long daily traces spread over a year of real network traffic. *MAWIFlow* is composed of over 6 billion network flows with almost 8 TB of data;
2. We analyze the behavior of several traditional ML classifiers using *MAWIFlow*. Our findings show that current approaches are unable to cope with traffic changes observed in real networks, and their accuracy decreases significantly in a few months after the training period;
3. We present *BigFlow*, a reliable stream learning intrusion detection engine that can maintain its accuracy over long periods of time. Our solution evaluates the classification reliability, while it allows to incrementally update the intrusion detection engine. *BigFlow* requires as little as 4% of storage and from 0.05% to 4% of training time, compared with current intrusion detection approaches in the literature;
4. We address the problem of network traffic classification using big data streaming processing, without data persistence, aiming to scale up to relevant data rates on commodity hardware. Our experiments show that *BigFlow* can cope with a 10 Gbps traffic rate in a 40-core cluster of commodity hardware.

The remainder of this paper is organized as follows: Section II presents the background for *BigFlow*; Section III presents the *MAWIFlow* dataset and evaluation using several traditional ML schemes; Section IV describes the *BigFlow* proposal, while Section V describes the prototype architecture and implementation; Section 6 VI presents the evaluation of our solution; Section VII describes the related works; and Section VIII concludes our work.

II. Background

A. Stream Processing

BigFlow is built on top of a stream processing platform for dealing with large volumes of network traffic in near real-time. Stream processing platforms (e.g., Apache Storm [18] and Apache Flink [19]) receive data from registered sources and compute over such data through a set of processing elements (PE). Each PE is responsible for a specific operation on the arriving data and for sending the result to another PE, until the computation completes. In general, the messages transmitted through the PEs can be forwarded according to three approaches: *shuffle*, *keyed*, or *broadcast*. In the shuffle approach, the PE messages are sent to another PE in a uniformly distributed manner. The keyed approach groups messages according to a key (e.g., IP address) and sends them to the PE associated with it. Finally, the broadcast approach transmits the messages to every PE of the same type. The near real-time processing using such platforms is achieved by keeping the computation in each PE type as small as possible, and by distributing the message load uniformly through many PE in parallel.

B. ML for Intrusion Detection

In general, network attacks are detected using either signature-based or anomaly-based techniques [30]. In the former approach, the attack patterns must be known and implemented in the system because the detection of attacks is achieved by scanning the packets for well-known attack patterns. The main drawback of this approach is the high number of patterns that need to be stored/analyzed as every attack has a unique signature [31]. Nonetheless, attackers often make changes to already known attacks to evade this detection technique. For instance, only in the first quarter of 2017, more than 55 thousand attack variations for only 15 attack families were discovered [32].

Recently, anomaly detection has been done using ML techniques, which can be broadly divided into unsupervised and supervised categories. Unsupervised ML techniques are simpler to use but usually result in many false positives [8]. Therefore, they are seldom used in practice. Supervised ML methods require a model of the network's behavior, which is built in a computationally expensive process – the *training stage* – using a training dataset [10]. Afterward, the built model can be used in production (real-world environment) by a classifier algorithm, to classify input events as either *normal* or *attacks*. Thus, as long as the network traffic behavior follows the same pattern captured in the training stage, the constructed classifier model can be used for the real-time detection of threats [10].

When using ML for intrusion detection, the network traffic behavior is represented by a set of features. In general, when network-level attacks are considered, the features are usually extracted according to the network flow. Table 1 lists a subset of the features that were used throughout the experiments in this paper. The features in Table 1 are divided into two groups: *Host-based* and *Flow-based*. The former refers to the features extracted from all the data sent from a specific host during a period. In contrast, the latter refers to the communication between two entities over the network, which can be from source to destination, destination to source, or both.

Unfortunately, general-purpose networks rarely exhibit stable traffic patterns [9, 11]. On the contrary, the set of target concepts (e.g., network traffic classes) learned during the

TABLE I. *NETWORK-LEVEL FEATURE SET USED IN THE EXPERIMENTS THROUGHOUT THIS WORK* [17]

Type	Grouping	Features
Host-based	Host to All	Number of Packets, Number of Bytes, Average Packet Size, Percentage of Packets (PSH Flag), Percentage of Packets (SYN and FIN Flags), Percentage of Packets (FIN Flag), Percentage of Packets (SYN Flag), Percentage of Packets (ACK Flag), Percentage of Packets (RST Flag), Percentage of Packets (ICMP Redirect Flag), Percentage of Packets (ICMP Redirect Flag), Percentage of Packets (ICMP Time Exceeded Flag), Percentage of Packets (ICMP Unreachable Flag), Percentage of Packets (ICMP Other Types Flag), Average Packet Size, Throughput in Bytes, Protocol
Flow-based	Source to Destination	Number of Packets, Number of Bytes, Average Packet Size, Percentage of Packets (PSH Flag), Percentage of Packets (SYN and FIN Flags), Percentage of Packets (FIN Flag), Percentage of Packets (SYN Flag), Percentage of Packets (ACK Flag), Percentage of Packets (RST Flag), Percentage of Packets (ICMP Redirect Flag), Percentage of Packets (ICMP Redirect Flag), Percentage of Packets (ICMP Time Exceeded Flag), Percentage of Packets (ICMP Unreachable Flag), Percentage of Packets (ICMP Other Types Flag), Throughput in Bytes
	Destination to Source	
	Both	

training stage often evolves over time [20]. For instance, the behavior of a network may change because new services are added [9] or owing to modifications of the attacks' execution.

The identification of changes in production networks typically involves a computationally demanding task of model rebuilding, which can only be performed if there is access (storage) of recently observed traffic and prior (manual) classification of events. Furthermore, model rebuilding cannot be postponed, because while a new model is being constructed, the model currently in use should maintain acceptable error rates, ideally as low as the ones observed during the training stage [9]. This makes the process unfeasible for most high-speed networks.

In other fields, a typical approach to deal with evolving environments is to resort to stream learning algorithms [20]. These techniques allow the update of the detection mechanism to be performed at the arrival of each new event, incrementally, without discarding the current model. Thus, the time needed for building an updated classifier model can be shortened [9]. However, these techniques typically rely on supervised learning, in which events need to be previously classified [8]. Moreover, it is necessary to devise a method for event selection that would be suitable for incrementally updating the model [20]. This renders the current approaches not applicable to networked environments [10].

Another solution that has been explored for improving the reliability of ML classifiers – but not for intrusion detection – is to reject classifications [12]. Therefore, the classification outcome may be rejected according to the given event class (*Normal* or *Attack*) probability (confidence). For example, events classified as attacks could only be accepted when their associated confidence measure is above 90%. This approach has been employed in areas where errors have a high associated cost, such as OCR [12] and medical diagnostics [13]. However, in the field of network detection, the reliability of detection is often neglected [9], leading to an unreliable intrusion detection system.

III. *MAWIFLOW DATASET AND ANALYSIS*

In this section, we describe a novel dataset based on real network traffic [22], and experiments in which we evaluated the accuracy of traditional intrusion detection methods over time.

A. MAWIFlow

To benchmark ML-based NIDS, we present the *MAWIFlow* dataset (i.e., a collection of records labeled as either Normal or Attack), assembled based on real network flows collected over a year. A dataset for the tasks that we study should fulfill a number of requirements, including realism and high variability, having labeled data with correctly classified events, being reproducible, and being publicly available [17]. Ideally, the data should be obtained from real network activity, as it provides all the expected properties from an evaluation testbed [8]. However, collecting such data is difficult and, when obtained, its sharing is unlikely owing to privacy concerns [9]. Furthermore, establishing proper event labels for network activity is a challenging task, which often requires human intervention [10].

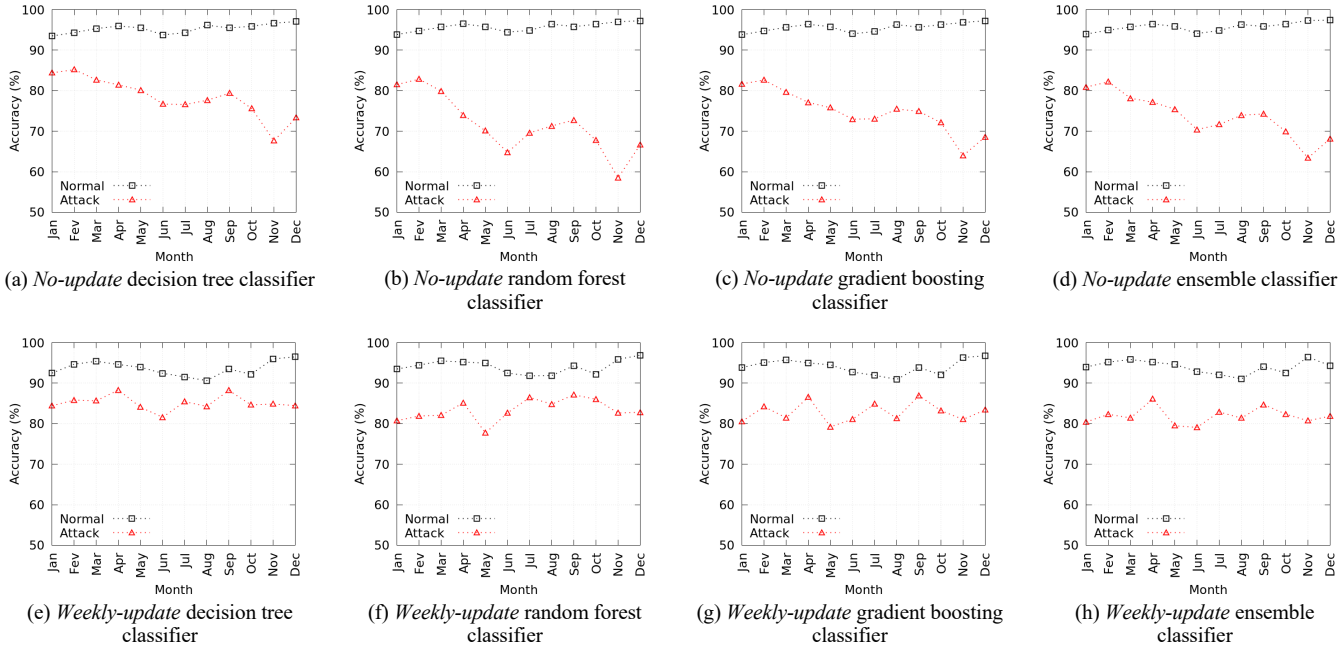
TABLE II. *MAWIFLOW* STATISTICS

Field	Value
Average Daily Network Packets	~110 Millions
Average Daily Network Flows	~22 Millions
Average Daily Throughput	~570 Mbps
Average Daily Anomalous Flows	~1.7 Millions
Average Daily Dataset Size	~21.7 GB
Total Network Packets	~30.36 Billions
Total Network Flows	~6.07 Billions
Total Dataset Size	~7.9 TB

MAWIFlow is based on real and publicly available network traffic. More specifically, it is based on the network flows that were extracted from the MAWI network packets traces [22] (*Samplepoint-F* in MAWI archive), collected daily for a 15-min-long interval, from a transit link between Japan and USA. During the period of recording, the *Samplepoint* was made of a 1Gbps network traffic link. In addition, the network traces are anonymized, i.e. network packet payloads are removed, and sensitive network packet header fields are anonymized. The labeling of records was performed using MAWILab [8], which labels the daily anomalous events (network flows) from MAWI through a combination of several unsupervised anomaly detectors. For the purpose of this work, we consider all of the network traffic available for the year 2016. Network anomalies are classified according to their attack types as labeled by MAWILab. Therefore, network anomalies can be made of several types of *portscan, network scan*, *denial-of-service*, *distributed denial-of-service*, amongst others network-level attacks [22].

The *MAWIFlow* dataset was built using the *BigFlow* feature extraction module (discussed in Section IV.A), which extracted 158 host-based and flow-based features, some of which have been employed in previous works (15 features in [14], 21 features in [15], 60 features in [16], and 62 features in [17]). Table 1 provides a partial list of those features. For the label assignment process, *MAWIFlow* assigns labels that are associated with the flows from which the features were extracted. Table 2 summarizes the *MAWIFlow* dataset. As can be seen, this dataset contains over six billion network flows, extracted by analyzing more than 30 billion network packets (real traffic) for the year 2016.

The original *MAWIFlow* dataset contains over 7.9 TB of data. A stratification process was needed to reduce its size, enabling its sharing and facilitating its use for the NIDS evaluation. Thereby, the *proportional random stratified*



(a) *No-update* decision tree classifier (b) *No-update* random forest classifier (c) *No-update* gradient boosting classifier (d) *No-update* ensemble classifier

(e) *Weekly-update* decision tree classifier (f) *Weekly-update* random forest classifier (g) *Weekly-update* gradient boosting classifier (h) *Weekly-update* ensemble classifier

Figure 1 – Average month accuracy behavior for different classifiers with and without periodic model updates during 2016 in the *MAWIFlow* dataset.

sampling without replacement method [24] was employed to generate the stratified *MAWIFlow* dataset. The resulting dataset comprised just one percent of the original dataset, while it maintained the original proportions of the network traffic classes (Normal and Attack), which were randomly chosen.[1]

Besides being the first publicly available dataset of this kind, *MAWIFlow* overcomes the main challenges associated with building realistic datasets for benchmarking intrusion-detection engines. More specifically, it has all of the desired characteristics described in [38], summarized as follows.

Realism: The network traffic used for building the dataset was obtained from real network traces. Moreover, *MAWIFlow* was built from over a year-long observation data of real network traces, enabling not only evaluation of the detection system during a specific period of time, but also the evaluation of its behavior over time, when facing new network traffic behavior;

Validity: The network traces used for building the *MAWIFlow* dataset were collected from real network traces. Although MAWI (network traces used in *MAWIFlow*) is provided in a sanitized manner, i.e., payload is removed and sensitive data from network packet headers are encrypted, the network flow reconstruction is still possible. In this manner, the sanitization process used by MAWI does not affect the features' values;

Prior labeling: The event labels were identified by state-of-the-art unsupervised ML techniques (assessed by MAWILab). In this manner, supervised ML techniques can be evaluated regarding their performance as compared to unsupervised techniques;

High Variability: *MAWIFlow* is highly variable not only owing to the used network traces but also owing to its long period of recording. The used network traces are real, valid, and collected from real network infrastructure, thereby it presents the expected variability from production environments. Nonetheless, owing to its long period of recording (the entire year 2016), the detection system can be evaluated considering the environment variability during an entire year.

Reproducibility and Public Availability: The used network traces were collected from publicly available sources (MAWI). Moreover, *BigFlow* (Section IV) source code is also publicly available.

B. *Accuracy Degradation of ML Classifiers*

The purpose of the analysis is to determine if ML-based approaches can maintain accuracy over time while processing traffic from real networks. In our evaluation, we considered three individual and different classifiers that are usually employed for intrusion detection: decision tree (DT) [42], random forest (RF) [43], gradient boosting (GB) [44], and an ensemble [45] classifier composed from DT, RF, and GB that decides based on majority voting across each classifier's decisions.

For each of the evaluated classifiers two update schemes were tested: *no-update* and *weekly-update*. The *no-update* scheme used a single training step using the data of *MAWIFlow* from the first seven days of January, and then employed the built model for the remainder of the year. In the *weekly-update* scheme, the model lasted for only seven days, and then a new model was built using the previous seven days of data as training, thus retraining (rebuilding) the classifier 52 times during the year (once every week).

Apache Spark MLib [23] version 2.1.1 was used for the implementation and evaluation of the aforementioned classifiers. The DT information gain criterion relies in gini impurity measure. The RF was composed of 50 decision trees, with a feature subset selection strategy as the square root of the number of decision trees. Finally, for the GB, 50 iterations were used with decision trees as weak learners. Owing to the imbalanced nature of network traffic (in *MAWIFlow* only 1.52% of flows were labeled as anomalies), the *random undersampling without replacement* method [24] was applied during the training stage, to balance the classes (Normal and Attack). The true negative (*Normal accuracy*) and true

[1] In order to validate the stratification procedure, all classifiers (Section III.B) were also evaluated using the original *MAWIFlow* dataset through Apache Spark MLib, the same accuracy behavior was evidenced.

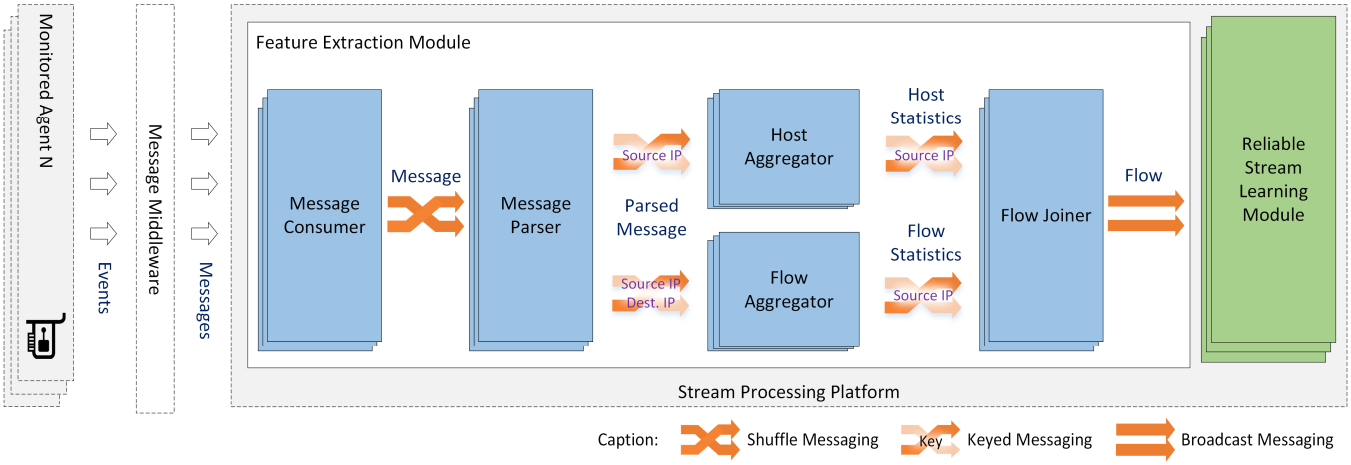


Figure 2 – *BigFlow* real-time feature extraction module architecture for high-speed networks.

positive (*Attack accuracy*) rates are shown in Figure 1. The figure shows the monthly average accuracy of the classifiers in the *no-update* and the *weekly-update* schemes, with the 62 features listed in Table I.[2]

All evaluated classifiers have shown an accuracy impact during the year 2016. Considering the *no-update* scheme (Figures 1-a, 1-b, 1-c, and 1-d), the classifiers were able to maintain accuracy for Attack for the first two months (January and February), while exhibiting a reduction during the remainder of the year. Comparing the average Attack accuracy in January with the rest of the year, we observed a reduction of 6%, 10%, 6.8%, and 7% for the DT, RF, GB, and ensemble classifiers, respectively. The worst case was evidenced in October, with the Attack accuracy drops of 16.8%, 23%, 17.2%, and 17.5% for the DT, RF, GB, and ensemble classifiers, respectively. On the contrary, the accuracy of Normal packets did not significantly change, and in the best case, it increased by 1.2% (ensemble classifier).

With regard to the *weekly-update* classifiers (Figures 1-e, 1-f, 1-g and 1-h), the results demonstrate that the periodic updates helped the classifiers to remain reliable. Their accuracy did not significantly change during the year, and in some cases, even improving compared with their initial accuracy in January. The highest increase in the accuracy was by 2.6% for Attack detection.

In summary, this experiment provides evidence that in production high-speed networks, anomaly detection classifiers must be updated periodically; otherwise, their outputs become unreliable over time. However, regularly updating the classifiers is challenging in high-speed networks, because the networks' activity must be stored for further analysis and should be labeled accordingly.

IV. BIGFLOW

To address the aforementioned evolving behavior of high-speed open networks, we present *BigFlow*, a reliable stream learning intrusion detection system. The goal is to maintain reliability in the outputs of the classifier and high accuracy over time, while substantially reducing the extent of human expert intervention and the amount of data that needs to be stored. The operation of *BigFlow* proceeds in two main stages: *feature extraction* and *reliable stream learning*.

Feature extraction is performed using a traditional stream processing framework. Its purpose is to compute the flow statistics, which are represented as a feature vector (an event or instance, in ML terminology). The flow statistics computation is performed in real time, summarizing the information about the traffic between two hosts in a time interval. Since only the statistical analysis results need to be stored in the memory, during the specified time interval, there is no requirement for the storage of the observed network packets.

The *reliable stream learning* stage receives as input the feature vector (composed from the flow statistics) and classifies it as either Normal or Attack. To operate in near real time, *BigFlow* employs a stream learning classifier with a verifier module. This module decides whether the classification outcome is reliable and should be accepted; otherwise, it is rejected. When an event is rejected, it is stored until it can be labeled. The rejected event is labeled by a human expert, normally by collecting more information about a new behavior, e.g., by consulting a public repository of vulnerabilities/threats such as the common vulnerabilities and exposures (CVE), or by finding that a new type of service is being used in the network. Then, the rejected instance is used to incrementally update the stream learning classifier.

The next subsections describe in detail these two stages, including the architecture of the modules that implement the stages and description of the main components.

A. Feature Extraction

To measure and classify the network activity, it is necessary to compute statistics about the network traffic exchanged between relevant entities over a period of time. There are several works that focus on extracting features for flow classification [14, 15, 16]. However, contrary to *BigFlow*, none of them is capable of monitoring high-speed evolving networks. In such a context, to avoid the storage of network data, the feature extraction process should be performed in near real-time. Thereby, we have established a feature set according to the processing demanded for its extraction, which is, in general, responsible for the most significant part of the overall demanded processing [17] (Section VI.B).

BigFlow can extract up to 158 features. The feature set considers both host (host statistics) and flow (host to host statistics) granularity. Host statistics are features that are extracted based solely on the data sent/received from a

[2] It is important to note that the same behavior was evidenced with the other features sets for all experiments; however, for space purposes only the results obtained using the set of 62 features [17] are shown.

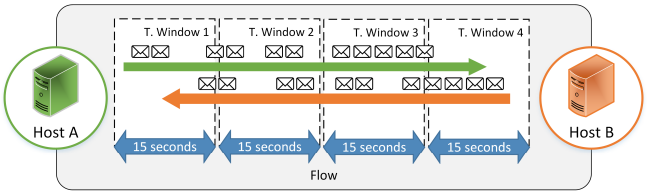


Figure 3 – *BigFlow* flow computation through the *Tumbling Window* approach.

specific host, e.g., percentage of SYN packets sent in a time period. On the other hand, flow statistics features comprise information about the communication between two hosts, e.g., average size of the packets exchanged between the hosts.

The architecture of the feature extraction module of *BigFlow* is shown in Figure 2. Monitored agents (e.g., hosts, network switches or routers) transmit the events through a message middleware. An event corresponds to a unit of analysis, e.g., a network packet or a netflow record. The message middleware acts as a broker of events, being responsible for providing a single interface for the monitored agents.

The *Message Consumer* module acts as the data producer for the feature extraction module. Its only purpose is to receive the available events from the message middleware, regardless of their content or source agent. Each collected event is forwarded to the *Message Parser* module in a PE of stream processing, using the shuffle approach (Section II.A). The *Message Parser* module in turn determines the event's source, fields, and type (e.g., network packet or netflow record).

As an example, consider two distinct monitored agents: a switch and a router. The switch exports network packet headers, while the router exports expired netflow records. The *Message Consumer* module reads both types of events from the message queue, and simply distributes them through the available *Message Parser* module, keeping the computing load even. The *Message Parser* module, in turn, processes the packet headers and netflow records according to each event type, collecting the relevant fields.

The *Host Aggregator* and *Flow Aggregator* modules perform the actual network flow statistics computation (feature extraction). To do that in near real-time and in a distributed manner, both aggregators receive messages through a *keyed* stream. The key for the *Host Aggregator* module is calculated by hashing the event source addresses (source IP address), while the key for the *Flow Aggregator* module relies on the XOR operation on both source and destination addresses (source and destination IP addresses). To divide the load, each module is responsible for a range of hash values. Thus, through XOR'ing, it is possible to forward messages from two specific hosts to the same flow aggregator PE, regardless of the direction taken by a packet.

To compute feature values from the grouped events, *BigFlow* discretizes them in time intervals, referred to as the *Tumbling Window* modules. Each *Tumbling Window* module stores and updates the features' values for a specific period, according to each received event. When a *Tumbling Window* expires (i.e., the period is over), the values of the flow features are exported in a host or flow statistics format, and the computation of the features' values starts over for a new window.

Figure 3 illustrates the *BigFlow* computation through *Tumbling Windows*. The figure considers two hosts exchanging messages over the network for 60 s, and a *Tumbling Window* period of 15 s. To compute the flow statistics, the *Message Parser* module forwards all arriving events exchanged between these two hosts to the same *Host* and *Flow Aggregators*. Each aggregator computes the flow features' values during 15 s ("T.Window 1" in the figure). When a *Tumbling Window* expires, it exports the host and flow statistics to the next module. As a new event arrives after the initial 15 s, the *Host* and *Flow Aggregators* create another *Tumbling Window* ("T.Window 2" in the figure) and start the flow features' computation again.

The usage of *Tumbling Windows* for computing flow features brings two important benefits. First, it ensures that all active flows will expire, without periodic checks, supporting a simple garbage collection mechanism. Second, it ensures that the amount of resources required for the computation of long-lived flow features values remains limited, thus allowing scalable processing.

Finally, the *Flow Joiner* modules are responsible for receiving all host and flow statistics values and for joining them in a single stream. The module receives the exported events through the hash of the source address of either host or flow statistics. Thus, a *Flow Joiner* is responsible for a range of hash values, causing all values from a given subset of hosts to be given to the same module. For each received flow statistic, the *Flow Joiner* aggregates it with the respective host statistics and exports the result to the next module.

Notice that a single host may have several exported flow features, while having a single host feature, e.g., a single host accessing services in several other hosts. Thereby, the *Flow Joiner* module must also store the host flow, to join it with several exported flow features in a single *Tumbling Window*. To this end, the *Flow Joiner* module also relies on the *Tumbling Window* module.

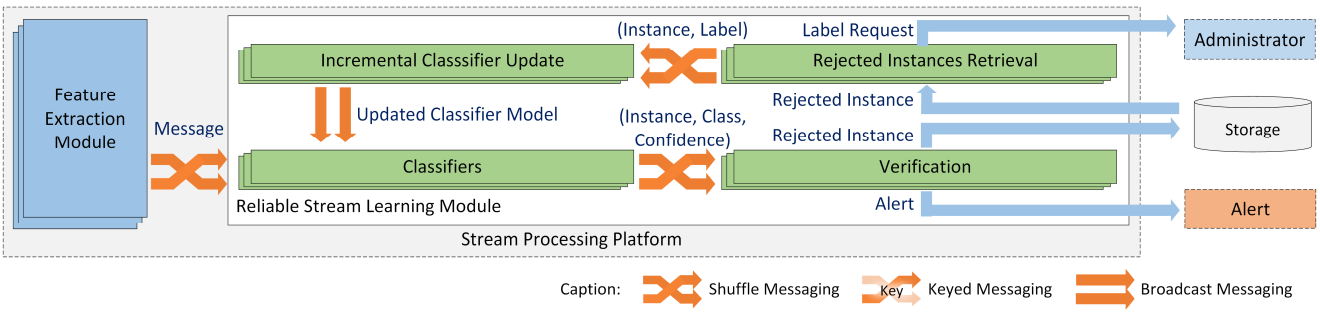


Figure 4 – *BigFlow* reliable stream learning module.

B. Reliable Stream Learning

After the network flow computation, it becomes possible to classify the feature vectors as either Normal or Attack (anomalous). As shown in Section III.B, the network content/traffic changes over time, rendering the classifier unreliable. Thereby, the employed classification mechanism must be able to reliably cope with such changes. In practice in production networks, the model must be updated regularly (e.g., every week), owing to evolving traffic patterns [9]; eventually, if a new vulnerability is critical, an update should be made immediately after rejection.

To deal with the intrinsic evolving nature of the network traffic, *BigFlow* relies on the stream learning intrusion detection. When a classification outcome is rejected, *BigFlow* stores that event until an administrator labels it. The rejected events are used later for incremental updates, thereby minimizing the costs of the model update, while still having an updated classifier model. Figure 4 shows an overview of the *BigFlow* reliable stream-based classification module.

1) Setup

At the startup, *BigFlow* trains the stream learning classifier using a training dataset. A classifier model is obtained and replicated, among several classification processors, to ensure that the classification throughput scales with the number of PEs. During the testing stage, the classification thresholds for each class (Normal or Attack) are defined. The class classification thresholds are used to define whether the classification outcome should be accepted or not.

2) Real-time Learning

The *BigFlow* stream learning module aims to provide reliable classifications, employing a *Verification* module; at the same time, to provide updated ML models, it executes *incremental classifier updates* using the rejected instances.

The verification module receives (from the classifier), the instance, the assigned class, and the classifier confidence measure on the assigned class. Using the classification thresholds established during the setup stage, the verifier module decides whether the classification outcome should be accepted or not. For instance, consider a confidence threshold of 70% for the Attack class; then, the verifier module accepts an instance labeled as Attack only if its confidence level is above 70%; otherwise, the event is rejected.

The rejected instances are stored (Figure 4). Periodically, these instances are retrieved, and their labels are requested by an administrator. This administrator can be a human that verifies the event label using publicly available label sources, such as the CVE, Twitter or security newsfeed, or the one that is able to understand new legitimate applications or traffic behaviors on the network. It can also be an auxiliary system composed of signature-based NIDSs that are periodically and automatically updated with a new indicator of compromises (e.g., Snort [49], and Bro [50]), which hopefully capture novel attack behaviors.

If an event is labeled, the instance and its correct label are used for the incremental model update; otherwise, the event remains stored until its class (normal or attack) becomes publicly known, or a certain threshold time is reached. In the latter case, the instance can be either discarded or assumed to be Normal. For example, a rejected event is stored for a month, and after this time, if it still had not been associated with an Attack using any of the public sources for labeling, it is deemed as a Normal event.

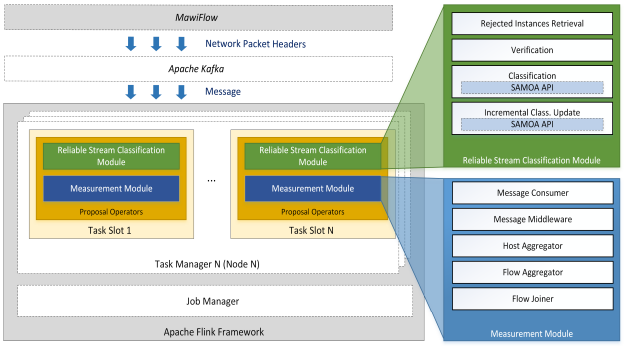


Figure 5 – *BigFlow* architecture.

BigFlow assumes that when an unknown event (attack or not) is classified, the classification confidence level is not reached; thereby, the event is rejected rather than being misclassified. The core idea is simple: high-confidence accepted results represent patterns which the classifier model is still able to identify, while low-confidence results require more attention on the administrator's side as they potentially represent new traffic behaviors that must be learned by the system.

A possible drawback of such an assumption is regarding uncertain classifier output over time [52]. In such a case, the classifier outputs can no longer be used, owing to changes in the network traffic behavior (as already evidenced in other fields [52]). This happens, for instance, when the classification model is not updated in a long period of time. Thereby, a classifier may wrongly classify unseen network traffic with high confidence, making the model no longer reliable. To address such a scenario, our proposal employs an ensemble of stream learning algorithms. The key idea is that a classification can only be accepted when the confidence level is met in all employed stream learning algorithms, i.e., the classification remains reliable as long as there is at least one reliable stream learning algorithm that outputs correct classification.

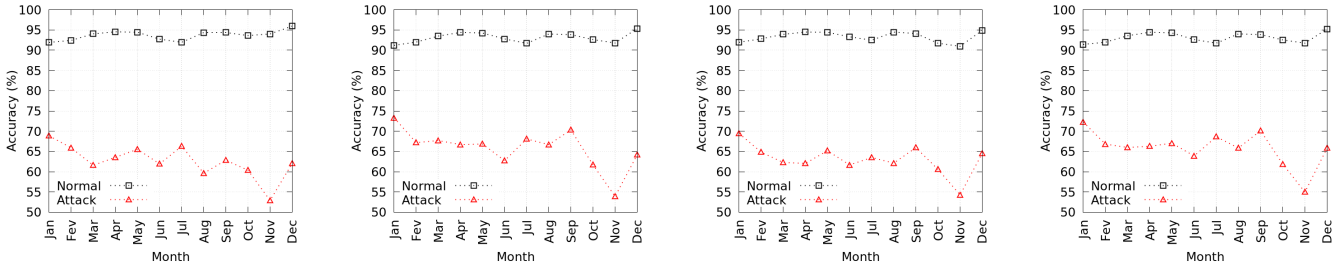
As a result, *BigFlow* provides an updated stream learning classification in near real-time with selective human assistance. This is because only instances that passed through the classifiers and were rejected require action from experts. Thereby, this approach requires minimal human intervention and, most importantly, mitigates the false positives/negatives alarms.

As the models are incrementally updated only with instances that were previously rejected, the proposal also minimizes the cost of model updates.

V. Implementation

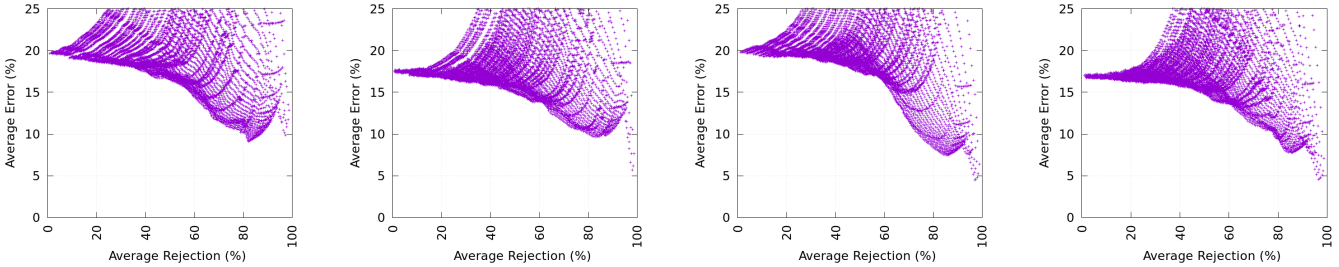
A *BigFlow* prototype was implemented and deployed in a distributed environment, as shown in Figure 5. The prototype takes as input network packet headers from MAWI [22], and for each network packet, its header is exported to the message middleware. The message middleware was deployed through the well-known open-source Apache Kafka, version 0.10.2.0.

Our prototype was implemented on top of Apache Flink stream processing framework [19], version 1.3.0. The proposed windowing mechanisms (*Tumbling Windows*) were also implemented using the native windowing mechanism provided by the Flink. A default value of 15 s for each



(a) *No-update* Hoeffding Tree (b) *No-update* OzaBoosting (c) *No-update* Leveraging Bag (d) *No-update* ensemble classifier

Figure 6 – Average monthly accuracy behavior for different stream learning classifiers without periodic model updates during 2016 in the *MAWIFlow* dataset.



(a) *No-update* Hoeffding Tree (b) *No-update* OzaBoosting (c) *No-update* Leveraging Bag (d) *No-update* ensemble classifier

Figure 7 – Error-reject tradeoff during January 2016 in *MAWIFlow* dataset. *Average error rate* is given by the average of FP and FN rates, whilst *Average rejection rate* is computed by averaging the rejection rate of both Normal and Attack classes. Thresholds for each class, Normal and Attack, was varied from 1.00 to 0.00 in a 0.01 interval, all operation points are shown.

Tumbling Window was used, as it provided the best results after some preliminary evaluation. The customized keyed messaging was implemented using the *KeySelector* Flink interface. The Apache Kafka messages were read through the Apache Flink connector API, version 0.10_2.10.

The reliable stream learning classification module was implemented using the massive online analysis (MOA) library [25], release 16.04. At the startup, the *Classification* and *Incremental Classifiers Update* modules loaded the same classification model. The rejected instances were stored in memory by the *Rejected Instances Retrieval* (Figure 4), which retrieved the rejected instances through Kafka. The PE parallelism level was set according to the number of worker nodes used in our experimental evaluation (Section VI.C).

VI. EVALUATION

The evaluation test was performed in two steps. First, our proposed reliable stream learning module was evaluated in terms of *accuracy over time* using the *MAWIFlow* dataset. Second, we evaluated the *BigFlow performance and scalability* as well as *the cost of updating the stream learning* module.

A. Accuracy

For the evaluation of the *Reliable Stream Classification* module, four stream learning classifiers were evaluated: *Hoeffding Tree* [51], *OzaBoosting* [54], *Leveraging Bag* [55], and an *Ensemble* of the prior three classifiers that performs majority voting on the individual outcomes. Similarly, to the tests conducted in Section III.A, the classifiers were trained using the first seven days of January, and them employed in the remainder of the year, without period updates. The *Hoeffding Tree* was evaluated with a grace period of 200, a Naïve Bayes leaf prediction strategy, information gain with respect to distribution of class values split criterion, and a tie threshold of 0.05. Both *OzaBoosting* and *Leveraging Bag* uses 50 *Hoeffding Trees* as their base learners, in which each base learner also uses the same parameters as the individual *Hoeffding Tree*.

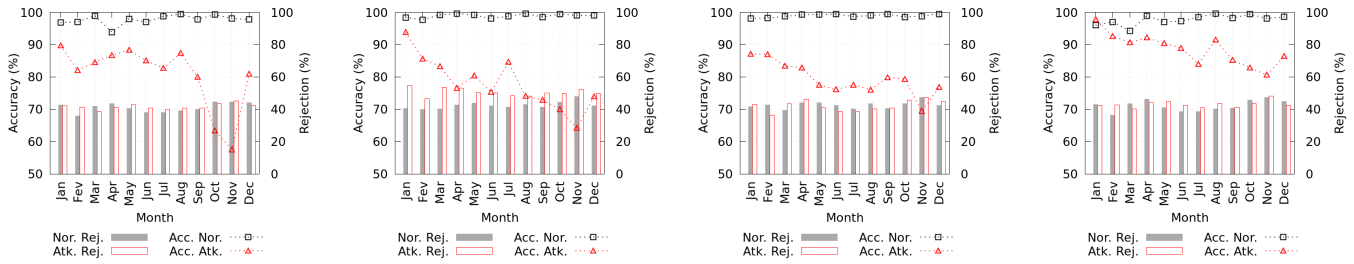
Our evaluation was performed in three steps: without *BigFlow*, with *BigFlow* without updates (i.e., rejecting results but not updating the model), and *BigFlow* with the verifier module and with weekly incremental model updates. A weeklong delay for the incremental model updates for the rejected instances was adopted to mimic the time until an attack label becomes publicly available.

Figure 6 shows how each stream learning classifier performs in *MAWIFlow* dataset without *BigFlow*. The same behavior evidenced in Section III.B can be seen in the absence of period model updates. The accuracy degradation occurs in the first months after training. In 2016, the attack error rate (FP) percentage increases by up to 16%, 20%, 16%, and 18%, for the *Hoeffding Tree*, *OzaBoosting*, *Leveraging Bag*, and *Ensemble* respectively. In contrast, the normal error rate remains similar in the remainder of the year. Thereby, these results also show that to remain reliable for a long period, the intrusion detection model must be updated.

BigFlow updates its models by the means of the rejected instances, which are considered to be unreliable (Section IV.B). To this end, *BigFlow*, through the verifier module, evaluates whether the classifier confidence met a specific threshold, according to the given class (*Class Related Threshold*, CRT [53]). Thereby, in order to evaluate the verifier module, each class must have its threshold set. However, even if a proper class threshold is selected, the event confidence level can be biased, i.e., an unseen event (unknown behavior) may have its confidence level high [52] (see Section IV.B.2). In the light of this, *BigFlow* computes the class confidence for the *Ensemble* classifier according to Equation 1.

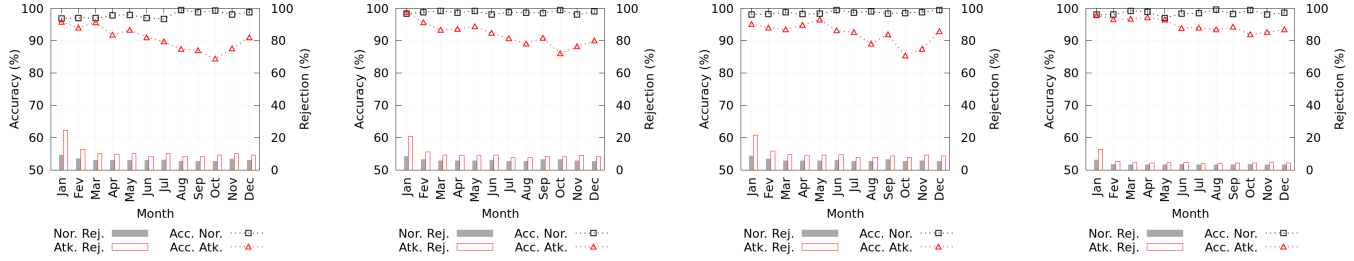
$$Class_{confidence} = \prod_{i}^{n} Class^{i}_{confidence} \qquad (1)$$

In which n denotes the number of used classifiers, and $Class^{i}_{confidence}$ the ith class confidence outcome for a class given by the *Ensemble* classifier. Thereby, only instances that



(a) Hoeffding Tree with verifier module (b) OzaBoosting with verifier module (c) Leveraging Bag with verifier module (d) Ensemble classifier with verifier module

Figure 9 – Average monthly accuracy and rejection rate behavior for different stream learning classifiers without incremental model updates during 2016 in the *MAWIFlow* dataset.



(a) Hoeffding Tree with verifier module (b) OzaBoosting with verifier module (c) Leveraging Bag with verifier module (d) Ensemble classifier with verifier module

Figure 10 – Average monthly accuracy and rejection rate behavior for different stream learning classifiers without incremental model updates during 2016 in the *MAWIFlow* dataset.

have a high confidence for all classifiers will have a high $Class_{confidence}$. The confidence values for each used classifier was computed as a probability value ranging from 0 to 1. The *Hoeffding Tree* confidence value was computed as the class probability as measured by the Naïve Bayes in a given node. For the *Leveraging Bag* the class confidence was computed as the normalized sum of each base learner. On the other hand, the *OzaBoosting* confidence values was computed as the weighted normalized sum of each base learner. For both *Leveraging Bag* and *OzaBoosting* the base learners computes their confidence values by the means of a *Hoeffding Tree*.

Figure 7 shows the relation between the average error rate and the average rejection rate for the evaluated classifiers in the training month (January). The average error rate refers to the average of the FP and FN rates, whilst the average rejection rate refers to the average rejection of both normal and attack events. It is possible to note that, for all evaluated classifiers, one can further reduce the average BigFlow error rate, when a certain rate of rejection can be tolerated. As previously evaluated, the classifiers increase their FP rate over time (Figures 1 and 6). In this sense, the classes confidence thresholds were set according to the attack error rate improvement. Figure 8 shows the non-dominated solutions (best operation points), considering the relation between the attack error rate (FP) and the average rejection rate. It is possible to note that the *Ensemble* classifier presents the best error-reject tradeoff. Moreover, a relation can be seen regarding the attack error rate and the average rejection rate. For instance, if a 20 percent of average rejection rate could be tolerated, the attack error rate percentage can be decreased by 4, and 5, for the *Hoeffding Tree*, *OzaBoosting*, *Leveraging Bag*, and *Ensemble* respectively.

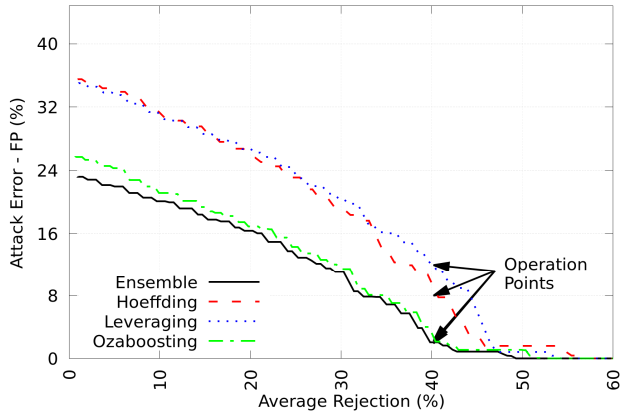
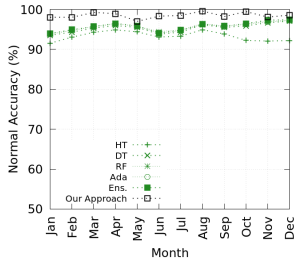


Figure 8 – Non-dominated solutions for the evaluated Stream Learning algorithms. Attack error refers to FN rates. Operation points were chosen at 40 percent of average events rejection rate.

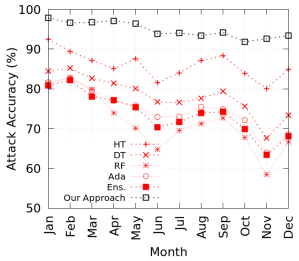
For the remainder of the evaluation tests, the classifiers thresholds were chosen when the average rejection rate met 40 percent (Figure 8, *Operation Points*). The operation points were established in order to enable the evaluation of *BigFlow* without updates (i.e., rejecting results but not updating the model), and *BigFlow* with the verifier module and with weekly incremental model updates. In this sense, a lower rejection rate would not enable the proper evaluation of the model update impact.

Figure 9 shows the average monthly accuracy and rejection rate for the evaluated classifiers with the verifier module set, however, without periodic model updates. Several observations can be made regarding the verifier module. The average rejection rate remains similar to the chosen operation point over time (40 percent). The normal accuracy (TN) significantly improves for all classifiers, reaching up to 99 percent, improving the TN by up to 8 percent. The attack accuracy (TP) also significantly increases, however, in general, it decreases over time, in the absence of incremental model updates.

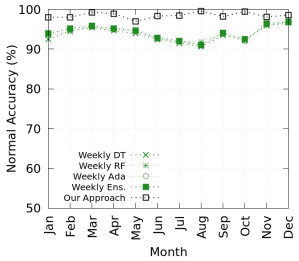
It is important to note that, when compared to their initial accuracy in January (Figure 6), without the verifier module, all evaluated classifiers remained reliable until October. After that period, the TP rate for the *Hoeffding Tree*, was lower than the TP rate for the *Hoeffding Tree* in January, without the



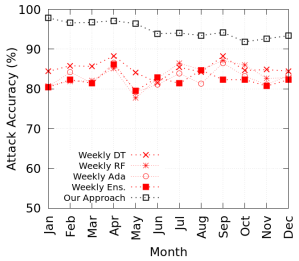
(a) Normal accuracy for our *Ensemble* approach versus no periodic updates



(b) Attack accuracy for our *Ensemble* approach versus no periodic updates



(c) Normal accuracy for our *Ensemble* approach versus weekly updates



(d) Attack accuracy for our *Ensemble* approach versus Weeklyc updates

Figure 11 – Reliable Stream Learning (*Ensemble*) module performance comparison during 2016 in *MAWIFlow* dataset.

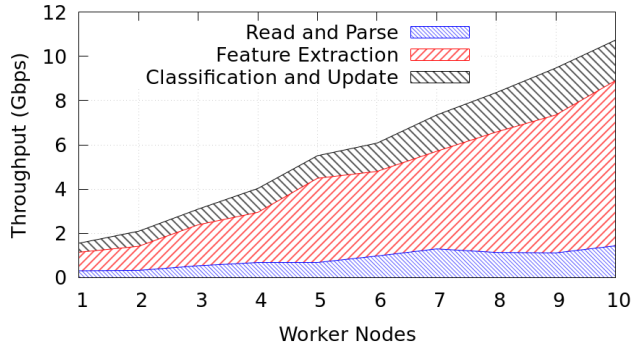


Figure 12 – *BigFlow* throughput performance according to the number of worker nodes.

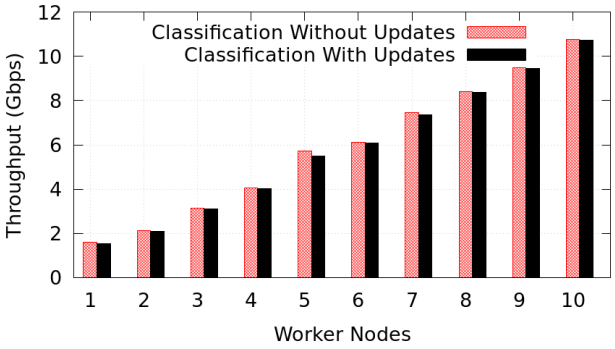


Figure 13 – *BigFlow* throughput comparison with and without updates.

verifier module (Figure 6-a versus Figure 9-a). In this sense, in the worst case, the classifiers can be considered reliable, for at least 10 months, when the verifier module is applied, despite the rejection rate. Finally, it is possible to note that the *Ensemble* classifier, by the means of the proposed $Class_{confidence}$ computation (Equation 1), is able to significantly improve its accuracy when compared to the other classifiers. This occurred because only instances with a high-class confidence value for all classifiers were accepted. Thereby, the event will be rejected even if the majority of the classifiers confidence values are biased, because, if one classifier output a low confidence value, the event will have a low final confidence.

The reliability improvement shown by the verifier module is a desired property for high-speed production networks. In such environments, correct event labels may not be available in a short period of time, e.g., the attack becomes publicly known after three months of its rejection by the verifier module. In such a case, the rejected events would be stored, and their labels would be verified on a regular basis. However, considering the incremental model update would only be possible three months after the attack occurrence, the classifier model would still be reliable during this time.

Finally, Figure 10 shows the classifiers accuracy and rejection rates with the verifier module and a weeklong delay for incremental updates. Several observations can be made regarding the incremental update impact. First, all evaluated classifiers remained reliable throughout the year, considering both TP and TN rates. Moreover, both TP and TN rates significantly increases when compared to only applying the verifier module, without the incremental updates (Figure 9 and 10). The average rejection rate percentage significantly decreases over time, presenting average rejection rates of 13%, 12%, 12%, and 5% in January, and decreasing up to 5%, 4%, 4%, 2% throughout the year, for the *Hoeffding Tree*, *OzaBoosting*, *Leveraging Bag*, and *Ensemble* respectively – in both cases. Nonetheless, it is possible to note that the proposed *Ensemble* approach significantly decreases the rejection rate when compared to the other classifiers, while also presents higher TP and TN rates.

As the results show, *BigFlow* maintains or even improves the model's reliability even when facing new network traffic behavior, regardless of the model being incrementally updated or not. Figure 11 shows a comparison between our proposal, the *no-update* and the *weekly-updated* classifiers shown in Section III.B. The proposed approach greatly improves the detection of attacks for all considered cases, outperforming even the *weekly-updated* classifiers.

Discussion

Although *BigFlow* was able to significantly decrease the average rejection rate by the means of incremental model updates (in the best case from 40 percent to up to 1 percent), the rejection of events in high-speed networks can be challenging. The main challenge refers to the number of events that are going to be rejected, and then, how to process them later.

First, for evaluation purposes, the tests performed previously have operated at 40 percent rejection rate point (Figure 8). For production usage, one will most likely operate at a lower rejection rate operation point, thereby rejecting less instances. Nonetheless, it is important to note that *BigFlow* was able to reject up to only 1 percent of instances, starting at a 40 percent rejection rate. This indicates that, the average rejection rate will significantly decrease over time when compared to the initial chosen operation point. Finally, in production usage, rejected instances will most likely be stored for a period, until its label is publicly known.

Second, the labeling task of such rejected instances can be achieved either by the help of a human expert, normally by collecting more information about a new behavior, e.g., by consulting automatically a public repository of

vulnerabilities/threats such as the common vulnerabilities and exposures (CVE), or by finding that a new type of service is being used in the network.

In this sense, the *BigFlow* rejection rate can be even further decreased. Nonetheless, the labeling process can be made automatically, if a labeling delay can be tolerated for instance.

The most important benefit of our proposal, compared to literature, is to enable the detection that an event cannot be classified accurately and immediately alert the administrator, even if the classifier makes a classification mistake with a high confidence. The action the administrator will perform is under her/his discretion, we only list some in order to show the proposal is feasible. One can be noticed that even a traditional approach, which demands the model rebuilding, a method for event labeling is still required, the main difference is that the output of rejection mechanism is a selective way to do that, facilitating the expert work.

B. Performance, Scalability, and Cost

For evaluating the scalability of our prototype, we set up a 12-node cluster in a single rack, connected through a 10 GbE interface. Each node has a 4-core CPU with 8 GB of memory. In all considered experiments, we set up the *BigFlow* prototype (Figure 5), with the *Ensemble* classifier in the following way: 1 node ran Apache Kafka, 1 node ran the Flink Job Manager and from 1 to 10 nodes ran Flink Task Managers. For throughput evaluation purposes, a set of only 62 features from Table I was considered. For the evaluation purposes, the entire month of January in 2016, was used, and a weeklong delay for the incremental model updates was considered.

Figure 12 shows the throughput and performance breakdown. The throughput performance is divided into *Read and Parse* (*Message Consumer* and *Message Middleware*, in Figure 2), *Feature Extraction* (*Host Aggregator, Flow Aggregator* and *Flow Joiner*, in Figure 2) and *Classification and Update* (*Feature Extraction Module*, in Figure 4). The proposed approach achieved 10.72 Gbps with 10 worker nodes. Regarding its scalability, the proposed approach increased the throughput by 1.02 Gbps for each additional worker node. The *Feature Extraction* module required the most significant part of the overall processing, representing 61% of the processing time on average, while *Classification and Update* together required only 23% on average.

Figure 13 shows the impact of the model's update on the system's throughput. In such a case, the system's throughput performance was divided into *Classification Without Updates* (*BigFlow* without *Rejected Instances Retrieval* and *Incremental Classifier Update* modules) and *Classification With Updates* (*BigFlow*). On average, the model's updates incurred a throughput loss of little more than 1%. Considering the throughput for the cluster of 10 worker nodes, the model's updates incurred a throughput reduction of only 0.25% (0.03 Gbps).

Finally, Table 3 shows the weekly training time and required storage for all the evaluated classifiers shown in Section III.B, considering they would be updated every week. *BigFlow* required (on average) only 4.2% of the storage required by other approaches. Regarding the weekly training time, *BigFlow* required at most 4.2% out of the total time when compared with the complete retraining of DT, RF, GB, ensemble, and Hoeffding tree classifiers.

TABLE III. WEEKLY COMPUTATIONAL AND STORAGE RESOURCES USED BY EACH APPROACH (EXCLUDING INITIAL SETUP)

Approach	Demanded Storage (Gb)			Training Time (hours)		
	Avg.	**Min.**	**Max.**	**Avg.**	**Min.**	**Max.**
Decision Tree	36.41	21.09	43.36	3.91	2.27	4.79
Random Forest				4.40	2.55	5.28
Gradient Boosting				182.7	104.5	213.0
Ensemble				189.0	108.3	224.0
Hoeffding Tree				2.14	1.22	2.58
BigFlow	**1.53**	**0.28**	**5.03**	**0.09**	**0.03**	**0.25**

VII. RELATED WORK

Several existing works address intrusion detection using ML techniques [46] [47] [48]. However, task of model updates has not been, in general, considered in related works [9] [10]. Thereby, the processing and storage costs required for such a task have been discarded [9]. The next subsections further describe related works that address the tasks of building proper *Benchmark Datasets for IDS*, *Flow Measurement and Classification*, and the *Classification Reliability*.

A. Benchmark dataset for IDS

Over the last years, generation of proper datasets for benchmarking intrusion detection systems has been the subject of several studies [31, 33, 37]. However, despite extensive efforts, currently the most used dataset is still the DARPA1998 dataset [33], with several known issues [34, 35]. When a benchmark dataset is built for IDSs, normally some strong assumptions about the training data are adopted [31]. For instance, Canali et al. [36] created their dataset by collecting several website contents from the Internet; they labeled each datum by using state-of-the-art tools and manually inspected the data to ensure proper labeling. In this case, the authors assumed that the most frequently visited websites worldwide are benign, despite several known cases of malicious websites [37]. On the other hand, Shiravi et al. [38] created user profiles on the basis of the user behavior for each application during an observed time interval, while Kendall [39] created a dataset by statistically reproducing the user behavior in an air force environment. In contrast, in UNSW-NB15 [56], the authors rely in a traffic generator tool to create their dataset in a controlled environment. In general, these approaches lack upgradability, wrongly assuming that network traffic is immutable and considering that the user behavior can be modeled [40, 41]. *MAWIFlow* tackles the problem of creating representative datasets by using real and valid network traces, while labeling is achieved using state-of-the-art signature-based detection techniques.

B. Flow Measurement and Classification

Approaches for flow measurement and classification of massive network activities in general rely on pre-stored data. Lee and Lee [5] proposed a Hadoop-based network traffic monitoring and analysis system. The authors performed flow measurements by mapping raw network activity (PCAP) files in HDFS. Their proposed approach achieved 14 Gbps in a cluster of 200 nodes (2 CPU cores each), however, requiring the prior storage of the PCAP files. The authors performed the classification relying on a simple connection threshold through Hive queries, which must be periodically updated in evolving networks. Fortugne et al. [8] focused on integrating several anomaly detectors in the Hadoop architecture for network monitoring. The authors also adopted a similar hash

function approach to divide network traffic in *splits*. Each *split* had an anomaly detection algorithm, which identified network activities based on their anomality scores, according to a specific threshold. However, their approach also required the execution of computationally expensive periodic updates (i.e., full retraining). Moreover, their reported system throughput is unfeasible for high-speed network monitoring.

Some works have applied stream processing techniques for the measurement of massive network activities. Baer et al. [26] proposed a data stream warehouse for network monitoring. The authors also relied on time windows for incremental and continuous execution of queries. Moreover, they combined their proposal with an ML framework for the classification of exported time windows. However, their approach relied on a supervised dataset, without considering the scalability of ML algorithms. They also did not address scalability, reliability, or model updates. A similar approach to *BigFlow* was also taken by Apache Metron [27]. Metron relied on Apache Storm [18] to perform feature extraction in time window intervals. The tool however required the storage of activities in the HBase for post classification, thus requiring also periodic updates. In a recent work, Viegas et al. [28] used a subset of 20 features from [17] to address the resiliency to adversarial attacks in a stream-based intrusion detection system for high-speed networks. *BigFlow* improved their reported throughput by a factor of 9, while extracting 138 additional features and addressing reliability over time through the rejection approach. Finally, a similar labeling process used in the *MAWIFlow* dataset was adopted by Mazel et al. [29] for the evaluation of unsupervised ML algorithms. The authors however extracted a small set of flow-based features and did not evaluate supervised ML schemes. To the best of our knowledge, *BigFlow* is the first approach that does not require the storage of the network activities for neither the feature extraction nor classification, while it is still able to deal with the evolving behavior of networks in the case of high-speed networks.

C. Classification Reliability

Regarding the reliability of classifications in the face of unknown behavior, a verification approach is often applied in other fields in which errors have a high cost, such as OCR [12], medical diagnostics [13], and software fault detection [21], to name a few. For instance, in the field of medical diagnostics, Hanczar and Dougherty [13] employed a verification strategy to reach a desired error rate, while in software fault detection, Mesquita et al. [21] rejected classification outcomes that did not meet a desired degree of certainty. Both approaches rely on an expert to establish the correct event labels. We did not find any work proposing the use of verification strategy for intrusion detection. In contrast, *BigFlow* employs a verification strategy to deal with the evolving network behavior and to reach reliability; moreover, differently from related works, it is able to incorporate the expert assistance into the prior stream learning model, significantly reducing the number of further rejected events.

VIII. Conclusion

Current approaches for network traffic classification are unable to meet the desired throughput; they are also unable to deal with the evolving behavior of network traffic. The approach proposed in this paper, *BigFlow*, aimed at providing high detection throughputs, reliability in face of new network traffic behavior, and a computationally modest model update mechanism.

A high detection throughput was reached by performing feature extraction and classification through stream processing frameworks. The approach used by *BigFlow* groups the exchanged data over the network and summarizes them in time intervals, significantly reducing the required computational effort and storage requirements on the intrusion detector.

To maintain reliability even with evolving network traffic, *BigFlow* employs a verification mechanism, which checks whether the classification outcome should be accepted in order to avoid high confidence in classification mistakes.

Finally, to provide a lightweight update mechanism, *BigFlow* exploits existing stream learning algorithms, incorporating expert assistance for labeling rejected events when they are better understood.

Our experimental evaluation demonstrated that *BigFlow* is feasible for use in production and high-speed networks: our prototype reached up to a 10.72-Gbps throughput in a cluster of 40 cores, being also capable of dealing with evolving network behavior over a year of real network traffic, as evaluated through our *MAWIFlow* dataset, by incrementally updating its detection mechanism with expert assistance.

Acknowledgment

This work was partially sponsored by Coordination for the Improvement of Higher Education Personnel (CAPES), grant 99999.008512/2014-0, by FCT through projects LaSIGE (UID/CEC/00408/2013) and Resilient Supervision and Control in Smart Grids, and by the European Commission through the H2020 grant agreement 700692 (DiSIEM).

References

[1] CISCO. Cisco Visual Networking Index : Global Mobile Data Traffic Forecast Update, 2016 – 2021, 2017, https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf

[2] P802.3cd - Standard for Ethernet Amendment, https://standards.ieee.org/develop/project/802.3cd.html.

[3] DDoS attack that disrupted internet was largest of its kind in history, experts say, https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet.

[4] Symantec, The continued rise of DDoS attacks, http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-continued-rise-of-ddos-attacks.pdf.

[5] Y. Lee, Y. Lee, Toward Scalable Internet Traffic Measurement and Analysis with Hadoop, ACM SIGCOMM Comput. Commun. Rev. 43 (2013) 6–13.

[6] R. Fontugne, J. Mazel, K. Fukuda, Hashdoop: A MapReduce framework for network anomaly detection., INFOCOM Work. (2014) 494–499.

[7] HDFS Architecture Guide, https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[8] R. Fontugne, P. Borgnat, P. Abry, K. Fukuda, MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking, Proc. 6th Int. Conf. - Co-NEXT '10. (2010) 1–12. doi:10.1145/1921168.1921179.

[9] R. Sommer, V. Paxson, Outside the Closed World: On Using Machine Learning for Network Intrusion Detection, 2010 IEEE Symp. Secur. Priv. 0 (2010) 305–316. doi:10.1109/SP.2010.25.

[10] C. Gates, C. Taylor, Challenging the Anomaly Detection Paradigm: A Provocative Discussion, Proc. 2006 Work. New Secur. Paradig. (2007) 21–29. doi:10.1145/1278940.1278945.

[11] P. Borgnat, G. Dewaele, K. Fukuda, P. Abry, K. Cho, Seven years and one day: Sketching the evolution of internet traffic, Proc. - IEEE INFOCOM. (2009) 711–719. doi:10.1109/INFCOM.2009.5061979.

[12] J.R. Navarro-Cerdan, J. Arlandis, R. Llobet, J.C. Perez-Cortes, Batch-adaptive rejection threshold estimation with application to OCR post-processing, Expert Syst. Appl. 42 (2015) 8111–8122. doi:10.1016/j.eswa.2015.06.022.

[13] B. Hanczar, E.R. Dougherty, Classification with reject option in gene expression data, Bioinformatics. 24 (2008) 1889–1895. doi:10.1093/bioinformatics/btn349.

[14] J. Dromard, G. Roudiere, P. Owezarski, Online and Scalable Unsupervised Network Anomaly Detection Method, IEEE Transactions on Network and Services Management 14 (2017) 34–47. doi:10.1109/TNSM.2016.2627340.

[15] N. Williams, S. Zander, G. Armitage, A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification, ACM SIGCOMM Comput. Commun. Rev. 36 (2006) 5. doi:10.1145/1163593.1163596.

[16] A. Moore, D. Zuev, M. Crogan, Discriminators for use in flow-based classification, Queen Mary Westf. Coll. Dep. Comput. Sci. (2005). doi:10.1.1.101.7450.

[17] [1] E. Viegas, A.O. Santin, A. Franca, R. Jasinski, V.A. Pedroni, L.S. Oliveira, Towards an energy-efficient anomaly-based intrusion detection engine for embedded systems, IEEE Trans. Comput. 66 (2017). doi:10.1109/TC.2016.2560839.

[18] Apache Storm, http://storm.apache.org/.

[19] Apache Flink, https://flink.apache.org/.

[20] M.M. Gaber, A. Zaslavsky, S. Krishnaswamy, Mining data streams: A Review, ACM Sigmod Rec. 34 (2005) 18–26. doi:10.1145/1083784.1083789.

[21] D.P.P. Mesquita, L.S. Rocha, J.P.P. Gomes, A.R. Rocha Neto, Classification with reject option for software defect prediction, Appl. Soft Comput. J. 49 (2016) 1085–1093. doi:10.1016/j.asoc.2016.06.023.

[22] MAWI Working Group Traffic Archive, http://mawi.wide.ad.jp/mawi/samplepoint-F/.

[23] Apache Spark MLib, https://spark.apache.org/mllib/.

[24] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (2009) 1263–1284. doi:10.1109/TKDE.2008.239.

[25] MOA – Massive Online Analysis, https://moa.cms.waikato.ac.nz/.

[26] A. Bar, A. Finamore, P. Casas, L. Golab, M. Mellia, Large-scale network traffic monitoring with DBStream, a system for rolling big data analysis, 2014 IEEE Int. Conf. Big Data (Big Data). (2014) 165–170. doi:10.1109/BigData.2014.7004227.

[27] Apache Metron, http://metron.apache.org/.

[28] E. Viegas, A. Santin, N. Neves, A. Bessani, V. Abreu, A Resilient Stream Learning Intrusion Detection Mechanism for Real-time Analysis of Network Traffic, IEEE Glob. Telecommun. Conf. GLOBECOM 2017. (2017). doi:10.1109/GLOCOM.2017.8254495.

[29] J. Mazel, P. Casas, R. Fontugne, K. Fukuda and P. Owezarski. Hunting attacks in the dark: clustering and correlation analysis for unsupervised anomaly detection, Int. J. Netw. Manag. (2014) 17–31. doi.org/10.1002/nem.1903

[30] A. Buczak, E. Guven, A survey of data mining and machine learning methods for cyber security intrusion detection, IEEE Commun. Surv. Tutorials. PP (2015) 1. doi:10.1109/COMST.2015.2494502.

[31] E.K. Viegas, A.O. Santin, L.S. Oliveira, Toward a reliable anomaly-based intrusion detection in real-world environments, Comput. Networks. 127 (2017). doi:10.1016/j.comnet.2017.08.013.

[32] SecureList. IT Threat Evolution Q1 2017 Statistics. https://securelist.com/it-threat-evolution-q1-2017-statistics/78475/

[33] R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, M. a. Zissman, Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation, Proc. DARPA Inf. Surviv. Conf. Expo. DISCEX'00. 2 (2000). doi:10.1109/DISCEX.2000.821506..

[34] S. Brugger, J. Chow, An assessment of the DARPA IDS Evaluation Dataset using Snort, UCDAVIS Dep. Comput. Sci. (2007) 1–19. doi:10.1.1.94.674..

[35] J. McHugh, Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory, ACM Trans. Inf. Syst. Secur. 3 (2000) 262–294. doi:10.1145/382912.382923.

[36] D. Canali, M. Cova, G. Vigna, C. Kruegel, Prophiler : A Fast Filter for the Large-Scale Detection of Malicious Web Pages Categories and Subject Descriptors, Proc. Int. World Wide Web Conf. (2011) 197–206. doi:10.1145/1963405.1963436.

[37] Symantec. Internet Security Threat Report 2017. https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf.

[38] A. Shiravi, H. Shiravi, M. Tavallaee, A. a. Ghorbani, Toward developing a systematic approach to generate benchmark datasets for intrusion detection, Comput. Secur. 31 (2012) 357–374. doi:10.1016/j.cose.2011.12.012.

[39] K. Kendall, A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems, Mater Thesis (1999).

[40] V. Paxson, S. Floyd, Wide-Area Traffic: The Failure of Poisson Modeling, IEEE/ACM Trans. Netw. 3 (1995) 226–244. doi:10.1.1.31.61.

[41] S. Axelsson, The Base-Rate Fallacy and the Difficulty of Intrusion Detection, ACM Trans. Inf. Syst. Secur. 3 (2000) 186–205. doi:10.1145/357830.357849.

[42] R. Quinlan, C4.5: Programs for Machine Learning. San Mateo, CA: Morgan Kaufmann Publishers (1993).

[43] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32. doi:10.1023/A:1010933404324.

[44] J. H. Friedman, Greedy function approximation: A gradient boosting machine, Ann. Statist (2001) 1189–1232.

[45] T.T.T. Nguyen, G. Armitage, A survey of techniques for internet traffic classification using machine learning, Commun. Surv. Tutorials, IEEE. 10 (2008) 56–76. doi:10.1109/SURV.2008.080406.

[46] S. Lee, J. Kim, S. Shin, P. Porras, V. Yegneswaran, Athena: A Framework for Scalable Anomaly Detection in Software-Defined Networks, Proc. - 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Networks, DSN 2017. (2017) 249–260. doi:10.1109/DSN.2017.42.

[47] S.E. Gómez, B.C. Martínez, A.J. Sánchez-Esguevillas, L. Hernández Callejo, Ensemble network traffic classification: Algorithm comparison and novel ensemble scheme proposal, Comput. Networks. 127 (2017) 68–80. doi:10.1016/j.comnet.2017.07.018..

[48] C. Feng, T. Li, D. Chana, Multi-level Anomaly Detection in Industrial Control Systems via Package Signatures and LSTM Networks, Proc. - 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Networks, DSN 2017. (2017) 261–272. doi:10.1109/DSN.2017.34.

[49] Snort – Network Intrusion Detection System. https://www.snort.org/.

[50] Bro – The Bro Network Security Monitor. https://www.bro.org/.

[51] G. Hulten, L. Spencer, P. Domingos, Mining Time-changing Data Streams, Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (2001) 97–106. doi:10.1145/502512.502529.

[52] R. Jordaney, K. Sharad, S.K. Dash, Z. Wang, D. Papini, I. Nouretdinov, L. Cavallaro, Transcend: Detecting Concept Drift in Malware Classification Models, 26th USENIX Secur. Symp. (USENIX Secur. 17). (2017) 625–642.

[53] G. Fumera, F. Roli, G. Giacinto, Reject option with multiple thresholds, Pattern Recognit. 33 (2000) 2099–2101. doi:10.1016/S0031-3203(00)00059-5.

[54] N. Oza and S. Russell, Online bagging and boosting, Proc. Artificial Intelligence and Statistics (2001) 105-112.

[55] A. Bifet, G. Holmes, and B. Pfahringer, Leveraging bagging for evolving data streams, Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics (2010) 135–150.

N. Moustafa, J. Slay, UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set), 2015 Mil. Commun. Inf. Syst. Conf. (2015) 1–6. doi:10.1109/MilCIS.2015.7348942.