EDUARDO KUGLER VIEGAS

TOWARDS RELIABLE INTRUSION DETECTION IN HIGH SPEED NETWORKS

Doctoral dissertation submitted to fulfill a partial requirement for the degree of Doctor in Computer Science in the Graduate Program of Computer Science at Pontifical Catholic University of Paraná, Brazil.

CURITIBA

2018

EDUARDO KUGLER VIEGAS

TOWARDS RELIABLE INTRUSION DETECTION IN HIGH SPEED NETWORKS

Doctoral dissertation submitted to fulfill a partial requirement for the degree of Doctor in Computer Science in the Graduate Program of Computer Science at Pontifical Catholic University of Paraná, Brazil.

Major Concentration Field: *Computer Science*

Advisor: Prof. PhD Altair Olivo Santin

CURITIBA

2018

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós-Graduação em Informática

PUCPR
GRUPO MARISTA

ATA DE SESSÃO PÙBLICA

DEFESA DE TESE DE DOUTORADO Nº 58/2018

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA – PPGIa
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ - PUCPR

Em sessão pública realizada às 09h00 de 02 de Julho de 2018, no Laboratório de Sistemas Distribuídos – Bloco 8 – 2º andar, ocorreu a defesa da tese de doutorado intitulada **"Towards Reliable Intrusion Detection in High Speed Networks"** elaborada pelo aluno **Eduardo Kugler Viegas**, como requisito parcial para a obtenção do título de **Doutor em Informática**, na área de concentração **Ciência da Computação,** perante a banca examinadora composta pelos seguintes membros:

Prof. Dr. Altair Olivo Santin (orientador) - PUCPR

Prof. Dr. Fabrício Enembreck – PUCPR

Prof. Dr. Luiz Eduardo Soares de Oliveira – UFPR

Prof. Dr. Nuno Fuentecilla Maia Ferreira Neves – UNIVERSIDADE DE LISBOA

Prof. Dr. Alysson Bessani – UNIVERSIDADE DE LISBOA

Após a apresentação da tese pelo aluno e correspondente arguição, a banca examinadora emitiu o seguinte parecer sobre a tese:

Membro	Parecer	
Prof. Dr. Altair Olivo Santin	(X) Aprovada	( ) Reprovada
Prof. Dr. Fabrício Enembreck	(X) Aprovada	( ) Reprovada
Prof. Dr. Luiz Eduardo Soares de Oliveira	(X) Aprovada	( ) Reprovada
Prof. Dr. Nuno Fuentecilla Maia Ferreira Neves	(X) Aprovada	( ) Reprovada
Prof. Dr. Alysson Bessani	(X) Aprovada	( ) Reprovada

Portanto, conforme as normas regimentais do PPGIa e da PUCPR, a tese foi considerada:

(X) **APROVADO**

(aprovação condicionada ao atendimento integral das correções e melhorias recomendadas pela banca examinadora, conforme anexo, dentro do prazo regimental)

( ) **REPROVADO**

E, para constar, lavrou-se a presente ata que vai assinada por todos os membros da banca examinadora.
Curitiba, 02 de Julho de 2018.

Prof. Dr. Altair Olivo Santin

Prof. Dr. Fabrício Enembreck

Prof. Dr. Luiz Eduardo Soares de Oliveira

Nuno Neves

Prof. Dr. Nuno Fuentecilla Maia Ferreira Neves

Prof. Dr. Alysson Bessani

We have met the enemy and he is us

Walt Kelly, 1971

Acknowledgments

To my mother and my father, I cannot express my gratitude, this work was only possible because you both have always supported me.

To my supervisor Altair Olivo Santin, who have always supported me since my under graduation, this work would definitely not be possible without him.

To Nuno Neves and Alysson Bessani from University of Lisbon, who have helped me throughout this work and advised me during my stay in Portugal.

To my friends who were always there for me: Vilmar Abreu, Cleverton Vicentini, Rafael Ribeiro, and many more. To my friends: I will always be thankful for getting to know every one of you, and I am sure that our friendship will last for years to come.

To my friends from Portugal who have welcomed me as a family: Vinicius Vielmo, Miguel Garcia, Tulio Alberton and many others. To my friends from Portugal: I am sure we will meet again, and wish the best to all of you.

To everyone from Lasige.

To everyone from SecPLab.

To everyone from PPGIa: Jhonatan Geremias, Cheila Cristina and Nicolas de Paula.

To Pontifical Catholic University of Paraná, for providing me the research grants needed to purse my PhD.

To Intel, for providing me the research grants needed to purse my PhD, and allowing me to improve as a researcher and as a person.

Summary

Figure List

Table List

Abstract

Existing machine learning solutions for network-based intrusion detection cannot maintain their reliability over time in production environments. In such context, detection schemes must be able to detect intrusion attempts at a high network bandwidth, besides having to deal with the lack of realistic training/testing data, changes in network traffic behavior, unreliable classifications over time and adversarial settings. In the light of this, this work introduces a new intrusion detection model, namely reliable intrusion detection, whose main characteristic is the usage of both batch and stream learning algorithms coupled together. The model exploits the characteristics of each type of learner in a cascade pipeline to overcome the challenges of high-speed networks. Batch learning schemes are designed in such a way, that they provide reliable classifications over time and are able to generalize the behavior from the training dataset in the model. On the other hand, is built stream learning schemes resilient to adversarial attacks to hinder attacks over the designed system. Finally, batch and stream learning algorithm are coupled together to provide classification reliability over time, while also reliably adapting to network traffic behavior changes. Experiments over two built datasets showed the model feasibility. In the first dataset, namely fine-grained intrusion dataset, each of the common assumptions adopted in the literature are evaluated and overcome by the reliable intrusion detection model. In the second, namely MAWIFlow, was built an intrusion dataset of over 30 TB of data that spans for over 10 years of real production environment networks. Experiments in MAWIFlow also showed the model feasibility regarding its scalability, and reliability over time, even in the absence of model updates.

Keywords: *Machine Learning; Stream Learning; Intrusion Detection; Classification Reliability; Intrusion Databases.*

Abbreviations List

CNN	Cable News Network
DARPA	Defense Advanced Research Project Agency
DNS	Domain Name System
GB	Gigabyte
HDFS	Hadoop Filesystem
HIDS	Host-based Intrusion Detection System
IDS	Intrusion Detection System
KDD99	Dataset used for the third KDD competition
MAWI	Packet traces from WIDE backbone
MAWILAB	Related work that provides daily labeling of network traces from MAWI
ML	Machine Learning
NIDS	Network-based Intrusion Detection System
OZA	Ozaboosting
PCAP	Packet Capture
RF	Random Forest
TB	Terabyte
US	United States

Chapter 1

Introduction

According to a CISCO network forecast report, the worldwide network traffic in 2016 was 96 EB/month, and, it 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 400 Gbps in a near future [2]. Moreover, current network-based cyber-attacks are also significantly increasing their capabilities. For instance, in October 2016, a DDoS attack with 100 thousand malicious endpoints surpassed a bandwidth of 1.2 Tbps at a Domain Name Server (DNS) infrastructure. This sort of attack could 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]. Thereby, operators need access to solutions to enable the real-time measurement and analysis of such malicious content over those massive network attacks.

To this end, over the last years, several works have applied machine learning (ML) techniques, mostly through pattern recognition schemes, for the detection of network-based attacks. In a pattern recognition scheme, the classification of intrusion attempts is, in general, achieved through a two-phase process: *training* and *testing* [5]. In the training phase, the classifier learns the environment behavior, as present in the training dataset, producing a model. Afterwards, in the testing phase, the model is evaluated regarding its accuracy using a test dataset, which is expected to represent the production environment behavior [6].

However, on the other side, the network traffic behavior changes in a daily-basis, either due to the discovery of new attacks [6], or due to the offering of new services [7]. In such context, due to the evolving behavior of such environment [8], and the high network throughput

[1], the identification of network attacks becomes a challenging task, in which a designed detection mechanism can become out-of-date before they are even deployed in real-world (production) environments [9].

1.1 *Motivation and Hypothesis*

Network-based intrusion detection through ML schemes have been the subject of several studies over the last years [6]. In such context, in their majority, they have aimed at improving the detection rate in a single dataset – more specifically KDD99 [10]. However, few research has been made regarding the applicability of their work [7]. Thereby, despite the promising reported results, such as high detection rates, there still a lack of adoption of such systems in production environments [6]. Thus, it remains mostly as a research topic [6], being the signature-based approach (event scan for well-known attack patterns) currently the most used technique [11].

The network-based intrusion detection field presents several challenges to ML techniques when compared to other fields [10]. Thereby, when a new ML-based approach is under development it must undergo through a more comprehensive evaluation. However, in general, the majority of works employs a traditional evaluation approach [7], in which the accuracy rates measured in a single test dataset are assumed to be evidenced in production [6], despite the challenges that networked environments present. In such a case, a ML-based scheme must be able to detect intrusion attempts at a ***high network bandwidth***, besides having to deal with the ***lack of realistic training/testing data, not generalization capable models***, ***changes in network behavior***, ***unreliable classifications over time*****,** and ***adversarial attack setting.***

Network throughput has significantly increased in the past years. However, current approaches for network traffic measurement and analysis in the Big Data context often rely on Hadoop-based clusters [12, 13]. In general, current solutions write network packets as raw data (PCAP) to a distributed filesystem (e.g. HDFS [14]) and, process them later on. Although such approaches offer significant improvements in scalability [12], they lack applicability in real-world environments. Nonetheless, current methods for the discovery of new network attacks in such context are mostly achieved through unsupervised machine learning techniques, which typically also require the storage of the network traffic over a period for the identification of unknown anomalies [15]. However, due to the massive amount of network data generation, its storage for further analysis is not feasible in most scenarios [16]. In such settings, the network traffic must be analyzed at line speed, allowing intrusion detection to be performed without delays.

On the other side, when one is training a ML-based intrusion detection technique, the lack of realistic training and testing data becomes a challenging task. To this end, several approaches have been proposed for the obtainment of properly built intrusion datasets [10]. Typically, such proposals are either performed in a controlled environment [17] or by the means of the production environment monitoring [15]. In the prior, the generated dataset often lacks the production environment highly variable nature [10]. While in the former, it lacks a ground truth (prior correct event labels), or its sharing is not possible because of privacy concerns [18]. Thus, despite several efforts, currently, one of the most used datasets was built in 1998 [19], with several known flaws [20, 21].

Thereby, due to the lack of realistic training/testing data, the building and evaluation of ML-based techniques regarding their generalization capacity is often not made [7]. This because, intrusion detection schemes must be able to generalize the behavior from the training environment to other scenarios, due to the highly variable nature of production environments, as it is not possible to train a classifier with all possible normal or attack behaviors, thus demanding that the detection scheme is generalizable-capable. However, current approaches in the literature, in general, assumes that the accuracy obtained in the testing dataset, will be observed during production usage [6]. Although, even an updated classifier, will have to deal with changes in both normal and attacker behaviors. For instance, a normal behavior might change due to the offering of new services or new contents. On the other hand, the attacker behavior, might also change due to the discovery of new attacks, or by even changing how an attack behaves [8]. Thereby, assemble a training and testing dataset that present all such properties is not feasible [6, 7].

However, regardless of whether the used training/testing dataset was properly built, and the detection scheme is generalizable or not, the network traffic also changes over time, either due to new types of malicious actions or due to alterations in the transmitted content (e.g. due to the offering of new services), rendering the obtained model outdated [9]. Consequently, the accuracy that was achieved during training might not be observed in practice, even if the dataset was properly obtained. In such cases, the intrusion detection engine will no longer be trusted by the operator, as alarms are not generated as expected [22]. However, the identification of changes in network behavior is a challenging task, which often requires human intervention to reevaluate whether the current model error rate is still acceptable or not. Thus, to achieve classification reliability, the model must be periodically tested and updated (e.g. every month). This action requires human intervention not only to rebuild the model, which takes time and storage, but also to keep the production model operational, until a new model is available, with

acceptable error rates.

On the other side, despite the need for model updates, intrusion detection schemes operate in the settings of an adversary. In such context, a sophisticated attacker may attempt to evade the intrusion detection mechanism, either by perverting the intrusion detection mechanism properties or by injecting attacks during the training stage [23].

Provide classification reliability in the presence of new network behavior and in the settings of an adversary is a challenging task. Current approaches often rely in the classification confidence given by the classifier [24]. In such cases, events with low confidence are rejected rather than being potentially misclassified. However, as such probabilities are computed according to the behavior present in the training dataset, when an event with an unknown behavior is classified its probabilities can be biased [24]. For instance, a new attack that does not behave similarly to known attacks may bias the classifier confidence values, wrongly increasing other classes confidence.

Such challenging scenario for ML-based intrusion detection is hardly noted in the literature [6, 7]. In contrast, the majority of works aim at improving the system detection rates, rather than actually questioning if their detection mechanism could actually be used in production, and how reliable their generated alarms will be over time. In the light of this, this work hypothesis is that: ***batch and stream learning schemes coped together are able to provide a more reliable network-based intrusion detection in real-world production environments over time***.

This hypothesis depends on the building of a reliable generalization capable batch learning detection scheme. In other words, it depends on a batch learning scheme, able to generalize the production environment behavior, regarding the classification of similar and new event's behavior, in a period of time, due to the difficulties involved in creating representative training intrusion datasets. Nonetheless, when new/unseen events are classified after such period, classifications must still be reliable, in other words, potentially misclassifications must be rejected rather than wrongly classified. On the other side, in order to reliably adapt to network behavior changes over time, the hypothesis also depends on the building of a stream learning detection scheme resilient to adversarial attacks. Thereby, the proposed stream learning detection scheme must be able to adapt to the network changes over time, while also not being susceptible to updates with mislabeled instances caused by an adversary. Finally, this work hypothesizes that a generalizable capable batch learning scheme, able to make reliable classifications over time, coped with a stream learning scheme, in a cascade manner, that is, resilient to adversarial attacks, and able to adapt to changes in network behavior over time, will

be reliable in production use over time.

1.2 *Goals*

The main goal of this work is to propose, develop and evaluate both batch and stream learning detection schemes, with the aim to provide reliable network-based intrusion detection in real-world production environments over time.

To reach the main goal, the following specific objectives must be met:

I. To conceive a set of methods that addresses the main problems reported in the literature when building datasets for IDS evaluation, while also enabling the fine-grained (production environment characteristics) and overtime IDS evaluation.
II. To conceive a set of methods that evaluates batch learning and stream learning intrusion detection mechanisms considering the network properties from production environments;
III. To conceive a method to generate generalization capable batch learning intrusion detection schemes;
IV. To conceive a method to generate reliable batch learning intrusion detection schemes, in the presence of unknown behaviors;
V. To conceive a method to generate stream learning intrusion detection schemes resilient to adversarial attacks;
VI. To conceive a method to assess the classification reliability even in the presence of new network behaviors;
VII. To conceive a method to enable reliable ML-based intrusion detection over time in the presence of new network behaviors;
VIII. To conceive a method to enable reliable ML-based intrusion detection in high-speed networks;

Therefore, objectives I and II aim at providing properly built intrusion detection datasets in order to evaluate the proposal. Objectives III and IV is related to provide a method for the proper evaluation of ML-based intrusion detection techniques. Objectives V and VI concerns the building of reliable and generalization capable batch learning intrusion detection schemes. Objective VII aim at providing stream learning intrusion detection schemes resilient to adversarial attacks. Objectives VIII and IX addresses the design of a detection scheme to provide a reliable intrusion detection scheme over time. Finally, objective X aim at enabling the proposed reliable intrusion detection for high-speed networks.

1.3 *Contributions*

This work provides means to enable the use of ML-based intrusion detection techniques in production environments in a reliable manner. In summary, this work presents the following main contributions:

I. An approach named *BigFlow*, which performs reliable and near real-time network traffic measurement (feature extraction) and classification in the Big Data context;

II. A tool-based method that produces real and valid network traffic in a controlled and reproducible environment for creating intrusion datasets. The datasets built through such method aim at evaluating both batch and stream learning intrusion detection schemes;

III. An intrusion dataset with real and labeled network traffic, based on MALAWI database, built by analyzing over 10 years of real network traces, composed by more than 30 TB of data and 30 billion network flows. The built dataset aims at evaluating stream learning intrusion detection schemes;

IV. A new and fine-grained evaluation method specific for batch learning intrusion detection schemes;

V. A new multi-objective feature selection method aiming to improve the generalization capacity of batch learning schemes, by considering the network properties in intrusion detection;

VI. A new rejection method that provides classification reliability even when facing unknown network traffic behavior;

VII. A new approach to reliably deal with evolving network data streams to perform anomaly-based intrusion detection, in the presence of an adversary;

VIII. A new classification reliability assessment method through a conformal evaluator module. It aims at providing a reliability degree while facing new network traffic behavior even in the absence of model updates. The conformal evaluator assesses the classifier confidence according to the behavior seen in the training dataset;

IX. A new reliable intrusion detection mechanism made of both batch and stream learning algorithms, providing classification reliability and ongoing updated classification models with minimal human assistance.

1.4 *Document Structure*

The remainder of this document is organized as follows. Chapter 2 describes the challenges that network-based intrusion detection faces towards reliable use in production

environments. In summary, big data processing techniques for network flow measurement and analysis, and the challenges that ML-based intrusion detection techniques faces are presented. Chapter 3 presents how related works addresses such challenges. Chapter 4 describes the model proposed in this work, namely reliable intrusion detection model. Chapter 5 tackles the challenge of building realistic intrusion datasets for the evaluation of network-based intrusion detection schemes. Chapter 6 reports the experiments performed over the reliable intrusion detection model using the built datasets. Finally, Chapter 7 concludes this work and presents future works.

Chapter 2

The Challenges of Network-based Intrusion Detection

This chapter objective is to present an overview regarding the techniques used throughout this document. More specifically, Section 2.1 presents big data processing techniques that are commonly applied for network measurement and analysis in high-speed networks. In contrast, Section 2.2 further describes intrusion detection techniques, while Section 2.3 addresses machine learning techniques applied to such context, and the challenges it faces towards reliable production usage.

2.1 *Big data processing*

The Big data concept is often referred in the literature when the amount of data cannot be processed by the means of traditional processing techniques [25]. To this end, its characteristics can often be summarized in four 'V's, being them [26]: (i) *Volume*: due to the increasing quantity of data; (ii) *Variety*: due to the large number of data sources and their types, which also includes non-structure data such as: text, video, audio and web, besides the traditional structured data; (iii) *Velocity*: due to the speed that such data is generated and the processing demanded by it; and (iv) *Value*: due to the cost involved for the collection, storage, and processing of such data.

For instance, consider an intrusion detection system operating in a gigabit network. In such context, a significant data *volume* is generated constantly, according to how the network entities communicate over time, moreover its *velocity* is strictly related to the network speed link. On the other side, the *variety* of such data may include: sources (hosts), network protocols, network attacks, services, amongst others. Finally, the *value* of such data is strictly related to

the environment it is operating, because if the intrusion detector fails at detecting network attacks, in a short period of time, the monitored environment may suffer the attack consequences.

In such context, to deal with these characteristics, a proper distributed processing system must be developed. To this end, in general, Big data processing techniques often falls in two categories: *batch* or *stream*. The next subsections briefly describe each in detail.

2.1.1. Batch processing

The main characteristic of batch processing techniques is the need to store previously the data for its processing [26]. An example of a framework that enables this kind of data processing is the Apache Hadoop [27]. Hadoop project have two main characteristics: its (i) *Distributed Filesystem* (*Hadoop Distributed Filesystem – HDFS* [14]), which is responsible to perform the data fragmentation, spreading and replication between the cluster nodes, and the (i) *MapReduce* programming model [28], which enables the processing division between several nodes, each with smaller sizes.

Although batch processing techniques are feasible for big data processing, they are not adequate for situations that demands real-time processing, e.g. real-time detection of network attacks. Such factor is mainly caused due to the need of data storage. Thereby, to enable real-time processing, stream processing techniques must be used.

2.1.2. Stream processing

In contrast to batch processing, stream processing techniques must be able to process large volumes of data in near real-time [26]. To this end, stream processing platforms (e.g. Apache Storm [29], and Apache Flink [30]) receive data from registered sources, and compute over such data through a set of Processing Elements (PE). Each PE is responsible to perform a specific operation on the arriving data and to send the result to another PE, until the computation ends. 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. On the other side, 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 in such platforms is achieved by keeping the computation in each PE type as small as possible, and by distributing the message load uniformly through several PE in parallel.

2.2 *Intrusion Detection*

An intrusion detection system (IDS) is a tool that is aimed to identify and classify malicious activities in an environment [31]. Figure 1 shows a typical IDS architecture composed of four modules. (i) *Data Acquisition*: this module is responsible for the collection of events from the monitored environment for further analysis, e.g., read network packets from a network interface card. (ii) *Preprocessing*: this module performs the processing needed before the detection engine is run on the collected events, e.g., extract a set of features. (iii) *Detection*: this module, based on the preprocessed event, decides whether an event is normal or an intrusion attempt. (iv) *Alert*: finally, if an event is considered as an intrusion attempt, this module generates an alert.

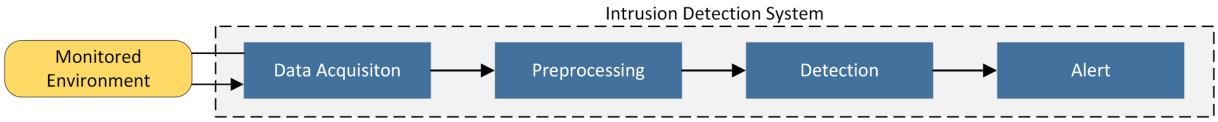


Figure 1. Typical intrusion detection architecture.

Several approaches for the classification of intrusion attempts have appeared in the literature. Currently, the most frequently used taxonomy, which is used in DARPA's intrusion datasets, was defined by Kendall [19]. According to Kendall, intrusions can be divided in four classes as follows. (i) *Probing* attacks that gather information about a target; (ii) *Denial-of-Service* (DoS) attacks, that is, any attempt to prevent legitimate users from accessing a service or a system; (iii) *remote to local* (R2L) attacks that attempt to gain access to a legitimate user's account on a system; and (iv) *user to root* (U2R) attacks, which occur when the attacker already has achieved access to a legitimate user's account and then attempts to obtain administrator privileges.

An IDS can be either network-based (NIDS) or host-based (HIDS), which determines the attack classes it is able to detect. An NIDS performs the detection at network level, detecting attacks such as probing (e.g. *portscan*) and a network-based DoS attacks (e.g., *synflood*). To this end, to perform the event classification, an NIDS accesses the data only at network level, e.g., network packets or network flow. In contrast, an HIDS detects application-based attacks, such as R2L (e.g., *buffer overflow*) and U2R (e.g., *privilege escalation*). Thereby, it needs to have access over the application data running on the protected systems, thus requiring access to logs and other data about the system in order to perform its detection.

In general, two approaches stand out in the literature for the detection of intrusions: signature and anomaly. The signature-based approach consists in searching well-known attack patterns in the received events. To this end, each event must be matched against the signature

database; if a known attack pattern is found, the event is classified as an intrusion attempt. Thereby, when a new vulnerability is discovered and reported, a related signature must be created. The main drawback of the signature-based approach is its inability to detect new attacks.

In contrast, the anomaly-based approach is aimed to detect new (unknown) attacks by modeling the activities that are considered normal within a system. While the identification of attacks is reached by identifying behaviors that deviate from the known normal modeled behavior pattern [6]. The event behavior is defined by a set of features extracted during the preprocessing stage. In studies in the literature, anomaly-based intrusion detection is in general treated as a pattern-recognition problem [7], by the means of machine learning techniques.

2.2.1. Machine Learning (ML)

Machine learning aim to assign a class (e.g., normal or attack/intrusion) to an event [31]. In the process, the events are captured from the environment and stored in a database. From each event in the database, a set of features is extracted and stored in a dataset. A machine learning algorithm is then used to infer a pattern from the dataset and to create a model that represents such behavior. However, events that present a behavior similar to that of other classes may be wrongly classified, e.g., an attack that is similar to a legitimate request.

Several approaches can be used to perform the classification task by the means of ML techniques. However, in general, an intrusion model is created using a training dataset. While for the estimation of the classifier accuracy rates, a validation dataset is used. The validation dataset is utilized for making possible improvements to the intrusion model. Finally, the final version of the intrusion model is evaluated through a test dataset. For the process to be reliable, each used dataset must contain different events. During the tests, the false-positive (FP) and false-negative (FN) rates are estimated. An FP occurs when a legitimate activity is classified as an intrusion, whereas an FN occurs when an intrusion is classified as a legitimate activity.

For the purpose of this dissertation, two ML approaches, vastly used in the literature [31] and shown in Figure 2, will be considered for intrusion detection: Batch Learning, and Stream Learning.

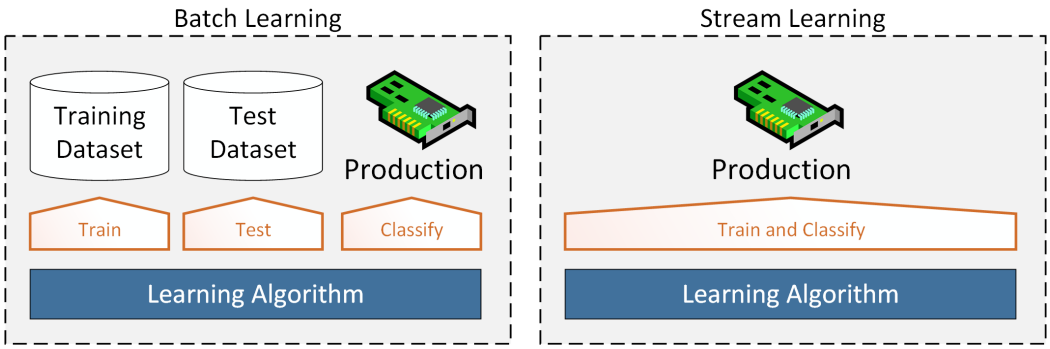


Figure 2. Batch and Stream Learning applied to NIDS.

Batch Learning (Figure 2, Batch Learning) techniques obtains a model by the means of a training dataset, in a process called training, and evaluates it using a test dataset. Afterwards, the model can be used in production for the classification of further events. In this sense, if the environment behavior changes, a new model must be built. To this end, the current model is discarded, and a new training and test dataset is assembled, only then, a new model can be obtained.

Unfortunately, general-purpose networks rarely present a stable traffic pattern [8, 9]. In such context, a typical approach to deal with evolving environments is to resort to stream learning algorithms [32] (Figure 2, Stream Learning). These techniques allow the detection mechanism update to be performed at each new event arrival, in an incremental way, without discarding the current model. Thus, the time needed for building an updated classifier model can be decreased [32]. However, these techniques typically rely on supervised learning, in which the events need to be previously classified. Moreover, it is necessary to devise a method to select the events that should be used for the incremental model update [33]. Thereby, rendering the current approaches not easily applicable to networked environments [7].

Ideally, the model used in production environments should be as up-to-date as possible, i.e. updated at each new arrived event. However, updates, in either Batch or Stream Learning, are often prone to human assistance because in production environments the event`s label are unknown. This process, in the event labeling task, is typically known as Active Learning. In contrast to Supervised Learning, in which all event labels are known, it assumes that a subset of event's label can be requested to an expert over time [33]. The main goal is to improve the model performance, by providing a subset of the true event's label in production.

2.2.2. Challenges for ML-based Approaches for Production Usage

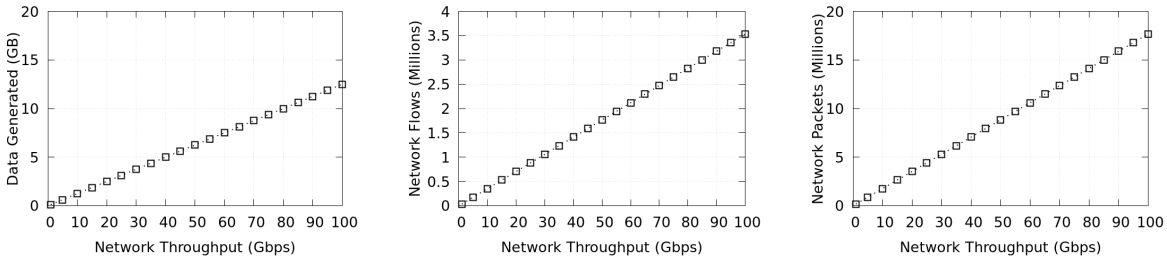
Despite extensive efforts made towards ML-based for intrusion detection in the last years, which have yield promising results regarding the system's accuracy rates, they are hardly

deployed in production environments [6]. The next subsections further describe the challenges that ML-based techniques face towards reliable use in production environments.

2.2.2.1 High network bandwidth

Network bandwidth has significantly increased in the past years. Thereby, increasing the rate of events to be classified, generated data and the number of network packets. In such context, several ML-based techniques have been proposed, however, only few authors have addressed the detection system throughput.

Figure 3 shows an example of how the network bandwidth affects the system received data in a real scenario. In such a case, if a network throughput of 100 Gbps is considered, the ML-based NIDS would need to classify up to 3.53 million flows/sec. On the other side, the preprocessing module (Feature Extraction) would need to process up-to 17.66 million packets/sec. Finally, if the used ML technique requires the data storage, as occurs in unsupervised learning for instance, a 12.5 GB/sec of storage would be needed.



(a) Data Generated (b) Network Flows (c) Network Packets

Figure 3. Network throughput relation to number of flows, network packets and generated data. Rates were computed according to MAWI [16] network traffic in 02.21.2018.

In such settings, due to the amount of generated data, the use of unsupervised learning techniques becomes unfeasible. Moreover, to improve or update the model over time by the means of active learning, is also not easily feasible. For instance, if 0.1 percent of event's true labels were requested, an expert would need to manually process up to 35 thousand events/sec. Thereby, in such context, one must resort to supervised learning techniques, either batch or stream based.

In general, supervised batch learning approaches are scalable, thereby able to cope with such amount of data [34]. However, due to the changes in network behavior, the model will quickly become out-of-date, i.e. unreliable classifications will occur. On the other side, supervised stream learning techniques also are scalable [35], but, in addition, enable incremental model updates over time. However, due to its evolving nature, they are prone to

adversarial attacks (attacks targeted at the model) and also requires event's true label over time for incremental model updates [32, 52].

Regardless of the used ML technique, the feature extraction process is still required. To this end, flow features are extracted by analyzing the exchanged network packets over time according to the data exchanged between the entities over the network, e.g. number of bytes sent/received between two hosts in a period of time. Thereby, because it requires to store network statistics for a period of time, such task becomes a computationally-expensive process [5]. Surprisingly, current approaches for the flow rebuilding (feature extraction) in the Big Data context often rely on Hadoop-based clusters [12, 13]. These approaches, in general, simply writes the raw network traffic activity log (PCAP) to the filesystem (e.g. HDFS) for further analysis. Although such approaches offer significant improvements in throughput, they lack applicability in real-world environments. In such settings, because the storage of network data is not feasible, the traffic must be analyzed at line-speed.

2.2.2.2 High variability in input data

Network traffic is prone to a high variability in a short period of time. For instance, within a day, a network link may crash, several different attacks may occur, new services may be provided, new service's content may be requested, amongst others.

On the other side, train and test datasets should present all the expected event's behaviors from production environments [10]. However, in such context, obtain all possible events is not possible. Thus, ML-based intrusion detection techniques must be able to generalize their known behaviors. In other words, the model, with a small subset of events, must be able to correctly classify new ones, with a similar or new behavior. However, the design of models capable of generalizing the environment behavior is a challenging task. Because, to this end, a series of evaluations must be performed to measure how the model will behave in each possible network setting.

For instance, consider a model trained only with *synflood* attacks. In production usage, it may need to classify other type of DoS attacks, such as: *icmpflood*, *udpflood*, *slowloris*, *htmlflood*, amongst others. Nonetheless, such attacks may vary their frequency and rate over time. In such a case, to measure the generalization capacity, the evaluation process must measure the accuracy rates according to each setting. In other words, measure the accuracy rates for each attack, considering different attack frequency and rates. The same occurs for normal behaviors, in which a normal event may change the requested service, content, frequency and rates.

2.2.2.3 Lack of realistic training/testing data

Intrusion detection community suffers from a lack of proper designed datasets. To this end, several techniques have been proposed in the literature [17, 22, 36]. However, the most frequently used dataset [31] remains the well-known DARPA1998 dataset [19], which is now almost 20 years-old. Most approaches used to create a public intrusion dataset attempted to statistically model the user behavior [17, 19, 36]. In general, a typical and real user is monitored during a certain period and its traffic characteristics are reproduced in a statistically similar manner. Thus, a static user behavior is imposed during the monitored period. Nonetheless, these approaches generate a site-specific traffic behavior that are difficult to reproduce.

The expected properties for an intrusion dataset are as follows [36]. (i) *Realism*: the dataset should contain network traffic that can be observed in production environments; (ii) *validity*: the dataset should contain well-formed packets, with a complete client-server interaction; (iii) *prior labeling*: in the dataset, each event must be correctly labeled (as belonging to a class, e.g., normal or attack), to allow correct classifier training; (iv) *diverse/high variability*: the dataset should present a diversity of services, client behaviors, and attacks; (v) *correct implementation*: in the dataset, the attacks must follow a well-known or "de facto" standard; (vi) *ease of updating*: the dataset should be able to incorporate new services and attacks that are discovered every day; (vii) *reproducibility*: the dataset should allow experts to perform a comparison between datasets; and, finally, (viii) *without sensitive data*: the dataset should not reveal sensitive data to allow the free dataset to be shared among researchers

Two approaches may be used for obtaining datasets for NIDS building: in the first, the production environment is recorded and in the second a controlled environment is created [22]. The production environment monitoring allows traffic that is real and similar to the environment to be obtained. However, because of privacy concerns it is not feasible to share the dataset [18]. On the other side, the creation of a database in a controlled environment using tools allows it to be shared freely, however, the approach suffers from traffic invariability problems [36]. In this sense, although several approaches have been proposed, the anomaly-based IDS literature lacks a ground-truth dataset.

2.2.2.4 Changes in network behavior

General-purpose network environments rarely present a stable traffic pattern. On the contrary, the set of target concepts (e.g., network traffic classes) learned during the training stage often evolve over time [6]. For instance, the network behavior may change because new services are added [7] or due to modifications on how the attacks are executed.

Nonetheless, the identification of such changes in high-speed production networks can be challenging. Thereby, in order to cope with them, the behavior model needs periodic updates. This typically involve a computationally-demanding task of model rebuilding, which can only be performed if there is access (storage) over the recent observed traffic and the prior (manual) classification of the events. In addition, the model rebuild cannot be postponed, as 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 [7]. This makes the process challenging for high-speed networks.

2.2.2.5 Unreliable classifications over time

Changes in network behavior, lack of realistic training/testing data, high variability in input data and lack of model updates will inevitably render unreliable classifications over time. In such a case, classifications made by the model will no longer be trusted by the operator.

A common approach to assess classification reliability often relies in the classification confidence given by the classifier [37, 38]. To this end, events with low confidence are rejected rather than being potentially misclassified. However, as such probabilities are computed according to the behavior present in the training dataset, when an event with an unknown behavior is classified its probabilities can be biased [24]. For instance, a new attack that does not behave similarly as known attacks, thus increasing its normal class confidence output by a classifier.

2.2.2.6 Adversarial settings

An emerging field of research, known as adversarial machine learning [23] considers the use of machine learning in the settings of an adversary – called adversarial settings. In such cases, the adversary (attacker) will attempt to evade the intrusion detection mechanism using sophisticated types of attacks, called causative and exploratory [39]. The causative attacks refer to attacks that occurs during the training process, e.g. the attacker inject misclassified intrusions into the training dataset as normal events. On the other hand, the exploratory attacks aim at exploring the machine learning algorithm properties, e.g. craft the intrusion attempt in a manner that the detection engine classifies it as a normal activity.

Chapter 3

Related Works

This chapter presents the related works in five main areas according to the challenges faced by ML-based intrusion detection techniques. Section 3.1 describes the related works towards high detection throughput in intrusion detection. Section 3.2 addresses how the literature deals with the lack of realistic training/testing data in intrusion detection, and whether those schemes are generalizable capable or not. Section 3.3 describes how related works addresses adversarial attacks to detection schemes. Section 3.4 shows the related works dealing with unreliable classifications over time. Section 3.5 address the related works regarding reliable intrusion detection over time.

3.1 *High detection throughput in intrusion detection*

This subsection further describes related works to the proposed solution for the network measurement and analysis of massive network activities, named *BigFlow*.

Approaches for flow measurement and classification of massive network activities in general relies in the prior storage of network data. For instance, Lee and Lee [12] proposed one of the first scalable internet traffic measurement approach in the literature. To this end, the authors developed a Hadoop-based network traffic monitoring and analysis system. In their work, they performed the flow measurement by mapping raw network activity (PCAP) files in HDFS. To enable such mapping, the authors have developed an API that is able to interpret PCAP files format, break such files into blocks, while not losing network packets during such process. Their proposed approach is scalable and achieved 14 Gbps in a 200-node (2 CPU core each) cluster, however, they required the prior storage of the PCAP files. In their work, the

authors extracted several feature sets according to each protocol layer (e.g. transport, application, amongst other). In their evaluation tests, the feature extraction throughput was significantly degraded according to the extracted set of features. The authors performed the classification by the means of a simple connection threshold through Hive queries, which must be periodically updated in evolving networks.

Since then, several works have also proposed more comprehensive and scalable network traffic classification approaches. For instance, Fortugne et al. [13] focused in integrating several anomaly-detectors in the Hadoop architecture for network monitoring. In their work, two approaches for network traffic measurement were proposed: *packet-count* and *astute*. In the prior, only a packet count according to the hosts sending them was extracted, while in the former 6 features according to several network packet values were extracted. The authors adopted a hash function approach to divide network traffic in *splits*. Each *split* had an anomaly-detection algorithm, which identifies network activities by their anomalous score according a specific threshold. Although their approach was implemented in Apache Hadoop, according to the evaluation tests, it lacks scalability when *astute* feature extraction was considered. Nonetheless, their approach also required the execution of computationally-expensive periodic updates (i.e., full retraining).

In contrast, some works have applied stream processing techniques for the measurement of massive network activities. Baer et al [40] was one of the first authors to address network traffic measurement and classification through stream processing techniques. In their work, they proposed a Data Stream Warehouse (DSW) for network monitoring. To this end, the authors relied in time windows for incremental and continuous queries execution, similarly to *BigFlow*. However, to achieve such goal, the authors defined a declarative language interface based on SQL. The declarative language interface support was enabled by building their prototype on top of PostgreSQL, thereby requiring changes over such framework. Moreover, they integrated their proposal with a machine learning framework for the classification of the exported time windows, building their prototype on top of Weka API. However, their approach relied on a supervised dataset, without considering the scalability of the machine learning algorithms. They also did not address scalability, reliability, nor model updates.

Vernon and Victor [41] have also addressed network flow measurement by the means of stream processing techniques. In their work, the authors developed an Apache Storm topology that digest netflow records through an AMPQ queue. When a record is read, its field values are interpreted, and flow statistics values are computed. Similarly to Baer et al [40], their

work also rely on a SQL-like language for further processing. Finally, when a flow is computed, they rely on HBase for the flow storage and Apache Hadoop for the flow processing. In their evaluation tests the authors have shown that their proposal is scalable, achieving up to 211 MB/sec in a 10-node cluster. Although the authors addressed scalability, and stream-based processing, their proposal still required the storage of data for the network flow classification.

A similar approach to *BigFlow* is also performed by Apache Metron [42]. Metron relies on Apache Storm [29] to perform the feature extraction in time window intervals. Similarly to *BigFlow*, Apache Metron is executed as a topology within Apache Storm framework. To perform the feature extraction, it also relies in time window intervals. In their approach, the user defines the set of features that are going to be extracted by the means of a JSON file. For each feature, the user can define the event field that is going to be summarized, and the related processing, e.g. the sum of events with the same IP source address. The tool however requires the storage of the activities in HBase for post classification. Moreover, the classification of network attacks is only addressed by the means of unsupervised techniques, hindering its usage in production environment.

Finally, in a prior work [43] a *BigFlow* prototype was implemented, using a subset of 20 features from [5]. The prototype goal was to address the resiliency to adversarial attacks in a stream-based intrusion detection system for high-speed networks. The classification process was achieved by the means of multi-view learning with a forest of *hoeffding tree* classifiers [44]. In the evaluation tests, the preliminary *BigFlow* version showed the scalability and feasibility of the proposal.

3.1.1. Discussion

A number of works addresses the flow measurement of massive network activities. In its beginnings, several approaches have been proposed to this task by the means of batch processing techniques [12, 13]. However, such approaches are unable to be used in production environments, mainly due to the amount of data being generated over time. In the light of this, several works have proposed the use of stream processing techniques to fulfill this task [40, 41, 43]. In general, those approaches divide the read network traffic into splits. Each split is then processed according to a time-interval. Although such techniques are scalable and performs the network traffic measurement in a stream-based manner, they either require framework modifications [40] or extracts a non-representative subset of features [40, 41].

Regarding the classification of massive network activities, such task is often not given the proper care. In most cases, the classification is only achieved when such data is previously stored [12, 13, 40, 41, 42]. Even so, they either rely in simple feature thresholds [12], unsupervised machine learning [12, 13, 42] (in which the computation to fulfill such task is discarded by the authors), or supervised ML (in which the labeling process is not addressed). Nonetheless, there are no works that address the classification reliability over time.

Table 1 shows a summary of related works comparison to *BigFlow*.

3.2 *On Building Realistic Training/Testing data for Intrusion Detection*

Many research studies have been conducted since the anomaly-based detection paradigm was introduced by Denning [45]. However, despite the extensive amount of work, few applications of any ML-based intrusion detection systems in production environments have been reported. In recent years, some researchers have begun to question the applicability of the results reported in the literature. Gates and Taylor [7] argued that only a few ML-based IDSs have been widely used. They considered mainly the assumptions originating in Denning's work. According to them, the lack of appropriately obtained training data and testing methodologies that consider the network properties, such as continuous changes in content, volume, and attacks, is the main reason why the ML-based detection approach is unsuccessful.

Sommer and Paxson [6] conducted an extensive review of intrusion detection. They argued that the field is significantly different from other fields in which machine learning techniques have been successfully applied. They claimed that machine learning is more effective at finding similarities rather than detecting outliers, for instance. On the other side, the high cost of errors inhibits its use in production environments. The lack of available public and updated data hinders an appropriate system evaluation and comparison [46, 47]. In addition, Sommer and Paxson [6] and Paxson and Floyd [48] reported that real-world environments present a significantly different behavior from the data the systems are normally trained.

Table 1. Related Works comparison to *BigFlow*.

Work	Scalability	Detection	Reliability over time	Usage in Production Environment
Lee and Lee [12]	Yes, however it degrades drastically according to the used feature set	Connection thresholds	Not addressed	Unfeasible, requires data storage
Fortugne et al. [13]	Yes, however their approach demands a significant processing capacity	Unsupervised anomaly detectors	Not addressed, however the authors use unsupervised anomaly detectors	Unfeasible, requires data storage, and does not scale properly
Baer et al. [40]	Yes	Supervised batch learning	Not addressed	Partially feasible, however classification reliability is not addressed
Vernon and Victor [41]	Yes	Not addressed, however detection can be achieved by the means of SQL-like query language, in a threshold similar fashion	Not addressed	Partially feasible, however classification reliability is not addressed
Apache Metron [42]	Yes	Anomaly detection	Not addressed, however relies in anomaly detectors	Partially feasible, classification reliability is not addressed, and for the detection of anomalies network data must be stored
Prior work [43]	Yes	Supervised stream learning	Not addressed	Feasible, however only address classification resiliency (to adversarial attacks), reliability is not addressed
BigFlow	**Yes**	**Supervised batch and stream learning**	**Addressed**	**Feasible**

Thereby, the reliability of an ML-based detection system mainly relies on an appropriately created training dataset. Normally, strong assumptions about the training data need to be adopted. Canali et al. [46] created their dataset by collecting several Website contents from the Internet; they labeled each datum by using state-of-the-art tools and manually inspecting the data to ensure that the Website contents were correctly labeled. The authors assumed that most of the frequently visited Websites worldwide are benign and that the distribution of feature values is different for each class of Websites. The strongest assumption is that the dataset used to train the models presents the same feature distribution as real-world environments.

Moreover, when a dataset is obtained in a controlled environment, the authors normally statistically reproduce the user behavior. Shiravi et al. [36] created user profiles on the basis of the user behavior for each application during an observed time interval. Kendall [19] created a dataset by statistically reproducing the user behavior in an air force environment. In general, these approaches lack upgradeability, wrongly assuming that network traffic is immutable and considering that the user behavior can be modeled [6, 7].

3.2.1. Discussion

The task of building realistic datasets for intrusion detection schemes evaluation have been the subject of several studies in the literature over the last years [31, 47]. However, despite extensive efforts, currently the most used dataset remains the KDD99 [21], with several known flaws [20, 21]. Current approaches for building new intrusion datasets, which can be achieved either by monitoring the production environment behavior [16], or by creating a controlled environment [10, 20], fails at generating the expected network properties from production environments. In such a case, due to the highly variable nature of networked environments, the datasets must enable the fine-grained IDS evaluation. A fine-grained IDS evaluation aims at enabling the intrusion detector system operation to evaluate how her system reacts to each of the network properties.

Therefore, the approach proposed in this work aims to provide a publicly available intrusion database through the use of well-known tools in a controlled environment, thus providing the properties expected from an intrusion database. Moreover, the proposed method aims at enabling the fine-grained evaluation of intrusion detection schemes, regarding the reproduction of all the expected production environment properties. For instance, the identification of similar/new attacks, and similar/new services.

3.3 *On Dealing with Adversarial Attacks*

The lack of usage of ML-based intrusion detection methods in production environments was noted over the last years by a number of works [6]. Such usage gap is caused by several aspects; however, it is a consensus that the detection method must be at least reliable and easy to update [6]. The detection reliability is often considered in other areas [49], to this end, in general the authors [50] rely in the classifier class probability output to reject or not the decisions, while other approaches uses an ensemble of classifiers and establishes the classification reliability through a majority voting approach [51].

This dissertation proposes a reliable intrusion detection model, which, in order to adapt to network changes over time, resort to stream learning techniques [52]. However, due to its incremental update nature, such techniques are prone to adversarial attacks. In such a case, a sophisticated attacker may attempt to pervert the stream learning algorithm properties to evade the detection system. For instance, an attacker may change how an attack behaves, causing model updates with misclassified instances.

Despite being often considered in other areas; the classification reliability in stream learning field, in the presence of an adversary, still is in its beginnings. For instance, when a window-based stream learning detector is considered, the sliding window can be attacked to deceive the outlier detector. Some authors, however, considered the adversarial settings in anomaly-based intrusion detection.

Ling Huang et al. [52] defined a taxonomy used in their work to classify possible adversarial attacks against the machine learning system. The authors also evaluated the impacts that a poisoned training dataset incur in the classifier accuracy, in all evaluated cases the classifier became unreliable when the training dataset had misclassified attacks injected.

In the spam detection scenario, Blaine Nelson et al. [51] evaluated the training dataset poisoning impact on accuracy, the authors reported a 36% misclassification increase when the attacker had control of only 1% of the training dataset. The authors also evaluated a causative attacks resistance approach by identifying whether the new added instance results in accuracy improvements or not, despite this approach is effective, the authors relied in a supervised dataset (when all instances are prior classified). Such an approach cannot be employed in production as the instances are not prior labeled and the accuracy cannot be estimated in real time. In the malicious PDF detection scenario, Srndic and Laskov [53] evaluated a set of attacks against a well-known learning-based classifier for malicious PDF files, the authors were able to decrease

the classification accuracy from almost 100% to 28%. The authors also suggested that a multiple classifier system should be more resilient to such adversarial attacks, due to the need to evade several complementary classifiers.

Few authors address causative attacks in the network intrusion detection field [54], for instance, Benjamin et al. [55] developed the ANTIDOTE which relies in a robust PCA and a robust Laplace threshold that is less sensitive to poisoning attacks. However, their approach remains susceptible to exploratory attacks.

3.3.1. Discussion

Because of the evolving nature of networked environments, intrusion detection schemes must be able to reliably adapt to changes over time [6]. To achieve such goal in real-time, one must resort to stream learning techniques [32]. However, in such context, a sophisticated attacker may attempt to pervert the detector properties, either at training or testing time [23].

Several works have focused on building resilient (*to adversarial attacks*) machine learning techniques [52, 51, 55]. However, address both *causative* and *exploratory* attacks remains an open challenge, even for batch learning techniques [23]. In addition, those kind of adversarial attacks to stream learning techniques still in its beginnings.

In the light of this, this is the first work to address both causative and exploratory attacks using stream learning algorithms for intrusion detection field. The proposed approach remains reliable during both attacks, causative and exploratory, by employing a rejection mechanism and a class-specific outlier detector.

3.4 *Ensuring reliability in classifications*

This dissertation, in order to enable the reliable use of ML-based intrusion detection schemes over time in production environments proposes the reliability assessment of classifications.

In general, in the literature, classification reliability is assessed by the means of Chow [37] or class-related-thresholds (CRT) [38]. In the prior, a single threshold is used the assess the reliability of a given classification, while in the former, each class has its own related threshold. Even so, in both approaches, the reliability measure of a given classification is computed by the means of the confidence value output by a given classifier.

Over the last years, these kinds of techniques have been extensively used in the literature to assess classifications. In general, they are used in fields with a high cost of errors. For

instance, in the medical diagnosis field, Hanczar and Dougherty [56] aimed at providing a desired level of acceptable error-rate. To this end, the authors have performed a wrapper-based feature selection in which only solutions that met the error and rejection rates thresholds are selected. The authors were able to provide the desired error rate from the system as a user defined parameter. In their evaluation tests, using both real and synthetic data, with a CRT approach, the authors have shown a relation between error-reject tradeoff. In such cases, the classification accuracy is improved when a reject option is considered.

Another use of reject option was proposed by Mesquita et al. [57] in the field of software fault detection. In their work, the authors have proposed the use of a one-class classifier with reject option. For the classification process, one classifier is built for each class, in which each classifier has its own reject threshold, similarly to CRT. For the classification process, a decision is considered reliable only if a consensus is found between the classifiers. In other words, only decisions that were not rejected by both classifiers, or when a consensus between the classes were found, are accepted. For the evaluation tests, the authors have used 5 datasets They were able to improve the system accuracy when a reject option is considered.

Several other works have also reported the accuracy improvement by the means of a reject option [58, 59]. However, surprisingly, in the intrusion detection field the reliability assessment is often ignored. In a prior work [60], a first attempt to address it was made. To this end, classification reliability assessment has been addressed using stream learning algorithms. In such context, unreliable classifications, as given by the classifier confidence value, in a CRT-based approach, were rejected and then used for incremental learning over time. The evaluation tests have shown that assessing the classification reliability through the classifier confidence can help at improving the overall system reliability. However, the findings during the evaluation tests have shown that the system significantly increases the rejection rate over time, even when it is updated.

In recent years, some works have begun to address reliability assessment in the presence of new environment behavior. For instance, in the malware classification context, Jordaney et al [24] have shown that traditional approaches are unable to provide classification reliability in the presence of new malware behavior. In their work, to provide classification reliability a system named *Transcend* was designed. It assesses the classification reliability by the means of a conformal evaluator. The purpose of the conformal evaluator is to measure the reliability of a given classification. To this end, the authors used a statistical comparison of samples seen during deployment with those used to train the model, thereby building metrics for classification

quality. Such task is achieved by computing two values *credibility* and *confidence*. *Credibility* define how well the instance fits into the assigned class, while *confidence* measures how well it does not fit to the opposite classes.

In their evaluation tests, the authors have used a similarity-based function for the statistical computation (*credibility* and *confidence* values computation). For evaluation purposes two datasets were used, the first dataset was made of malwares from 2010 to 2012, while the second from 2010 to 2014. The authors then trained a classifier using malwares from one dataset and evaluated it on the other. When *Transcend* is not considered the classifier greatly decreases its performance. However, when their approach is used, they are able identify unreliable classifications and reject them, maintain the system reliability.

The work of Jordaney et al. [24] was one of the first works to show, in the security field, that aged models can become unreliable. Moreover, they showed that traditional classification reliability assessment approaches are unable to provide classification reliability in such context. However, some aspects regarding the applicability of their work in the network-based intrusion detection context must be noted. First, the authors have used two different datasets to evaluate the reliability of aged classification models. Each dataset was obtained by different authors, works, environments, settings, amongst others. In this sense, one cannot properly conclude whether a model perform poorly in a different dataset because of the time of their building, or because of how the dataset was obtained - considering the aforementioned characteristics. Second, in order to evaluate if a decision is reliable or not, the authors have computed the *credibility* and *confidence* values. Such values were computed for each classified instance in each dataset, and then evaluated accordingly. The authors goal was to establish whether a concept drift have occurred or not through the analysis of both values in both datasets. However, in the network field to achieve such task one would also need to store the values occurred in a period of time. In contrast, the reliability assessment should be made in an instance-based approach, rather than a period-based one.

A single dataset was used in Kantchelian et al. [61] to evaluate the behavior of ML-based malware detection. The authors have built a dataset containing malware samples from 2007 to 2013. In their evaluation tests, the authors findings have shown that such models are only able to properly classify older instances than new ones. In other words, ML-based algorithms are unable to cope with changes in malware behavior changes over time. The authors also show that, ML algorithms improve their detection rate over time when further instances

are used for training. Thereby, such findings corroborate the results obtained by Jordaney et al. [24].

Finally, another approach to assess classification reliability in the security context have been proposed by Maggi et al. [62]. In their work, the authors show that web applications behavior changes over time. In order to identify such changes, the authors analyze the application response content for new fields. In their evaluation tests, the authors were able to significantly decrease the false-positive rates, by up to 100%. Although their approach presented significant improvements in the detection accuracy, it hinders the usage for other areas. The main difficulty is regarding the application-specific approach. Because the use of their approach in other fields would imply in knowing how each of the monitored applications behaves. Nonetheless, in the network-based intrusion detection context, such an approach cannot be used since network behavior (network flows) varies greatly.

3.4.1. Discussion

Classification reliability have been extensively addressed in other fields with a high cost of errors, such as medical diagnosis [56], optical character recognition [58, 59], amongst others [57]. However, surprisingly, in network-based intrusion detection it still is in its beginnings [60]. In general, classification reliability can be achieved by the means of a typical CRT-based approach [56, 57, 58, 58, 60]. In such cases, the classifier confidence value is used as a metric to measure the classifier reliability for a given instance. However, it has already been demonstrated in related works that classifier confidence values can be biased in the presence of unknown behaviors [24]. Thereby, rendering traditional CRT-based approaches not feasible for classification reliability assessment over time. In such context, a promising approach has been proposed in *Transcend* [24], in which classification reliability is assessed by the means of a conformal evaluator.

Table 2 shows a summary of related works comparison to the proposed classification reliability assessment approach.

Table 2. Classification reliability assessment approaches comparison.

Work	Field	Approach	Address unreliable classifier	Usage in production environment over time
Hanczar and Dougherty [56]	Medical diagnosis	CRT-based to achieve a desired error rate level	No, relies on confidence given by the classifier	No, classifier must be reliable
Mesquita et al. [57]	Software fault detection	CRT-based for one-class classifiers to achieve classification consensus	No, relies on confidence given by the classifier	No, classifier must be reliable
Prior work [60]	Network-based intrusion detection	CRT-based to maintain the classification reliability	No, relies on confidence given by the classifier	No, classifier must be reliable
Jordaney et al. [24]	Malware detection	Conformal evaluator is used to assess decisions made by classifier	Yes, classifier decisions are evaluated through a conformal evaluator	Yes, however the identification of proper thresholds for rejection was not addressed
Maggi et al [62]	Malicious website detection	Examines application response looking for changes	Yes, changes in environment behavior are identified by examining the event content	Yes, however their approach requires application-level knowledge
Proposed approach	**Network-based intrusion detection**	**Conformal evaluator is used to assess decisions made by classifier**	**Yes, classifier decisions are evaluated through a conformal evaluator**	**Yes, threshold can be defined according to administrator needs**

3.5 *Reliability in intrusion detection*

Since the introduction of the anomaly-based intrusion detection paradigm by Denning [45], the challenge of intrusion detection has been tackled by a number of works. In such context, machine learning techniques, mostly by pattern recognition means, have stand out over the last years.

Several approaches have been proposed so far in this sense. For instance, Gudadhe et al. [63] proposes the use of supervised learning by the means of a boosting technique to classify intrusion attempts. In their work, an ensemble is created using decision trees as base-learner. In their boosting procedure, classifiers are built iteratively, in which each next classifier is built on top a weighted adjusted training dataset. The instances weights are adjusted according to the classification correctness from the previous classifiers, in which misclassified instances are given more weight. For the evaluation tests, the authors have used the well-known KDD99 [20] intrusion dataset, in which their method presented promising results. However, the authors did not evaluate their approach on unseen data.

Another approach proposed by Gaikwad and Thool [64] have performed intrusion detection by the means of a bagging technique. Similarly to [63], they have used decision trees as their base-learners. Their bagging approach creates each decision tree using a subset of instances with replacement from the training dataset. In order to further reduce training time, the authors perform a feature selection by the means of a genetic algorithm approach. In their evaluation tests, also using the KDD99 dataset, the authors were able to reduce training time without decreasing the accuracy rates. The detection of new attacks was not addressed.

Another popular approach relies in ensemble of classifiers. For instance, Haq et al. [65] proposes the use of several classifiers in a majority voting scheme. In their work, naïve bayes, decision tree and a bayesian network were used for intrusion detection. The classifier's output were combined in a majority voting scheme. For the reduction of training time the authors have relied in a wrapper-based feature selection for each classifier. In their evaluation tests, using KDD99, the authors were able to improve the system accuracy, when compared to a single classifier, while also decrease training time by the means of a feature selection technique. The detection of new attacks was also not addressed.

In contrast to the aforementioned works, some authors consider a scenario without ground truth, i.e., environments in which the event's label are not known previously. For instance, Fontugne et al. [15] proposes the use of several unsupervised machine learning algorithms for the detection of anomalies in internet-wide traffic. In their work, the authors use

four unsupervised state-of-the-art algorithms for the identification of anomalies in MAWI [16] network data. MAWI is a publicly available database with daily-provided network traffic. Such network traffic is obtained by the monitoring of a transit link between Japan-USA. For each day, 15 minutes of traffic is recorded, between 14:00 and 14:15. Anomalies are identified in a daily-basis by the means of a voting scheme over the unsupervised machine learning algorithms. The authors evaluate several combination approaches to this end, in a seven-year range. Although their work performs a comprehensive evaluation of state-of-the-art unsupervised machine learning algorithms, their technique requires the storage of such data. However, differently from other works, their approach relies in a representative network database, MAWI. Moreover, because of the nature of unsupervised machine learning algorithms, they are able to identify new attacks.

As can be seen, several approaches are proposed for identification of intrusion attempts. However, the reliability of such schemes for the classification of attacks over time is typically not addressed. When a supervised machine learning approach is considered, the authors, in general, does not address the need for model updates, and how such task could be accomplished in production usage. On the other side, when an unsupervised machine learning approach is evaluated, the storage of network data is necessary.

3.5.1. Discussion

Several works have proposed and evaluated highly accurate intrusion detection models in the last years. However, despite their promising reported results, there remains a gap between the number of works that uses ML-based approaches and their actual use in production. One of the main issues regarding such gap is regarding the considered evaluation during the system development. In general, the authors relies in traditional machine learning evaluation approaches, in which a single test dataset is considered. Moreover, several works perform such evaluation by the means of an old dataset, KDD99, with several known flaws [20, 21].

In contrast, real network environments present several challenging aspects to ML-based approaches. Similar to malware detection, network behavior changes over time. Thereby, ML-based approaches must be able to cope with such changes. However, surprisingly, the majority of works does not address such changes. Thereby, the challenge of addressing the network changes over time remains yet to be solved.

Table 3 shows a summary of related works comparison to the proposed reliable intrusion detection scheme.

Table 3. Reliability in intrusion detection works comparison.

Work	Approach	Classification reliability	Classification reliability over time	Model updates
Gudadhe et al. [63]	Supervised batch learning. Boosting technique	Not addressed	Not addressed	Not addressed. Model rebuilding must be performed
Gaikwad and Thool [64]	Supervised batch learning. Bagging technique	Not addressed	Not addressed	Not addressed. Model rebuilding must be performed
Haq et al. [65]	Supervised batch learning. Ensemble technique	Not addressed	Not addressed	Not addressed. Model rebuilding must be performed
Fontugne et al. [15]	Unsupervised batch learning	Not addressed. However unsupervised techniques are used	Not addressed. However unsupervised techniques are used	Technique is executed in a daily-basis, network traffic must be stored
Proposed approach	**Batch and Stream learning coped together**	**Conformal evaluator is used to assess decisions made by classifier**	**Conformal evaluator is used. Models are incrementally updated over time**	**Yes, stream learning algorithms are incrementally updated over time**

Chapter 4

A Reliable Intrusion Detection Model

Several approaches have been proposed in the literature for intrusion detection by the means of machine learning techniques. However, despite extensive efforts, they, in general, fail at providing generalization capable models, dealing with changes in network traffic content, provide reliable classifications, deal with high network bandwidth, while also address adversarial settings.

In the light of this, this work introduces a new intrusion detection model, namely reliable intrusion detection. The main characteristic of the reliable intrusion detection is the usage of both batch and stream learning algorithms coped together. In this sense, the proposed model aims at exploiting the characteristics of each type of learner, in a cascade pipeline approach, to overcome the challenges faced by ML-based approaches for intrusion detection in production environments.

The proposal overview and how it addresses such challenges are introduced in Section 4.1. Section 4.2 presents *BigFlow*, which aim at addressing the challenges of high-speed networks. Section 4.3 introduces a methodology toward the building and evaluation of generalization capable batch learning models. Section 4.4 presents an approach for ensuring reliable classifications of batch learning models over time. Section 4.5 addresses resiliency to adversarial attacks for stream learning algorithms. Section 4.6 presents a classification reliability assessment approach, which aims at providing a classification reliability degree in face of new network traffic behavior. Finally, section 4.7 introduces how the reliable intrusion detection model addresses behavior changes over time. Finally, section 4.8 presents the proposal discussion.

4.1 *Proposal Overview*

Current ML-based approaches for intrusion detection lacks reliability to face production environments over time. In the light of this, this work aims at providing a set of techniques, that enables the reliable near real-time intrusion detection to be performed in high-speed networks over time. Figure 4 shows the overview of the proposed reliable intrusion detection model.

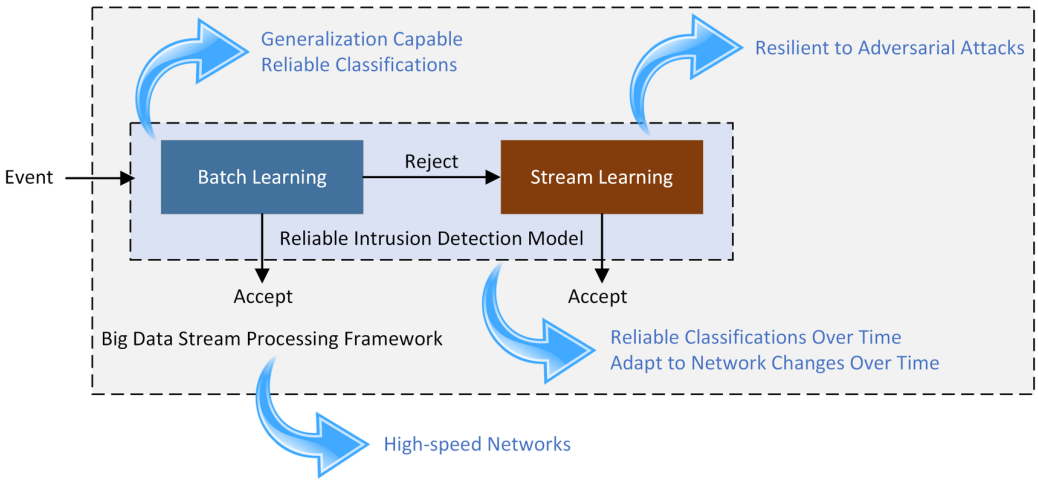


Figure 4. Overview of the proposed reliable intrusion detection model, and how each learner overcome the challenges of production environments.

The reliable intrusion detection model receives as input an event to be classified, which is then passed to a batch learner algorithm, that presents the following properties:

- *Generalization Capable* (Section 4.3). The model must be able to generalize the behavior from the training dataset to other environments. In such a case, the generalization must take into account the classification of known, similar and new attacks; known, similar and new services, and their contents. In this manner, it becomes possible to ensure that the system can correctly classify the events, regarding the time of its building, even from a limited training dataset;
- *Reliable Classifications* (Section 4.4). Despite the model being able to generalize the behaviors that occur in the production environment, new traffic content (e.g. new attacks, or services) will occur over time. Thus, the model must be able to identify whether the instance can be reliably accepted, or should be rejected, considering the behavior seen during the period of its building;

An instance that was rejected by the batch learner, is assumed to be an unseen behavior. In such a case, a new traffic behavior might be occurring and thereby a stream learning algorithm must classify it. The stream learning algorithm is used in order to enable the system

to incrementally adapt to changes in network behavior over time, without incurring in the batch model retraining. The stream learner algorithm must present the following property:

- *Resilient to Adversarial Attacks* (Section 4.5). Differently from batch learning, stream learning algorithms are able to incrementally adapt to changes in network behavior over time. Such property significantly decreases training time, because the current model is not discarded, a desired property for high speed networks, but, on the other hand, it is prone to evasion attacks. Thereby, the stream learning algorithm must be able to identify whether a new instance can reliably be incorporated in the model or not;

Finally, despite being able to reliably adapt to changes over time, in production usage, the stream learning model may not be updated for a period of time. For instance, the system administrator may not able to provide the event's label. Thereby, the stream learning algorithm identifies unreliable classifications over time by the means of a conformal evaluator.

- *Reliable Classifications Over Time* (Section 4.6). The identification of unreliable classifications in the presence of unknown traffic behavior is a challenging task. This because it is not possible to identify unreliable classifications by the means of the classifier confidence, as it may not be up to date. Thereby, a conformal evaluator must be used to assess the classification reliability even in the presence of new network behaviors;
- *Adapt to Network Changes Over Time* (Section 4.7). Finally, the proposal coped together, batch and stream learning, provides a model towards a reliable near real-time intrusion detection in high-speed networks over time;

The next subsections further describe each technique in detail.

4.2 *BigFlow*

In order to address the evolving behavior of high-speed networks, an approach namely *BigFlow* is presented. *BigFlow* performs the feature extraction in high-speed networks 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 near real-time, summarizing the information about the traffic between two hosts in a time interval. Because only the statistics values need to be stored in memory, during the specified time interval, there is no requirement for the storage of the network packets.

The next subsection describes in detail the feature extraction stage, including the architecture of the modules that implement it and a description of the main components.

4.2.1. Feature Extraction

In order to measure and classify network activity, it is necessary to compute statistics about the network traffic exchanged between the relevant entities over a period of time. There are several works that focus on extracting the features values for flow classification [5, 12, 13]. However, contrary to *BigFlow*, none of them is capable of monitoring high-speed evolving networks. In such context, to avoid the storage of network data, the feature extraction should be made at near real-time. To this end, the feature set was established according to the processing demanded for its extraction, which is, in general, responsible for the most significant part of the overall demanded processing [5].

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 the features extracted based solely in the data sent/received from a 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[1].

The architecture of the feature extraction module of *BigFlow* is shown in Figure 5. A set of 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 to provide a single interface for the monitored agents.

The *Message Consumer* 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* in a PE of the stream processing, using the shuffle approach. The *Message Parser* in turn, establishes the event source, fields, and type (e.g., network packet or netflow record).

[1] The complete list of extracted features can be found in Appendix 1

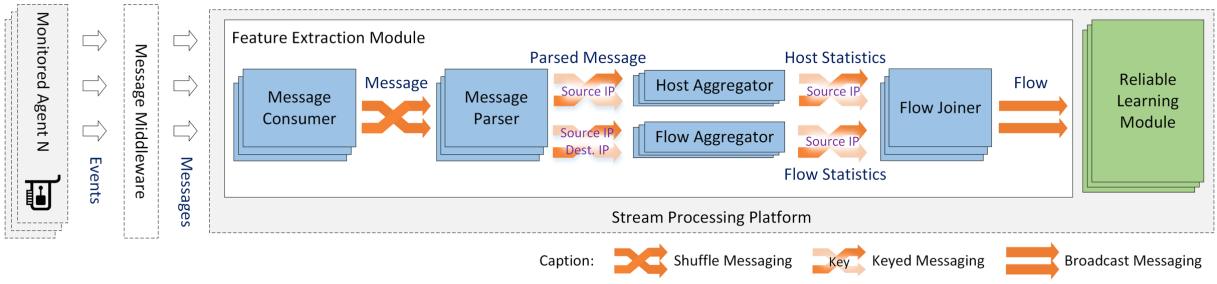


Figure 5. *BigFlow* real-time feature extraction module architecture for high-speed networks.

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* reads both types of events from the message queue, and distributes them through the available *Message Parsers* keeping the computing load evenly distributed. The *Message Parser*, 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* is calculated using the hash of the event source addresses (source IP address), whilst the key for the *Flow Aggregator* relies on the XOR of 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 as the *Tumbling Window*. Each *Tumbling Window* 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 flow features values are exported in a host or flow statistics format, and the feature values computation starts over again for a new window.

Figure 6 illustrates how *BigFlow* computation through the *Tumbling Window* is done. The figure considers two hosts exchanging messages over the network for 60 seconds, and a *Tumbling Window* period of 15 seconds. 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 seconds ("T.Window 1" in the figure). When the *Tumbling Window* expires, it exports the host and flow statistics to the next module. As a new event arrives after the initial 15 seconds, the *Host* and

Flow Aggregators create another *Tumbling Window* ("T.Window 2" in the figure) and start the flow features computation again.

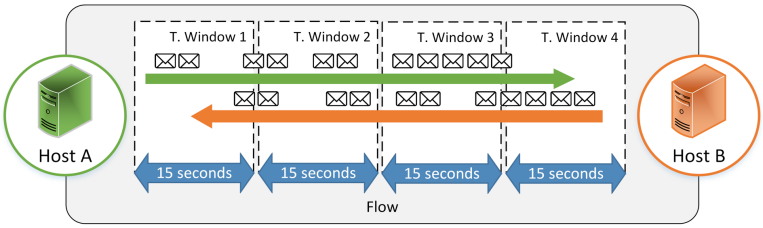


Figure 6. *BigFlow* flow computation through the *Tumbling Window* approach.

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* module is responsible to receive all host and flow statistics values and join 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, each *Flow Joiner* PE is responsible for a range of the 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* 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* also relies on the *Tumbling Window*.

4.3 *Batch Learning - Generalization*

The reliable intrusion detector receives as input the feature vector, and forward it to the batch learning algorithm. The batch learner in its turn must be able to correctly classify the event, according to the behavior seen in the training dataset. However, as it is not possible to assemble a training dataset with all possible production environment behaviors, the first model requisite is related to the generalization capacity.

Three steps are normally involved in a typical batch learner evaluation method. Initially, the classifier model is created using a trained dataset. Then, a validation dataset is used to improve the created model. Finally, the model is evaluated by means of a test dataset. Because of the lack of publicly available data in the NIDS field, experts normally divide a single dataset

into three parts. Thus, a typical evaluation method assumes that the used datasets resemble the production environment.

However, this assumption does not hold in networked environments. In such a case, the creation of an intrusion database that presents all the possible behaviors of a production environment is not a feasible task [7]. Even if it were possible to create a perfect database, it would still not be effective because it would consider that network traffic is immutable [6]. Thus, an evaluation method, that enables to validate the expected properties for batch learner algorithms, when applied to the network-based intrusion detection area is required. The purpose is to validate the common assumptions reported in [6, 7]. The overall process is shown in Figure 7 and further described in the following sections.

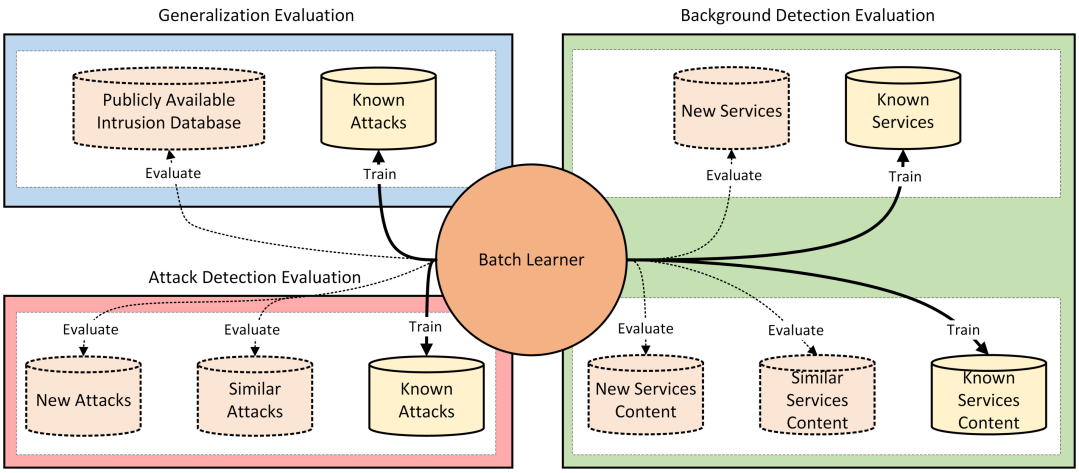


Figure 7. *Batch Learner* evaluation method towards generalization capable models.

4.3.1. Attack Detection Rate

The most important assumption about ML-based intrusion detection systems is that it is capable of detecting new attacks. The premise is that an attack, whether new or known, shows behavior that is significantly different from that of a typical system's usage and, thus, can be identified by detecting outliers for instance [66]. However, incoming events are classified, in general, on the basis of their similarity to the known and prior-labeled events in the training dataset, according to a similarity metric.

Thus, only new attacks that behave similarly to already known attacks can be correctly classified. By definition, it is not possible to train a machine learning detection technique using unknown attacks. However, it is possible to measure an intrusion detection system's capability by controlling the events included in the test datasets. For instance, a system can be trained with a limited type of attack and tested with similar or completely different attacks. The definition

of the similarity of events is context-dependent and must be determined according to expert knowledge. The process is shown in Figure 8.

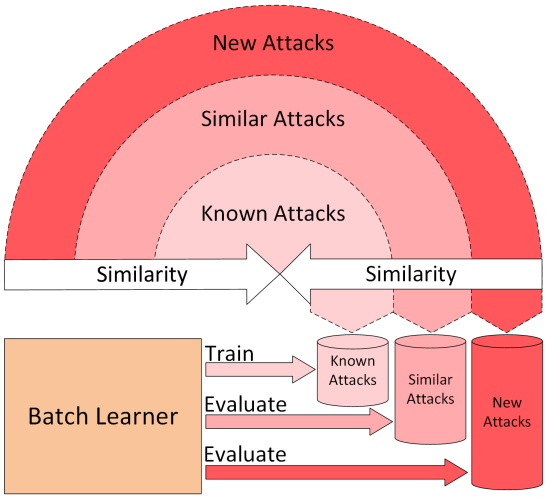


Figure 8. *Batch Learner* attack evaluation method.

Initially, the batch learner is trained with a limited set of attacks that are similar and present an expected behavior during the system usage in production environments (Figure 8, *Known* Attacks). Then, the created detection model is evaluated using databases containing similar and new attacks (Figure 8, *Similar* and *New* Attacks). Similar attacks in this context are attacks with a high similarity to the attacks on which the system was trained, whereas new attacks are attacks that are significantly different from known attacks.

The similar attack detection rate is defined as the system's capacity to detect events with behaviors similar to those of known attacks. This property is desirable in intrusion detection systems because of the highly variable nature of networked environments. Thus, a system must be able to cope with similar attacks, as a single database cannot contain all possible attacker behaviors. Similar attacks may present a different pattern and can evade signature-based systems, where detection is performed by matching against well-known attack patterns. However, similar attacks may present the same or a similar behavior and should be detected by the detection scheme if the used features are adequately discriminant.

The new attack detection rate defines the system capacity to detect significantly different types of attacks; i.e., attacks that present a behavior that is completely different from any known behavior on which the system was trained. This type of incidence occurs in production environments, as it is not possible to train a system with all known or new attacks. The used detection scheme must be able to relate the new attack behavior to the known attacks and correctly detect it, which is the premise when using any ML-based approach.

During the database creation for machine learning detection schemes, the background traffic must also be generated. The incidence of an attack affects the response of a service to legitimate requests; e.g., one attack type can make a service unavailable, whereas another can make a service reply only to a specific set of requests. Thus, the occurrence of an attack can affect also the background traffic detection rate.

Note that databases must use the same background traffic creation approach. Thus, this approach uses the background traffic as the baseline, allowing it to identify the behavior of the same set of services under each type of attack and the performance of the detection engine while detecting such changes.

4.3.2. Background Detection Rate

The background traffic is composed of two entities: the client and the server. The client generates requests according to the contents and services provided by the server. Thus, the background traffic may vary in content and the service requested.

Because of the lack of publicly available data, researchers have assumed that network traffic is immutable [67]. Anomaly-based systems are normally tested using a single database, divided into three parts having the same background traffic behavior.

It is known that network traffic continuously changes [6, 7]. Thus, it cannot be assumed that a behavior evidenced in a certain period during the intrusion database creation will remain immutable over time. The detection scheme must be able to track the background traffic behavior changes while still performing its detection at a reasonable rate when a classifier model update is not possible. Moreover, it must be able to properly identify variations of known normal behaviors.

When evaluating an intrusion detection method, one must consider the database limitations. It is not possible to create every service and content that will be evidenced in production environments. Thus, it is not feasible to create every possible background traffic behavior. However, a detection scheme must be able to detect different background traffic contents and different services, with their new type of content. To present such properties, the evaluation databases must be modified to allow them to be tested in an ML-based system.

Similar to the attack detection rate method (Section 4.3.1), the background detection rate method consists of restricting the used databases to two perspectives: the service content (Figure 9) and the services (Figure 10).

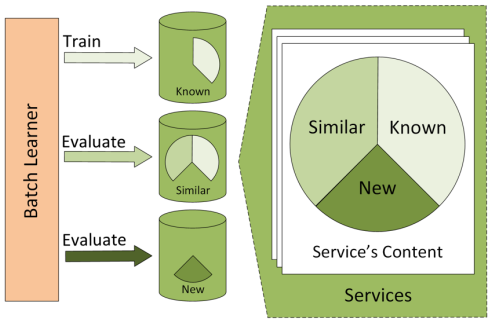


Figure 9. *Batch Learner* service content evaluation method.

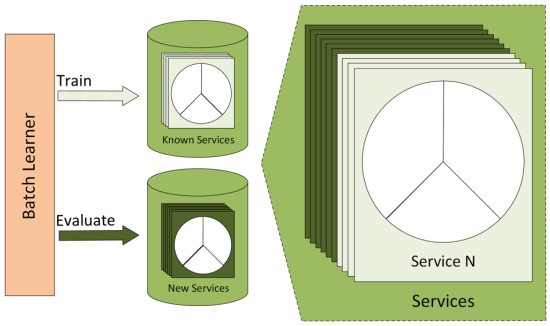


Figure 10. *Batch Learner* service evaluation method.

The service content detection rate is established by limiting the client requests, which are divided into three groups: known, similar, and new content (Figure 9). However, the service detection rate is established by restricting the number of services in the training dataset and evaluating the classifier with a new set of services not used during the training stage (Figure 10).

In the evaluation of the background detection rate, the attacks are used as the baseline. The same set of attacks is generated in each scenario, allowing the attack occurrence to be evidenced in each used service and its content.

4.3.3. Generalization Evaluation

Several methods have been proposed in the literature for creating intrusion databases; however, despite extensive efforts, they are all exposed to the problems inherent in the method used for their creation [68]. Thus, to evaluate the database used during the system design, as well as the method used for event detection, an evaluation method that uses a publicly available intrusion database is needed. Thereby, the system evaluation using a publicly available database provides a baseline comparison reference and the generalization rate.

The generalization is a desirable property for any machine learning technique. The set of extracted features must allow the classifier to generalize the problem appropriately by distinguishing the classes, regardless of the current environment in which it is operating. Thus, the classifier model built from the set of extracted features may be used in other environments that aim to detect the same type of events.

In this way, evaluation that uses a publicly available database ensures that the conceived detection scheme can operate independently of the environment in which it was conceived.

4.3.4. Generalization Evaluation Method Summary

The intrusion detection field faces challenges that are significantly different from those of other areas where machine learning has been successfully applied [6]. The proposed evaluation scheme is aimed to test the expected properties of a machine learning intrusion detection scheme. The following properties can be provided by the proposed evaluation method (Figure 7):

- Detection rate for known, similar, and new attacks (Section 4.3.1);
- Detection rate for known and new services (Section 4.3.2);
- Detection rate for known, similar, and new services' content (Section 4.3.2);
- Detection rate while operating in a different environment (Section 4.3.3).

4.3.5. On Building Generalization Capable Batch Learning Models

To improve the detection rates mentioned above (Section 4.3.4), a multi-objective feature selection method specific to the intrusion detection field is proposed. This method considers that, during the system development, the system designer that takes into account the following detection properties of the detection system: *attack* ($attack_{rate}$), *normal* ($normal_{rate}$), and/or *generalization* ($generalization_{rate}$), is able to build generalization capable models.

As described in Section 4.3.1, an intrusion detection system may face three distinct attack behaviors in production environments: known, similar, and new. During the classifier training, the detection algorithm learns only the known behavior. However, in production environments, the probability of each attack behavior occurring is unknown. For instance, an IDS that was trained with a network-based DoS attack behavior (known) may also face application-level DoS attacks (similar) having an unknown occurrence probability. However, in the production environment, the system administrator expects that an intrusion detection engine is able to detect attacks according to the accuracy rate obtained during the classifier testing, regardless of the current attack type the system is facing. Thus, the attack detection rate in a production environment can be calculated according to Equation 1.

$$attack_{rate} = average\begin{pmatrix} attack_{rate}^{known}, \\ attack_{rate}^{similar}, \\ attack_{rate}^{new} \end{pmatrix} \quad \text{Equation (1)}$$

Where $attack_{rate}^{known}$ denotes the system detection rate for known attacks, $attack_{rate}^{similar}$ denotes the detection rate for similar attacks, and $attack_{rate}^{new}$ denotes the

detection rate for new attacks (Figure 7, *Attack Detection Evaluation*). Thus, the system attack detection rate ($attack_{rate}$) is represented by the average detection rate of known, similar, and new attacks in production environments.

The same property is expected in a normal (background) traffic perspective. The requested service, either known or new, must be correctly detected, as well as its content: known, similar, and new. Thus, the normal detection rate is established according to Equations 2, 3 and 4.

$$normal_{rate}^{service} = average\begin{pmatrix} normal_{rate}^{known\ service}, \\ normal_{rate}^{new\ service} \end{pmatrix} \quad \text{Equation (2)}$$

$$normal_{rate}^{content} = average\begin{pmatrix} normal_{rate}^{known\ content}, \\ normal_{rate}^{similar\ content}, \\ normal_{rate}^{new\ content} \end{pmatrix} \quad \text{Equation (3)}$$

$$normal_{rate} = average\begin{pmatrix} normal_{rate}^{service}, \\ normal_{rate}^{content} \end{pmatrix} \quad \text{Equation (4)}$$

Where $normal_{rate}^{known\ service}$ denotes the system detection rate for known services, $normal_{rate}^{new\ service}$ denotes the detection rate for new services, and $normal_{rate}^{known\ content}$, $normal_{rate}^{similar\ content}$, and $normal_{rate}^{new\ content}$ refer to the detection rate of known, similar, and new services' content, respectively. The system's normal detection rate ($normal_{rate}$) is represented by the average detection rate of $normal_{rate}^{service}$ and $normal_{rate}^{content}$.

Finally, the generalization capacity of a system is directly established by the system detection rate in another environment (Figure 7, *Generalization Evaluation*). Thus, the generalization rate ($generalization_{rate}$) is established according to Equation 5.

$$generalization_{rate} = generalization_{rate}^{public\ dataset} \quad \text{Equation (5)}$$

It is important to note that attack ($attack_{rate}$), normal ($normal_{rate}$), and generalization ($generalization_{rate}$) are conflicting properties (objectives). For instance, an increase in $generalization_{rate}$ may decrease the intrusion detection rate for normal and attack events because of the increase in the generalization capacity (commonly referred to as the receiver operating characteristic curves for two class decision systems) [69].

Thus, the operating points must be established according to the system designer's needs. For example, the generalization property may be desired in systems that will be used in several

different environments (commercial products for instance); however, in proprietary systems, this property may not be desired.

4.4 *Batch Learning – Reliability in Classifications*

Despite the building of generalization capable models, new event's behavior might occur over time, for instance when a new attack occurs. In such a case, the batch learner model must be rebuilt, in order to properly incorporate the new event knowledge into the model. However, the model rebuilding in high-speed networks is not an easily feasible task, which might occur only after a significant period of time. In the light of this, a method must be devised to enable the identification of reliable classifications, which enables to establish whether the event presents a similar behavior to the training dataset, or not.

Hereafter, a rejection technique and a combination of classifiers to provide a more reliable detection is further explained. This solution aims at providing a reliability degree over extended periods of time, even though it does not classify all the input events.

4.4.1. Changes in Feature Values Distribution

When a classifier is operating, its accuracy depends on the feature values distribution being similar to that of the training dataset (usually composed of real network traffic). If the distribution changes significantly, the classifier model should be updated, or its accuracy may decrease. This update usually requires expert knowledge to label new events and to rebuild the model, which may not be practical in real-world environments, or may occur after some delay. To test a classifier designed to operate in such environments, a method to assess whether it is still reliable even when the network traffic changes is required. Here an evaluation scenario is described, and an event rejection method is proposed, which allows the classifier to operate reliably even when it cannot be easily updated.

To overcome the limitations of other works in the literature, a rejection method that takes into account the frequent content changes observed in real-world network traffic is proposed. In addition, the usage of several independent classifiers using different machine learning algorithms is proposed. After each classification, the approach checks whether there are enough similarities between the classifier outputs class (normal or attack) and the class occurrence observed in the training dataset. If there is not a predominant match, the classification is deemed unreliable and the event should be rejected because the features used to build the model and the current event are not similar enough for a reliable classification. An

event rejection means that none of the classifiers can reliably assign a class to an input event; in this case, the event is rejected rather than being potentially incorrectly classified.

4.4.2. Scenario

Figure 11 shows a real-world scenario whose feature distribution changes over time. It considers the feature set of SYNFlood attacks [4] as baseline in the attack model. If an HTTPFlood attack [4] occurs, it can still be detected because the feature distribution of the two attacks are similar. However, if the network traffic changes significantly (as in an Exploit attack), the classification output becomes unreliable due to the significant change in the feature set. In such cases, if the model cannot be updated, another technique should be used to provide a reliable classification.

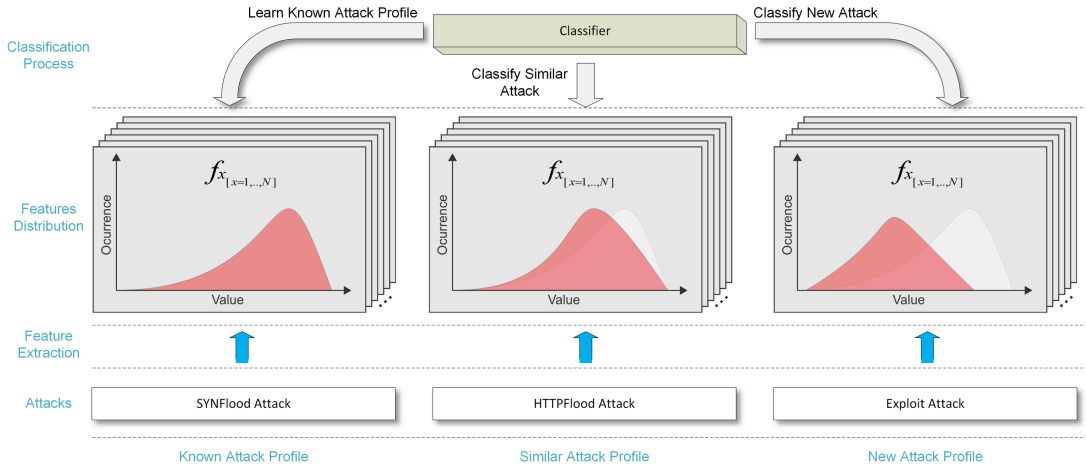


Figure 11. Changes in features distribution, considering SYNFlood Attack as reference.

4.4.3. Rejection Engine

One way to detect changes in the network traffic profile is to monitor the distribution of values in the extracted feature set (Figure 11, *Exploit Attack*). A significant change in feature distribution may indicate that a new attack is occurring. However, it is not easy to detect profile similarities from network events occurring in real time. In such context, two ranges for each attribute (one for each class) are defined to determine whether a feature value is valid. When an extracted feature lies within the appropriate range, the feature is considered valid.

For evaluation purposes, three traffic scenarios can be used: a baseline scenario, a scenario with network traffic changes but similar to the baseline scenario, and a scenario with new attacks (Figure 11). The baseline scenario is used to obtain the rejection range thresholds and the attack models; the other scenarios are used to evaluate the rejection method.

For each feature (fx, x=1, 2, …, N) and class (normal or attack), two rejection thresholds ($t_{lower}$and t_{upper}) are computed. The thresholds define the range within which a feature value is valid. The range is class-specific because the feature distribution for each class is different. The thresholds are defined with respect to an ∝ value (Figure 12), which establishes a percentage of instances in the validation dataset that fall outside the defined thresholds, but still provide the desired model reliability. To determine the value of ∝, an experimental analysis must be performed.

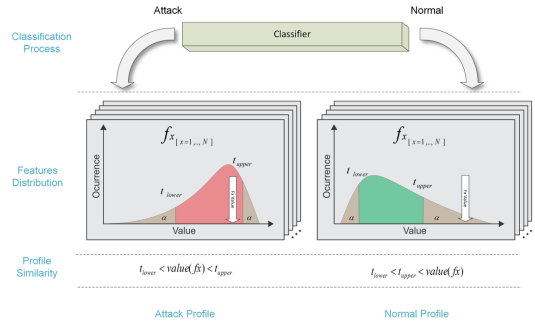


Figure 12. Features within the range for a class (attack).

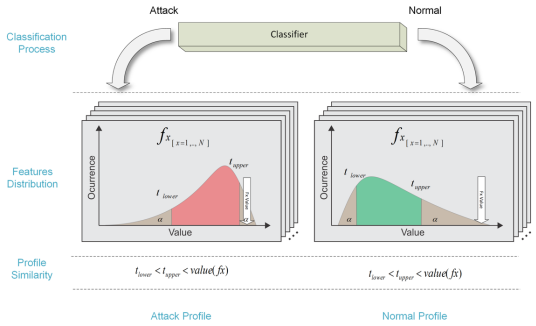


Figure 13. Feature outside the threshold range for both classes.

Thereby, for each feature fx, $profile_{similarity\_fx} = 1$, if the values for fx lie in the threshold interval ($t_{interval}$): t_{lower}< value (fx) < t_{lupper}; otherwise, $profile_{similarity\_fx} = 0$. For example, $profile_{similarity\_fx} = 1$ for the attack profile (Figure 12) and $profile_{similarity\_fx} = 0$ in Figure 13 for both profiles.

If N denotes the number of features in the feature set, the profile instance similarity ($profile_{similarity}$) is defined according to Equation 6.

$$profile_{similarity} = \frac{\sum_{x=0}^{N} profile_{similarity\_fx}}{N} \qquad \text{Equation (6)}$$

Finally, the classifier output should be rejected when it presents a low $profile_{similarity}$ (e.g., $profile_{similarity}$< 0.7); otherwise, the event is labeled with the class informed by the model. Using this approach, it is possible to establish the profile similarity without the need to keep the feature values history to identify a change in feature distribution, increasing the overall system throughput in high-speed networks.

The output of the combined classifier is assigned via a combination algorithm (Figure 14), choosing the majority of the outputs of the individual classifiers whose outputs were not rejected. In the example of Figure 14-c, the output is rejected because no individual classifier

output is valid, while in Figure 14-a and Figure 14-b the output is accepted because there is at least one classifier that can reliably classify the event.

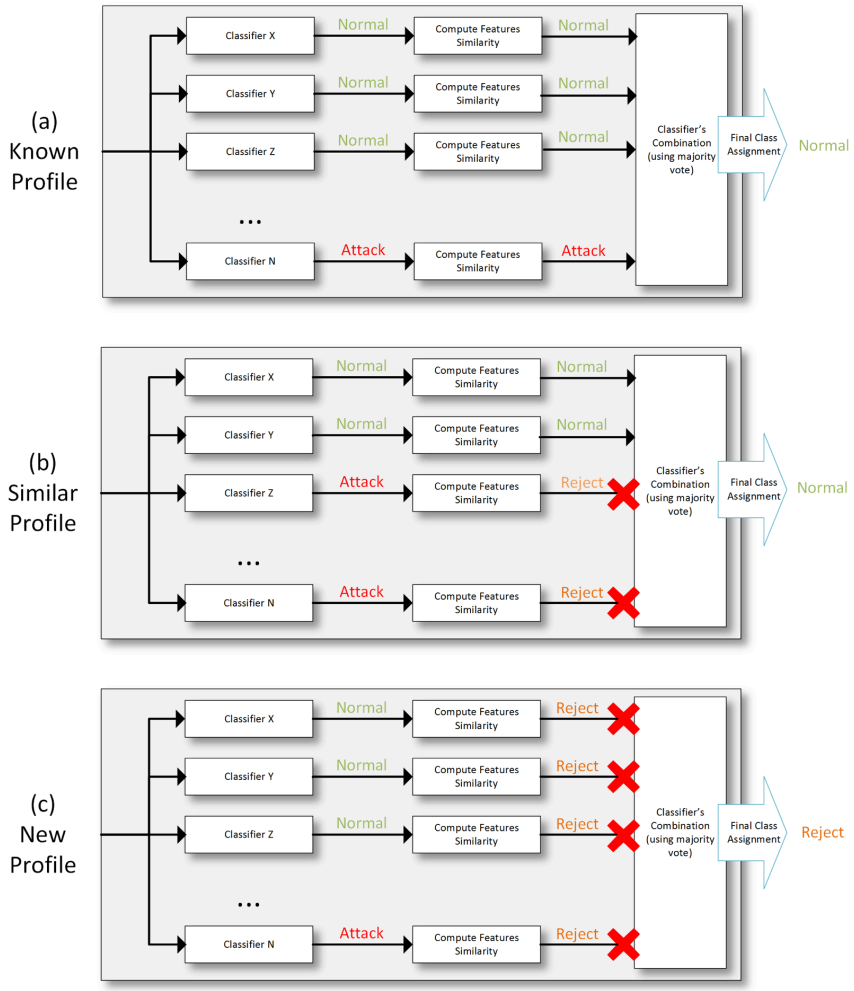


Figure 14.Final class assignment using majority vote as batch learner classifier combination.

4.5 *Stream Learning – Resilience to Adversarial Attacks*

When the batch learner rejects an event, it is deemed as a new behavior and thereby an unreliable classification. Unreliable events are forwarded to a stream learning classifier, which in its turn, is able to incrementally adapt to changes in network behavior over time. However, in order to incrementally update the model, such schemes, in general, rely on a supervised dataset, in which the event's labels must be known.

In contrast, in production environments, the obtainment of the network event's label can be a challenging task. Therefore, in such a case, one typically, resort to unsupervised stream learning algorithms. In general, those set of techniques typically, relies in a window-based approach, in which the update is performed based on time intervals (*sliding window*). In such a case, recent events are given a greater importance during the detection stage, whilst older events

are often discarded. Thereby, the model incremental update can be achieved without the assistance of an expert, but, on the other hand, it is prone to adversarial attacks. For instance, a sophisticated attacker, may change the attack behavior, either to evade the detection mechanism, or to pervert its properties, rendering model updates with misclassified instances.

In the light of this, the next subsections describe the design of an unsupervised stream learning approach for anomaly-based intrusion detection that can automatically update the intrusion detection engine over time, while still being resilient to adversarial attacks. Therefore, the proposed approach relies in a class-specific stream outlier detection algorithm to be resilient to both causative and exploratory attacks (Section 2.2.2). The proposed stream learning resilient to adversarial attacks is shown in Figure 15 and described in the next subsections.

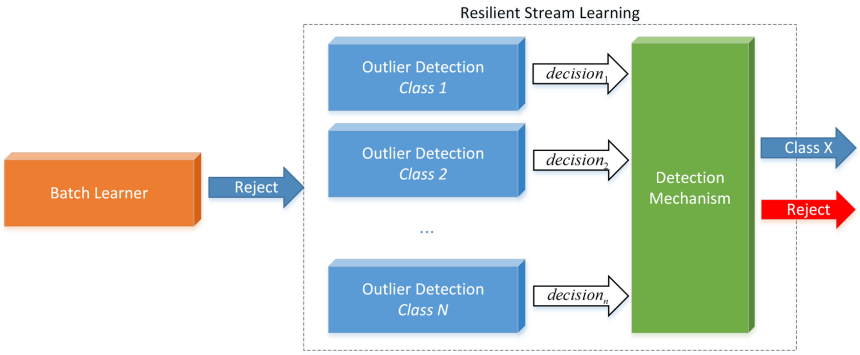


Figure 15. Resilient to adversarial attacks anomaly-based intrusion detection through stream learning algorithm.

4.5.1. Detection Scheme

The proposed adversarial resilient stream learning relies in a class-specific stream outlier detection algorithm. For example, an outlier detection for normal events and an outlier detection for attack events. The detection is performed accordingly to Figure 15, (i) the set of features are extracted from the considered event, e.g. a network packet; (ii) a feature vector is supplied to each outlier detection algorithm; (iii) each outlier detection perform its detection, assigning a class either outlier (event does not belong to outlier detection class group) or inlier (event does belong to outlier detection class group); (iv) the detection engine receives the decision from each outlier detection and attempts to find a consensus among the decisions; (v) if a decision unanimity is found the class is assigned, otherwise, the event decision is rejected.

When receiving an event decision, the detection engine decides whether the event classification is reliable or not. The class assignment reliability of an event classification (output

'class X' in Figure 15) comes from the nullity of intersection of decision from all classifiers. The reliability computation process is shown in Equation 7, where $decision_i$ denotes to each Outlier Detection classifier output.

$$\bigcap_{i=1}^{n}(decision_i) = \emptyset \quad \text{Equation (7)}$$

As an example, consider two outlier detection algorithms (Eq. 7), one for normal events and one for attacks; an event which is classified as an inlier for normal and outlier for attack is reliable – the decision is an unanimity, because there is not an intersection between classification classes in the different detection engine for the same event. However, an event which is classified as inlier for more than one outlier detection should be rejected, as the decision is not reliable. Rejected classifications indicates that a potential evasion attempt or a false alarm might be occurring and another detection mechanism should be used, for instance, a signature-based intrusion detection mechanism or manual inspection.

4.5.2. Ensuring Adversarial Machine Learning - Exploratory

Unlike the traditional stream learning algorithms, to provide resilience to exploratory attacks, the *immutable behavior* of each outlier detection algorithm is considered. The *immutable behavior* is defined by a restriction that do not allow an outlier to become an inlier in the outlier detection algorithm over time (in a considered sliding window). In such a case, it considers that in the anomaly-based intrusion detection field an event that is initially classified as outlier will not become an inlier at any moment in time. For example, an attack that was classified as an outlier (attack) by the normal outlier detection algorithm, must not be classified as a normal event afterwards, even if its occurrence increases in the sliding window over time.

By using the *immutable behavior*, the attacker will not be able to exploit the sliding window range to pervert (pollute) the classification of events being analyzed by an outlier detector. It is important to note that events classified as inlier continue to be added into the stream learning sliding window, thus the algorithm is still able to adapt to changes in the stream. But, the proposal mitigates a possible evasion attack, when the number of outlier events become predominant in a sliding window, therefore they will trigger the behavior mutation from outlier to inlier.

4.5.3. Ensuring Adversarial Machine Learning - Causative

To provide resilience to causative attacks, the proposal relies in both *immutable behavior* (Section 4.5.2) and class-specific outlier detectors. It considers that resilience to

causative attacks must be provided at two stages: *initial training* and *ongoing readapting* (retraining).

The *initial training* is related to the initial outlier detector sliding window population – the filling of events in a sliding window. In this stage, the outlier detectors sliding window are still being populated, thus susceptible to causative attacks. Thereby, the proposed approach assumes that at least there are an initial population allowing the correct classification for one outlier detector, since the sliding window will be updated according to the initial events. A way of assuring the reliability of the initial training is preset the sliding windows with a predominant number of copies of the same inlier event. Thus, given the outlier detector is reliable, the classification outputs can be trusted if decision unanimity is reached, otherwise the classification is rejected.

To provide a secure ongoing readapting, the proposal relies in both class-specific single class detection mechanism and *immutable behavior* (Section 4.5.2). The single class detection mechanism provides resilience to event behavior manipulation. For example, the attacker must manipulate the event behavior in a manner that it behaves as a normal event, while also being an outlier for the attack outlier detection mechanism. Whilst, the *immutable behavior* difficult the attack over the sliding window, since the attacker must have skills to manipulate the events in a manner that pervert all outlier's detectors.

4.6 *Reliable Learning – Conformal Evaluator*

The aforementioned approach provides resiliency to adversarial attacks for unsupervised stream learning algorithms. In such a case, the model update is performed without the assistance of an expert. However, when the supervised approach is considered, one must request the event's label over time, for update purposes. Thereby, a method must be designed to provide classification reliability in the presence of new network behaviors. This because, the stream learning model may become outdated, as it is prone to expert availability, for providing the event's label.

To provide reliable classifications over time, one must first evaluate the classification reliability degree. To this end, the classification reliability is assessed by the means of a conformal evaluator. The conformal evaluator aims at assigning a reliability degree for each classified instance. To achieve such task, the conformal evaluator computes two values: *Credibility* and *Confidence*. *Credibility* defines how well the instance fits into the assigned

class, while *Confidence* defines how well the instance does not fit into the other classes. Finally, a classification outcome is accepted according to Equation 8 and 9.

$$Reliability_{degree} = Credibility \times Confidence \quad \text{Equation (8)}$$

$$f(decision) = \begin{cases} unreliable, \ Reliability_{degree} < t_{class} \\ reliable, \ Reliability_{degree} \geq t_{class} \end{cases} \quad \text{Equation (9)}$$

In which the $Reliability_{degree}$ denotes the degree of reliability for the classified instance, obtained by multiplying the $Credibility$ and $Confidence$ values. Finally, a decision can be considered reliable if its $Reliability_{degree}$ is higher than a specific class threshold (t_{class}).

4.6.1. Computing Credibility and Confidence values

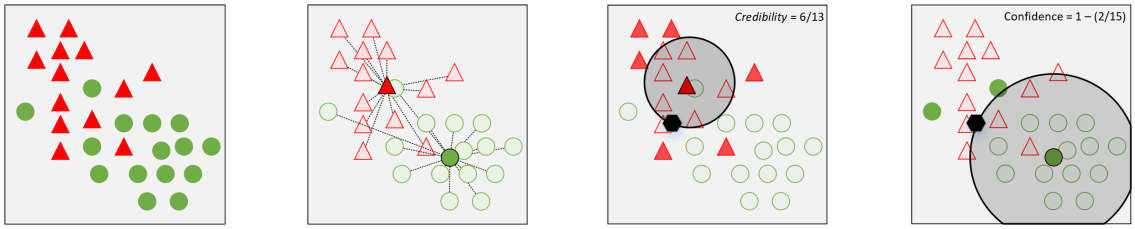
Reliable classifications can be defined as instances with a similar behavior to their assigned class, i.e. instance that presents a known/similar behavior as the ones present in the training dataset. However, in general, traditional ML confidence values does not measure the degree of similarity, but rather the classification correctness. For instance, the decision tree classifier confidence values are typically computed as the ratio of instances in the training dataset that were classified as belonging to the given class, i.e. the accuracy rate in the given tree node.

In contrast, to assess the reliability, one must first compute the *Credibility* and *Confidence* values. However, the computation of such values cannot be achieved by the means of a classification confidence, as it fails at providing reliable confidence values when new instance behavior is occurring.

In this sense, to compute the *Credibility* and *Confidence* values, one must rely in the assistance of similarity-based metrics. Similarity-based metrics enables the measurement of distance to a specific group, thereby, measuring how well the sample fits not only to a given class but also to the training dataset itself.

Figure 16 shows an example of how *Credibility* and *Confidence* values could be computed for a two-class dataset. The process occurs as follows: (a) A two-class training dataset of triangles and circles classes is considered; (b) The centroid for each class is computed; (c) Given a new example classified as triangle by a classifier, the *Credibility* value is computed as the ratio of triangles examples in the training dataset that falls outside the area between the

measured instance and the class centroid (triangle centroid); (b) *Confidence* is computed as the inverse of *Credibility* to circle class.



(a) Two-class training dataset

(b) Computation of class centroids

(c) *Credibility* computation for an instance classified as triangle

(d) *Confidence* computation for an instance classified as triangle

Figure 16. Computation example of *Credibility* and *Confidence* values through a similarity-based algorithm in a two-class dataset. In this example the *conformity* measure is given as the distance to centroid.

In the figure, the distance to class centroid is used as a similarity (*conformity*) metric. It is important to note, that other measures can be used as a *conformity* metric. For instance, considering a random forest classifier [70], one may use the ratio of trees that classified the instance as a given class as the *conformity* metric. On the other side, when one is using a support vector machine [71], the distance to the hyperplane can be used as a *conformity* metric. In other words, any metric that enables to measure similarity (*conformity*) for a given group can be used to assess the classification quality, when computing the *Credibility* and *Confidence* values.

4.6.2. Ensuring Reliability

After the computation of the *Credibility* and *Confidence* values a decision can be made regarding the reliability of a given classification (Eq. 9). To this end, a classification is considered reliable when the $Reliability_{degree}$ surpasses a reliability threshold (t_{class}) according to the class chosen by the classifier.

For the proposed conformal evaluator, reliability is reached if the classified instance is similar to the classified class (*Credibility*) while also not similar to other classes (*Confidence*). In this sense, in production, the conformal evaluator may face the following scenarios:

- *High Credibility, High Confidence (s1)*: instance is similar to classified class, and not similar to other classes. High $Reliability_{degree}$, event should be accepted, most likely a correct classification;

- *High Credibility, Low Confidence (s2)*: instance is similar to classified class and other classes. Low $Reliability_{degree}$, event will be accepted or not according to t_{class}. Could be a classification error or an instance similar to all classes;
- *Low Credibility, High Confidence (s3)*: instance is not similar to given class or other classes. Low $Reliability_{degree}$, event will be accepted or not according to t_{class}, most likely it is a new behavior;
- *Low Credibility, Low Confidence (s4)*: instance is not similar to classified class, and similar to other classes. Low $Reliability_{degree}$, most likely a classification error, event should be rejected;

In this manner, in scenario s1, events can be accepted while maintaining the classifier reliability, as they present a similar behavior to the training dataset. However, in scenarios s2, s3, and s4, events should be rejected or not according to the administrator needs. In such cases, a low t_{class}, may imply in accepting more instances, which could either be a new behavior or an instance with a similar behavior to all classes, but not necessarily an error. Thereby, a series of evaluation tests must be performed to establish the t_{class} values.

The proposed conformal evaluator aims at addressing the challenge of ensuring reliability in face of new network traffic behavior. However, the occurrence of a high rejection rate over time to provide reliability may inhibit the usage of proposed reliable intrusion detection model in production. To this end, the detection scheme must be updated according to the incoming network traffic. The next subsection further details how such task can be achieved, when supervised stream learning algorithms are used.

4.7 *Reliable Learning – Adapting to Network Behavior Changes*

By the means of the proposed *Classification Reliability Assessment* method, it becomes possible to measure how reliable the classification is. However, the classification task is still necessary. In this sense, provide an ongoing updated classification scheme is a challenging task. To this end, the proposed approach combines several ML techniques, as shown in Figure 17. The proposed technique aims at providing the following properties:

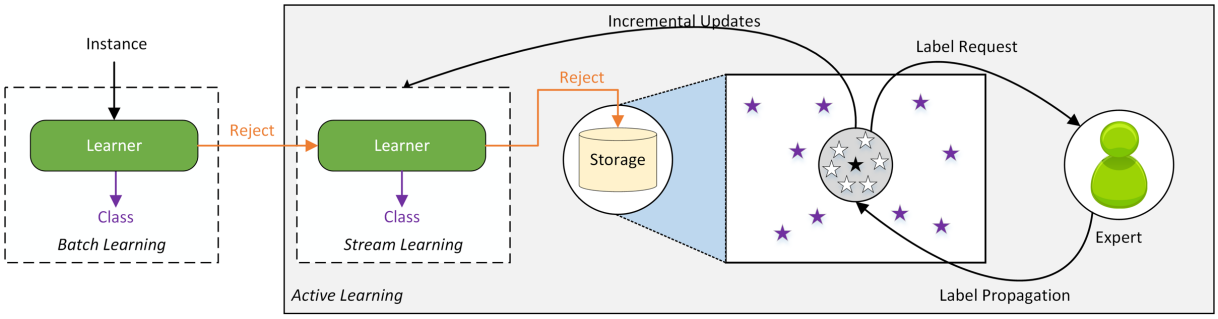


Figure 17. Proposed reliable learning architecture.

- *Reliability*: classifications are coped with conformal evaluator. Only reliable decisions are accepted;
- *Decrease Rejection Rate*: to provide reliability one must reject unreliable instances, which could lead to unfeasible IDS over time in production usage. In order to further decrease the rejection rate, despite updating or not the detection scheme, the proposal employs batch and stream learners in a cascade pipeline classification approach;
- *Reliable Update Classification Scheme*: the scheme reliable and ongoing update is enabled by employing Batch and Stream Learning algorithms coped together. The Batch Learning is not updated over time to ensure that the system maintain its reliability. On the other side Stream Learning is incrementally updated with expert assistance, to address changes over time in network behavior;
- *Decrease Expert Label Request*: rejected instances are stored and their label periodically requested. To significantly decrease the expert label request a label propagation approach is employed. The assumption is that nearby instances belong to the same class as given by an expert;
- *Decrease Storage*: the number of rejected instances that needs to be stored is significantly reduced by employing stream and batch learners, and updating the detection scheme over time;

The next subsections further describe each of the detection scheme properties, and how they are achieved.

4.7.1. Reliability

Reliable classifications can only be assured with the assistance of a conformal evaluator. Thereby, a conformal evaluator is coped together with stream learner. The objective is to either accept a reliable classification, or reject unreliable ones. When an instance is rejected it is

assumed to be unreliable and should be stored. Stored instances can be used afterwards, for incremental model updates.

4.7.2. Decrease rejection rate

A high number of rejected instances may lead the proposed reliable intrusion detection model to be unfeasible for production usage, since rejected instances must be stored and manually inspected. Thereby the proposal aims at decreasing the rejection rate by applying incremental model updates. In addition, batch and stream learners are used to decrease the system rejection rate in a cascade manner.

On the other hand, incremental model updates are performed ongoing. This enables the reduction of instances with unknown behaviors (Section 4.6.2, scenario 3), thereby further decreasing the rejection rate.

4.7.3. Reliable update classification scheme

The reliable and ongoing update of the classification scheme is achieved by the means of incremental model updates over the stream learning algorithm. The update process is as follows: (i) a subset of rejected instances is periodically obtained; (ii) the label of chosen instances are requested to an expert; (iii) the labels are propagated to nearby unlabeled instances; (iv) Stream Learning algorithms and their respective conformal evaluators are incrementally updated with labeled instances; (v) the subset of now labeled instances are removed from storage.

The usage of stream learning algorithm enables the ongoing update of the detection system. The conformal evaluator can also be incrementally updated, according to the used similarity-based algorithm. For instance, the incremental update of the conformal evaluator shown in Figure 16 can be achieved by updating the classes centroid, and adding the new instances into the dataset.

4.7.4. Decrease expert label request

The scheme assumption is that instances can be automatically labeled by the means of a label propagation technique, significantly reducing expert label requests. Thereby, to address possible label tagging errors introduced by such an approach, the approach relies in the batch learner. The batch learner is not updated throughout time in the proposed scheme. In this manner, possible mislabeled instances can only be introduced to the system if they are events previously rejected by the batch learner, i.e. the scheme maintain the detection reliability over

known behaviors, while also reliably adapts to new ones by the means of the stream learning algorithm.

However, one must note a tradeoff between expert label request and the label propagation error. This because the radius increase in the label propagation process, may introduce mislabeled instances to the stream learner. On the other side, a small label propagation radius, may significantly increase the expert label request periodicity and the required storage for rejected instances. Thereby, the tradeoff between such radius increase must be evaluated.

4.7.5. Decrease storage

Reliability is achieved by the means of rejecting unreliable classifications. However, the proposed scheme stores rejected instances until their label is known with the assistance of an expert. Thus, a high rejection rate can lead to an unfeasible amount of needed storage. In this sense, to decrease the needed storage, one must both reject less instances and also remove them from storage as soon as possible.

Therefore, rejection of less instances is achieved by ongoing updating the detection scheme over time. On the other side, the fast storage cleanup is achieved by the means of label propagation technique.

4.8 *Discussion*

A reliable near real-time network traffic classification for high-speed networks, namely reliable intrusion detection model was presented. Although the proposal is based on the use of existing machine learning techniques, it aim at addressing the open challenges in network-based intrusion detection systems (NIDS). To this end, the proposal checks whether the classifications outcome should be accepted or not, by the means of the conformal evaluator. When an event is rejected, this indicates with high probably that a new network traffic behavior is occurring. Although classification rejection was used in other areas where errors have a high cost (e.g., optical character recognition (OCR) [59] or medical diagnosis [56]), The proposal is the first intrusion detection system to exploit this technique in the context of assessing classification reliability over time. Second, the proposal employs stream learning and batch learning techniques coped together to analyze traffic in near real time. The goal is to support incremental model updates based on the rejected instances, while also maintaining its reliability over time. The expectation is that after a period (e.g., within one week), the rejected event can be properly classified by an expert or a tool (signature-based NIDS) based on public information (e.g., new indicators of compromise). At this point the proposal is able to incorporate the new knowledge

in the attack model. A major benefit is that the model updates are based only on correctly classified events, decreasing the risk of inaccurate detections, which may lead to high false positive rates while processing further packets.

Rejecting low-confidence classifications in a NIDS – the key idea of this work –lead to two important benefits: better detection accuracy (i.e., fewer misclassifications) and the identification of new characteristics of the evolving traffic, which is then used for updating the classifier model. These benefits improve the proposal reliability over time, even with changing network traffic behavior, while also greatly decreasing the amount of computational and storage resources needed to operate the system.

Chapter 5

The Building of Realistic Intrusion Datasets

In order to properly evaluate the reliable intrusion detection model, one must first have properly built intrusion datasets. However, one of the main issues regarding such task concerns to the lack of availability of such data.

Therefore, this chapter addresses the building of proper intrusion datasets, more specifically, two approaches for intrusion dataset building are presented and evaluated. First, a fine-grained intrusion dataset, obtained by the means of building a controlled environment is presented. The fine-grained intrusion dataset aims to enable the building of generalization capable models (Section 4.3), the evaluation of the classification reliability (Section 4.4), and the building of unsupervised stream learning algorithms resilient to adversarial attacks (Section 4.5). Second, an approach for building an evolving intrusion dataset is presented. The built evolving dataset, namely *MAWIFlow*, is a breakthrough in the intrusion detection community as the most comprehensive intrusion dataset currently available. *MAWIFlow* aim to enable the evaluation of the reliability of classifications over time (Section 4.6), and the proposed reliable intrusion detection model (Section 4.7).

The fine-grained intrusion dataset building method and the controlled environment setup is described in Section 5.1. *MAWIFlow* building task and its characteristics are presented in Section 5.2. Finally, a discussion is made in Section 5.3.

5.1 *Fine-grained Intrusion Dataset*

The proposed fine-grained intrusion database creation method aims to ensure that the database contains the properties expected from an IDS testing intrusion database. To achieve this, the proposed method creates intrusion databases in a controlled environment and

reproduces a user's behavior through well-known tools. The method considers two different users: the legitimate client and the attacker. The traffic is generated considering the client-server model. For generating the server-side network traffic, the honeypot technique is used, whereas the client traffic is generated through real-world workload tools. Thus, real and valid traffic is generated for the client-server communication. The attacks are generated using known, standardized, and widely used tools frequently implemented for system auditing. The traffic creation method overview is shown in Figure 18 and further described in the following sections.

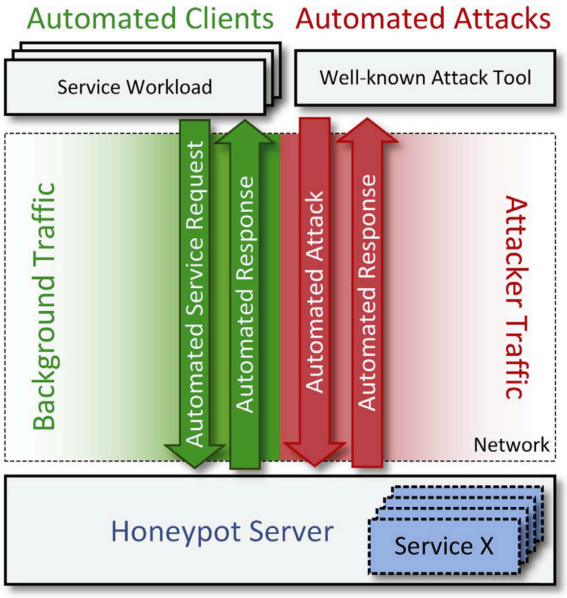


Figure 18. Proposed database creation method for fine-grained evaluation of intrusion detection systems.

5.1.1. Normal (background) traffic creation method

It is recommended that the traffic included in an intrusion database used for ML-based detection systems be real and valid. Thus, the method used for its design must ensure that the client-server interaction occurs correctly, thereby guaranteeing that the client behavior evidenced in the database is similar to that observed in the real world.

The normal (legitimate) traffic must be generated according to two perspectives: the client and the server. The client is responsible for requesting the services available on the server, whereas the server is responsible for providing the appropriate response to each request in terms of content and service behavior. It is expected that the provided services, as well as their requested contents, are highly variable.

Independent of the considered application, each user network traffic behavior, in general, is random and does not follow a statistical distribution when compared to that of a different application user [72, 73]; for instance, the behaviors of two different users browsing a Web application are not necessarily similar. Thus, the proposed method generates the normal

(background) traffic by providing a set of services, where each service has a set of contents that may be requested. Each client performs a real and valid request through a real-world workload tool; thus, real, and valid requests are generated for the client side. Each client sends a request for a previously defined service and a specific service content. After the client-server communication, the client waits a variable time and transmits a new request for another service. Thus, the client behavior is modeled according to the observed application usage, e.g., a client browsing a Web page for a certain duration.

In turn, the server must be able to interpret the received request and generate the appropriate reply. The use of tools specific to the service being provided makes it difficult to update the intrusion database, whereas the use of a technique that mimics the server responses allows the database to be updated easily. Thus, the honeypot technique was considered, which allows the responses of a vulnerable server to be mimicked, with automated and valid responses to be generated.

A set of predefined services are provided. Thus, every request, regardless of the requested service, is correctly interpreted and receives a legitimate reply. Under this assumption, the proposed fine-grained intrusion dataset creation method generates real, valid, and easy-to-update background traffic. The complete background traffic generation process is shown on the left side of Figure 18.

5.1.2. Attack traffic creation method

The lack of an appropriate implementation guarantee is the main problem that has been reported to occur during attack generation. In general, researchers (see, e.g. [21]) implement a known attack according to their discretion, which makes the attacks difficult to reproduce as there is no guarantee that the implementation follows a well-known defined standard.

Immediately after a new attack becomes known and is reported, entities specify it. Initiatives such as the Common Vulnerabilities and Exposures (CVE) include the details of new vulnerabilities and the affected services. Implementations that are, for example, CVE-compatible, guarantee that an attack will behave as expected (reported). Thus, tools that follow well-known standards are auditable and can be assessed.

The fine-grained intrusion dataset creation approach, unlike those where the authors implemented their own version of the attacks, is based on the use of well- known and de facto standardized tools to generate the attacks. This approach ensures that the implementation of all the attacks included in the database is dependable when they become public, as the approach

follows a well-known standard. The complete attack traffic generation process is shown on the right side of Figure 18.

5.1.3. Dataset Building

The following sections discuss the details of the application of the described fine-grained intrusion dataset creation method. An extensive description of the background and attack traffic generation is provided. Then, the testbed network infrastructure is discussed in detail.

5.1.3.1 Background Traffic Generation

The most desirable property of an intrusion database is that the background traffic is as realistic as possible. The normal traffic must be highly variable, real, and valid. However, background traffic generation is a complex and difficult task, mainly because of the complexity involved in modeling user behavior [74], which is, in general, random and application-dependent. The network traffic generated is dependent on the user demand for an application and is specific to the environment being reproduced.

Taking these factors into account, each client is treated as an entity with a pseudo-random behavior that does not follow any statistical distribution pattern, as reported in [48]. Each client shows a unique behavior when requesting a service. Each client might request one or more services.

To achieve this property, a set of predetermined services on the basis of the frequently used services discussed in [75] were established. The following services were considered to be generated on the testbed environment: HTTP, SNMP, SMTP, NTP, and SSH. Every name resolution (DNS) was also generated as a consequence of using the listed protocols.

To create the honeypot server (Figure 18), which executes the server-side applications, in the proposed method the Honeyd [76] tool was used. To develop the client-side application for use with the servers, a workload tool was used as the service request tool. It is important to emphasize that the only purpose of using a workload tool was to generate valid and real requests.

Each client requested the services hosted on the honeypot server; the emulated services and their client behaviors are described in Table 4. To ensure traffic variability, each client randomly varied the requested content, according to the description shown in Table 4, and the time between the requests varied from 0 to 4 s. The variation in the time between each request and in the requested content was designed to mimic the non-modellable behavior of clients. By

using this method, it became possible to simulate a user browsing a Webpage and also sending an e-mail, for instance.

Table 4. Services used for the background traffic generation.

Service	Description (Client behavior varying from 0 to 4 s interval)
HTTP	The 1000 most visited Websites worldwide were downloaded using www.alexa.com/topsites and hosted on the honeypot server; each HTTP client requests a pseudo-random Website from this set of contents.
SMTP	Each SMTP client sends a mail with 50-400 bytes in the subject line and 100-4000 bytes in the body.
SSH	Each SSH client logs in to the honeypot host and executes a random command from a list of 100 possible commands.
SNMP	Each SNMP client walks through a predefined management information base (MIB) from a predefined list of possible MIBS.
NTP	The client performs time synchronization through the NTP protocol.
DNS	Every name resolution was also made to the honeypot server

Every client generated a real and valid request for a service and received a real and valid reply from the honeypot server. Thus, all the generated background traffic was real and valid. Finally, to mitigate possible repetition in the generated traffic, each scenario was executed for 30m, a reasonable time considering the request variability shown in Table 4. To allow the scenarios to be reproduced, the behavior for each client was logged.

5.1.3.2 Attack Traffic Generation

For the attack traffic generation, the taxonomy adopted by Kendall [19] was considered. To validate the proposed method, two groups of attacks were used as baseline attacks: probing and DoS. The attacks and tools used and their descriptions are listed in Table 5.

Each attacker generated a specific attack type (Table 5), and to make the attacks highly variable, each attacker varied the frequency and duration during each testbed. A single machine generated each attack, allowing the automatic class labeling based on the network packet source IP address. It is important to note that this approach does generate environment-specific features; e.g., on the basis only of the IP address, an ML-based system can identify every attack. Thus, the system being evaluated using the fine-grained intrusion database must be aware of this restriction and should not use any environment-specific features, such as the time-to-live (TTL) and IP address fields[2].

Table 5. Tools used for attack network traffic generation.

Category	Attack Type	Tool Used	Description
DoS	SYN flood	Hping3	Sends several requests to open TCP connections, varying the attack send frequency and the duration time
	UDP flood	Hping3	Sends several UDP datagrams to an open DNS port, varying the attack send frequency, the duration time, and the size of each datagram
	ICMP flood	Hping3	Sends several ICMP messages to the target, varying the attack send frequency, the duration time, and the size of each datagram
	TCP keepalive	Slowloris	Initiates several HTTP connections and keeps them open for a period, varying the number of connections to be opened

[2] The list of features extracted in the fine-grained intrusion dataset can be found in Appendix 1

	SMTP flood	Postal	Sends several emails to an SMTP server, varying the duration time, body size, subject size, and frequency
	HTTP flood	LOIC	Sends several HTTP-get requests to a specific URL, varying the duration time and frequency
Probing	UDP scan	Nmap	Searches for open UDP ports, varying the attack send frequency and the duration time
	SYN scan	Nmap	Searches for open TCP ports by sending TCP packets with the SYN flag set while varying the attack send frequency and the duration time
	NULL scan	Nmap	Searches for open TCP ports by sending TCP packets without any flags set while varying the attack send frequency and the duration time
	TCP connect	Nmap	Searches for open TCP ports by completing the three-way handshake while varying the attack send frequency and the duration time
	FIN scan	Nmap	Searches for open TCP ports by sending TCP packets with the FIN flag set while varying the attack send frequency and the duration time
	XMAS scan	Nmap	Searches for open TCP ports by sending TCP packets with the FIN, PSH, and URG flags set while varying the attack send frequency and the duration time
	ACK scan	Nmap	Searches for open TCP ports by sending TCP packets with the ACK flag set while varying the attack send frequency and the duration time
	OS Fingerprint	Nmap	Identifies the OS from the target (https://nmap.org/book/osdetect.html) while varying the attack send frequency and the duration time
	Service Fingerprint	Nmap	Identifies the services and their versions from the target (https://nmap.org/book/man-version-detection.html) while varying the attack send frequency and the duration time
All	Vulnerability Scan	Nessus	Identifies service-level vulnerabilities while varying the attack send frequency and the duration time

5.1.3.3 Testbed environment

The scenarios, which are described in more detail below, were composed of 100 interconnected client machines. The number of clients was established to maximize the possible client behaviors (Table 4). Each client was an Ubuntu 16.04 machine; the network traffic was dependent on the workload tool used according to the service being requested. A single honeypot server was used in each scenario. The honeypot server was implemented using the Honeyd 1.5c tool, installed on an Ubuntu 16.04 machine, mimicking a vulnerable Ubuntu 14.04 server. The attacker machines ran Kali Linux version 2.0; 16 machines were used to generate the attacks, with each attacker machine generating a single type of attack (Table 5).

A single LAN network running at 100 Mbits/s connected the machines. The defined LAN network speed allowed the generated traffic to be recorded on a single machine without dropping packets or mirroring the traffic [36]. All legitimate requests and attacks were generated against the honeypot server (Figure 18); the generated traffic was stored on the honeypot server.

The establishment of a single LAN network allowed the creation and replication of the proposed scenarios to be simplified. The definition of more complex scenarios [6, 36] would hamper the replication of the proposed method. In the next sections, each of the created scenarios is further described.

a. Attack detection scenarios

As stated in Section 5.1.3.2, two attack categories were defined as the baseline attacks: probing and DoS. Thus, six scenarios were defined to generate databases to evaluate the attack detection rate (Figure 7, Attack Detection Evaluation).

In the attack detection scenarios, each client was responsible for generating the background traffic for each client service request, as shown in the Venn diagram in Figure 19. The overlapping circles denote the clients that generated both services. The distribution among clients and services was designed to simulate the traffic distribution described in [75].

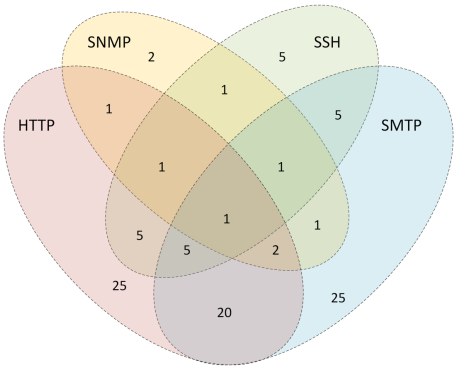


Figure 19. Venn diagram of background service distribution among clients.

The background network traffic remained immutable at the content request and client level. However, the attacker traffic varied according to each scenario, as described in Section 4.3.1. For each considered attack type, i.e., probing and DoS, three scenarios were created. Each scenario was run for 30m. The attacks started at the 10th minute and lasted for 15 m (scenario time: the 10th to the 25th minute), following the attacker behavior described in Table 5. Thus, it was possible to capture the environment behavior without, with, and after the attacks. The network traffic distribution and the attacks used for each scenario are shown in Table 6.

Three levels of attack similarity were defined in the databases: network-level vulnerabilities, service-level vulnerabilities, and service-level exploitation. The first scenario was named known. The purpose of the known scenario was to generate the classifier model and to define the known detection rate while detecting only the known attacks. Thus, only attacks at the network level, focusing on network protocol vulnerabilities, were generated. The similar databases contain attacks with a similar behavior but focusing on service-level vulnerabilities. Finally, the new database has a new type of attack that focuses on service exploitation.

Table 6. Network traffic distribution for attack detection scenarios.

Scenario	**Attacks Generated**	**Traffic (Network Packets)**			**Size (MB)**
		Background	***Attack***	***Total***	

Probing (Known)	UDP scan, SYN scan, NULL scan, TCP connect, FIN scan, XMAS scan, and ACK scan	28,618,365	36,628	28,654,993	8.476
Probing (Similar)	OS fingerprint and service fingerprint	28,477,884	10,441	28,488,325	8.499
Probing (New)	Vulnerability scan	28,391,914	17,753	28,409,667	8.512
DoS (Known)	SYN flood, UDP flood, ICMP flood, and TCP keepalive	26,747,521	761,269	27,508,790	7.945
DoS (Similar)	SMTP flood and HTTP flood	40,278,594	26,390,723	66,669,317	12.143
DoS (New)	Vulnerability scan	27,522,317	3,429	27,525,746	7.265

The created attack databases mimicked the behavior seen in production environments. Normally, when developing an ML-based NIDS, only attacks detectable by an NIDS are included in the training dataset. However, when used in production environments, the system will face a wider range of attacks. The databases were created according to the method described in Section 5.1.3 and were used to validate the evaluation method described in Section 4.3.

b. Background detection scenarios

The background detection rate scenarios were generated using the attacks as the baseline. Thus, two sets of attacks were used, each generated separately, resulting in different databases, allowing correct method evaluation. The sets of attacks used consisted of the probing (known) and DoS (known) attacks, shown in Table 6.

To generate the services' detection databases, the services were divided into two groups: known and new services. The known services served HTTP and SNMP clients and the new services served SMTP, NTP, and SSH clients. Each scenario was executed for 30m; the distribution of clients followed the Venn diagram shown in Figure 20.

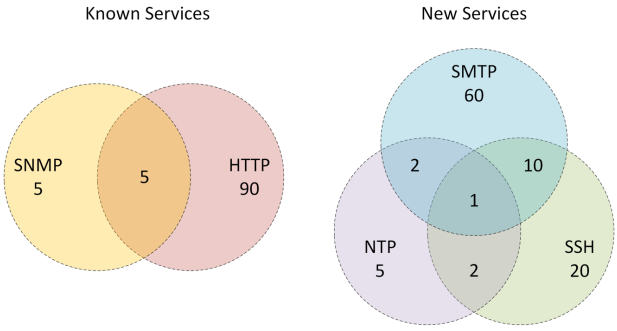


Figure 20. Venn diagrams for the background service detection rate scenarios showing the service distributions among clients.

Finally, to generate the content detection databases, the behavior of each client was modified. Three different behaviors were defined, divided into known, similar, and new content request behavior. The client's distribution followed the Venn diagram shown in Figure 19, and the client's behavior for each scenario is described in Table 7. The traffic distribution for each background detection database is shown in Table 8.

Table 7. Client behavior for service's content detection scenarios.

Service	Database		
	Known Contents	***Similar Contents***	***New Contents***
HTTP	Request a Webpage from 1 to 200	Request a Webpage from 1 to 500	Request a Webpage from 501 to 1000
SMTP	Each SMTP client sends a mail with 50-400 bytes in the subject line and 100-720 bytes in the body	Each SMTP client sends a mail with 50-400 bytes in the subject line and 100-1800 bytes in the body	Each SMTP client sends a mail with 50-400 bytes in the subject line and 1801-4000 bytes in the body
SSH	Each SSH client logs in to the honeypot host and executes a random command from a list of 20 possible commands	Each SSH client logs in to the honeypot host and executes a random command from a list of 50 possible commands	Each SSH client logs in to the honeypot host and executes a random command from a list of 50 never-seen commands
SNMP	Each SNMP client operates through a predefined MIB from a predefined list of possible MIBS	Each SNMP client operates through a predefined MIB from a predefined list of possible MIBS	Each SNMP client operates through a predefined MIB from a predefined list of possible MIBS
DNS	Every name resolution is also defined in the honeypot server		

Table 8. Network traffic distribution for backround detection scenarios.

Database	***Scenario***	***Generated Attacks***	***Traffic (Packets)***			***Size (MB)***
			Background	***Attack***	***Total***	
Service Detection	*Known*	*Probing (Baseline)*	6,260,424	34,239	6,274,663	731
	New		36,807,285	37,973	36,845,258	12,325
	Known	*DoS (Baseline)*	6,874,239	753,838	7,628,077	972
	New		37,273,872	812,384	38,086,256	12,738
Content Detection	*Known*	*Probing (Baseline)*	25,240,803	35,782	25,276,585	7,909
	Similar		26,216,937	36,245	26,253,182	7,959
	New		30,600,739	38,235	30,638,974	8,905
	Known	*DoS (Baseline)*	27,376,278	746,287	28,122,565	8,782
	Similar		28,241,742	784,972	29,026714	8,932
	New		33,235,457	797,728	34,033,185	9,748

5.1.4. Discussion

The proposed fine-grained intrusion database creation method allowed real and valid network traffic to be generated, as the honeypot generated valid replies to each of the received requests. The event classes were automatically defined (labeled as a feature vector), to avoid manual labeling and to provide an error-free approach because of the number of packets to be evaluated. The automatic labeling was determined according to the source IP address for each network packet. This was possible because the attacker's machine generated only attack content. Additionally, the use of manual class labeling or clustering techniques [77] was avoided, reducing the labeling error.

Low variability or repeated traffic occurrences were mitigated, as the client content requests, the time between each request, and the requested application were varied. The use of well-known tools allowed the databases to be updated at each new vulnerability (attack) report, as the used tool became responsible for the network traffic update to ensure the correct attack implementation. To allow the deployed scenario to be reproduced, every client and attacker behavior was logged. Finally, privacy problems did not occur, because the databases were obtained in a controlled environment and the generated network traffic did not include any sensitive data.

Using the proposed fine-grained intrusion database creation method, it was possible to create 16 databases (Tables 6 and 8), each of which was aimed to validate the common assumptions adopted in the literature [6, 7], as described in Section 4.3, and to present network-specific detection rates.

The next section describes the steps involved in creating an intrusion dataset for the evaluation of ML-based intrusion detection techniques over time.

5.2 *An Evolving Intrusion Dataset*

To benchmark the behavior of ML-based IDS over time, a dataset, named *MAWIFlow*, with labeled (i.e., classified as either Normal or Attack) real network flows collected over 10 years is built. Similarly to the fine-grained intrusion dataset, *MAWIFlow* also fulfill a number of requirements, including be realistic and highly variable, prior labeled with correctly classified events, reproducible, and publicly available [36].

MAWIFlow is based on real and publicly available network traffic. More specifically, it is based on the network flows extracted from the MAWI network packets traces [16], collected daily for a 15-minute interval, from a transit link between Japan and USA. The labeling of records is achieved through MAWILab [15], which labels the daily anomalous events (network flows) from MAWI through a combination of several state-of-the-art unsupervised anomaly detectors. For the purpose of this dissertation, the network traffic captured in a 10-year range, from 2007 to 2016 was considered.

The *MAWIFlow* dataset is built through *BigFlow* feature extraction tool (see Section 4.2) [60] which extracts 158 host-based and flow-based features, some of which have been employed in previous works, being them: 15 features from Orunada et al. [78], 21 from Nigel et al. [79], 60 from Moore [80], and 62 from Viegas et al. [5]. Host-based features are features extracted according to the communication between two hosts, e.g. bytes sent/received in the last 15 seconds. On the other side, flow-based features refer to the features extracted according to the communication from a specific host, e.g. average packet size sent from host[3].

For the label assignment process, *MAWIFlow* assigns the label that was associated to the flow from which the feature was extracted. Table 9 shows a summary of the *MAWIFlow* dataset. Figure 21 shows the network distribution over time, and the distribution amongst the

[3] The complete list of features for each view can be found in Appendix I

classes. As can be seen, it contains over 28 billion network flows, extracted by analyzing more than 138 billion network packets from 10 years of real network traffic.

Table 9. *MAWIFlow statistics.*

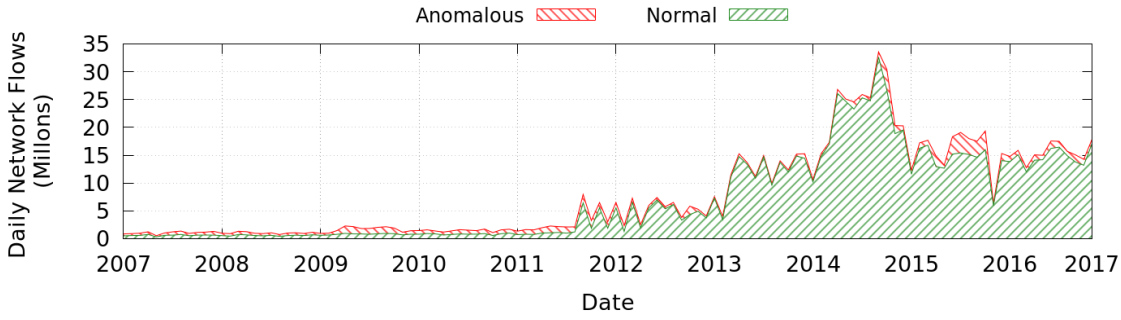
Field	Value
Average Daily Network Packets	~37.8 Millions
Average Daily Network Flows	~7.7 Millions
Average Daily Anomalous Flows	~1.7 Millions
Average Daily Dataset Size	~8.41 GB
Total Network Packets	~138.64 Billions
Total Network Flows	~28.34 Billions
Total Dataset Size	~30 TB

The original *MAWIFlow* dataset is composed of over 30 TB of data. Thereby, a stratification process was needed to reduce its size, enabling its sharing and facilitating its use for NIDS evaluation. Thus, the proportional random stratified sampling without replacement method [81] was employed to generate the stratified *MAWIFlow* dataset. The resulting dataset comprises just one percent of the original dataset, while maintaining the original proportion amongst the network traffic classes (Normal and Attack), which are randomly chosen.

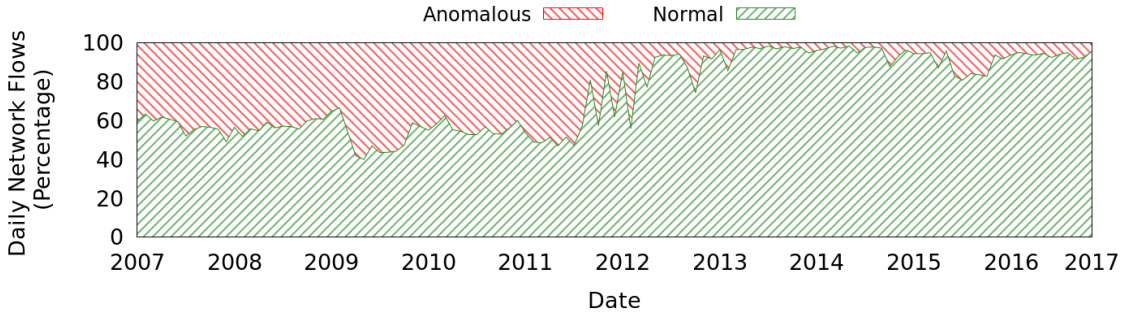
Besides being the first dataset of this kind publicly available, *MAWIFlow* overcome the main challenges faced when building realistic datasets for benchmarking intrusion-detection engines. More specifically, it presents all the desired properties described in [36]:

Realistic: the network traffic used for its building was obtained from real network traces. Moreover, *MAWIFlow* was built from over 10 years of real network traces, enabling not only evaluation of the detection system during a specific period of time, but also to evaluate how it behaves over time, when facing new network traffic behavior;

Valid: the network traces used for the *MAWIFlow* building were gathered 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;



(a) Daily number of network flows in *MAWIFlow*



(b) Daily distribution of network flows in *MAWIFlow*

Figure 21. *MAWIFlow* network traffic distribution throughout 10 years.

Prior labeled: event labels were identified by state-of-the-art unsupervised machine learning techniques (assessed by MAWILab). In this manner, supervised ML techniques can be evaluated regarding their performance using unsupervised techniques as their baseline performance;

Highly Variable: MAWIFlow is highly variable not only due to the used network traces but also due to its long period of recording. The used network traces are real, valid, and gathered from a real network infrastructure, thereby it presents the expected variability from production environments. Nonetheless, due to its long period of recording (from 2007 to 2016), the detection system can be evaluated considering the environment variability for 10 years;

Reproducible and Publicly Available: the used network traces were gathered from publicly available sources (MAWI). Moreover, *BigFlow* [60] (Section 4.2) source code is also publicly available.

MAWIFlow, is the first of its kind to provide means of assessing the reliability of ML-based IDSs. It presents all the expected properties from intrusion detection datasets [36], while also span for 10 years of real network traffic, resulting from the analysis of over 30 TB of data.

Chapter 6

Experiments

The experiments are divided according to the methods presented for the reliable intrusion detection model (Chapter 4).

More specifically, in Section 6.1 the evaluation of reliable batch learning algorithms is performed. The first experiment, in Section 6.1.1, aims at evaluating the reliability of traditional batch learning building methods regarding their generalization capacity, and the proposed approach for building generalization capable models. After, the method for providing reliability in classifications for batch learning classifiers is evaluated in Section 6.1.2.

In Section 6.2, the proposed stream learning approach resilient to adversarial attacks is evaluated. The proposed approach is compared to both traditional batch and stream learning techniques in the fine-grained intrusion dataset.

Finally, Section 6.3 addresses the evaluation of the reliable intrusion detection model over time. Therefore, the first experiment evaluates traditional intrusion detection techniques regarding their reliability over time in *MAWIFlow*. After, the proposed conformal evaluator is evaluated in Section 6.3.2. Finally, the method to reliably adapt to network behavior changes over time is evaluated in Section 6.3.3.

The proposed *BigFlow* is evaluated regarding its scalability and throughput in Section 6.4.

6.1 *Batch Learning*

This section presents the evaluation regarding the building of reliable batch learning models for intrusion detection in a twofold manner. First, the proposed approach for building generalization capable batch learning models is evaluated (Section 6.1.1). Second, in order to address the changes in network traffic behavior over time, the proposed approach for providing reliability in batch learning models classifications is evaluated (Section 6.1.2).

6.1.1. Generalization

The method for building generalization capable models was evaluated by the means of the fine-grained intrusion dataset (see Section 5.1).

Because of how the classes are disposed in the created datasets, a stratification process was used so that each class was equally represented in the training, testing and validation datasets (Tables 6 and 8). The stratification process consisted of randomly selecting 25% of the events from the class with fewer occurrences; then, the same number of events were randomly selected from the other classes. The datasets were obtained using this stratification process, with 25% of the events being used for training, 25% for validation, and the remaining events for testing.

For the model building process, the Weka framework version 3.8.0 was used [82]. Two batch learners were used during the evaluation tests: naïve Bayes (NB) and decision tree (DT). For the NB classifier, all numerical attributes were discretized according to the method of Fayad and Irani [83]. The C4.5 DT algorithm was used with a confidence factor of 0.25.

6.1.1.1 Model Evaluation

To evaluate the proposed batch learner evaluation method towards generalization capable models (Section 4.3, Figure 7) and the multiple objective feature selection method for building generalization capable models (Section 4.3), two classifiers were used: DT and NB. During the evaluation, the FP denotes the number of normal instances (normal client network packets) wrongly classified as attacks, whereas FN is related to the number of attack instances wrongly classified as normal.

To obtain the generalization capacity (Figure 7, Generalization Evaluation), the publicly available DARPA1998 database [20] was used. Despite its well-known problems [20, 21], DARPA1998 is still extensively used in studies in the literature (see, e.g., [31]) and provides a reasonable benchmark for research studies. The database consists of a nine-week air force

environment simulation. The data of the first seven weeks are used for training and of the last two weeks for testing. For each day, DARPA1998 provides, among other files, a *tcpdump* file containing the network packets and a description file describing the classes for each connection.

For the evaluation tests, only the DARPA1998 training data were used; the feature extractor was modified to label the network packet classes according to the description file[4]. The connection classes were divided according to Kendall's [19] taxonomy; two attack groups were considered: probing and DoS. Table 10 presents the network traffic distribution for the used classes in DARPA1998.

Table 10. Traffic distribution on DARPA1998.

Category	Class	Number of packets (representativeness)
All	Normal	28,426,093 (94.96%)
DoS	Synflood — Neptune	1,507,319 (5.04%)
Probing	Port scan — Nmap	2,211 (0.01%)
Total		**29,935,623 (100.00%)**

The rates presented for the attack and normal datasets were obtained using the test dataset, which was built using the aforementioned stratification process. The following subsections present and discuss the results obtained using the traditional intrusion detection techniques and the proposed multi-objective feature selection method for building generation capable models.

6.1.1.2 Traditional Model Building Process

The traditional model building process was divided into two groups: in one (*selection*) the traditional feature selection was performed and in the second (*no-selection*) it was not.

The *selection* group relied on the traditional feature selection method, as described in [5, 84, 85]. For this purpose, a wrapper-based GA feature selection method was used, the objective of which was to increase the accuracy in the validation dataset. The GA was used with 100 generations and 100 populations for each generation, a mutation probability of 3.3%, and a 60% crossover probability. The *no-selection* group selected a subset from 50 features of the extracted features during the model building process[5].

To perform the traditional model building process, the approach normally used in studies in the literature [5] was considered. The *no-selection* and *selection* groups were trained, validated, and tested using the known datasets (Figure 7, Known Attacks, Known Services, and

[4] Details regarding the fine-grained intrusion dataset feature extractor can be found in Appendix 1, and in [5]
[5] It is important to note that the feature extraction process for the fine-grained intrusion dataset was not made by *BigFlow*, in contrary, it used the extractor described in [5]

Known Services' Content), whereas the generated models were evaluated using the remaining datasets (Figure 7). The presented rates were obtained from the test dataset, when the dataset was also used for training (Figure 7, Known datasets), and from the entire dataset, when the dataset was used only for the tests (Figure 7, Similar, New, and Publicly Available datasets). The obtained attack detection rates are presented in Table 11.

Table 11. Rates obtained for attack detection scenarios.

Attacks	***Classifier***	***Model Building Method***	**Dataset**								
			$attack^{known}$			$attack^{similar}$			$attack^{new}$		
			Acc. (%)	*FP (%)*	*FN (%)*	*Acc. (%)*	*FP (%)*	*FN (%)*	*Acc. (%)*	*FP (%)*	*FN (%)*
Probing	*Decision Tree*	*no-selection*	***99.99***	*0.02*	*0.00*	***98.62***	*0.04*	*2.72*	***64.66***	*0.09*	*70.59*
		selection	***99.99***	*0.00*	*0.02*	***93.70***	*0.04*	*12.57*	***53.30***	*0.00*	*91.39*
		multi-objective (attack)	***99.82***	*0.23*	*0.12*	***99.41***	*0.11*	*1.07*	***99.76***	*0.09*	*0.38*
		multi-objective (normal)	***99.93***	*0.01*	*0.13*	***99.27***	*0.04*	*1.42*	***74.56***	*0.05*	*50.83*
		multi-objective (generalization)	***99.88***	*0.15*	*0.09*	***99.66***	*0.11*	*0.57*	***96.34***	*0.16*	*7.17*
		multi-objective (all)	***99.87***	*0.13*	*0.13*	***99.29***	*0.04*	*1.38*	***96.12***	*0.02*	*7.73*
	Naïve Bayes	*no-selection*	***99.75***	*0.27*	*0.22*	***99.04***	*0.15*	*1.76*	***57.38***	*0.18*	*85.06*
		selection	***99.99***	*0.01*	*0.00*	***99.37***	*0.00*	*1.26*	***65.72***	*0.02*	*68.54*
		multi-objective (attack)	***99.61***	*0.72*	*0.07*	***99.35***	*0.46*	*0.84*	***98.61***	*0.38*	*2.39*
		multi-objective (normal)	***99.88***	*0.12*	*0.12*	***98.80***	*0.19*	*2.22*	***62.18***	*0.14*	*75.51*
		multi-objective (generalization)	***99.47***	*0.66*	*0.41*	***96.90***	*2.68*	*3.52*	***90.92***	*2.32*	*15.84*
		multi-objective (all)	***99.67***	*0.45*	*0.21*	***97.76***	*1.26*	*3.22*	***92.03***	*1.13*	*14.80*
DoS	*Decision Tree*	*no-selection*	***99.97***	*0.01*	*0.06*	***99.95***	*0.03*	*0.08*	***75.61***	*0.00*	*48.77*
		selection	***99.99***	*0.00*	*0.03*	***99.96***	*0.03*	*0.06*	***51.11***	*0.00*	*97.78*
		multi-objective (attack)	***99.99***	*0.01*	*0.01*	***99.91***	*0.15*	*0.03*	***92.59***	*0.23*	*14.59*
		multi-objective (normal)	***99.98***	*0.01*	*0.02*	***99.90***	*0.14*	*0.05*	***79.35***	*0.00*	*41.31*
		multi-objective (generalization)	***99.98***	*0.01*	*0.03*	***99.94***	*0.03*	*0.09*	***74.27***	*0.00*	*51.46*
		multi-objective (all)	***99.98***	***0.01***	*0.02*	***99.93***	*0.04*	*0.10*	***90.08***	*0.00*	*19.84*
	Naïve Bayes	*no-selection*	***99.90***	*0.09*	*0.11*	***97.76***	*0.79*	*3.69*	***57.29***	*0.00*	*85.41*
		selection	***99.95***	*0.01*	*0.05*	***98.95***	*0.09*	*2.00*	***50.70***	*0.00*	*98.60*
		multi-objective (attack)	***99.92***	*0.00*	*0.15*	***98.10***	*1.96*	*1.83*	***88.68***	*0.70*	*21.94*
		multi-objective (normal)	***99.35***	*1.17*	*0.13*	***98.95***	*0.09*	*2.00*	***51.92***	*0.00*	*96.15*
		multi-objective (generalization)	***98.94***	*0.10*	*2.06*	***98.03***	*1.39*	*2.55*	***54.78***	*0.00*	*90.43*
		multi-objective (all)	***99.93***	*0.00*	*0.13*	***98.65***	*1.73*	*0.97*	***81.33***	*0.35*	*36.99*

The DT and NB classifiers could obtain a reasonable high accuracy rate in both the Probing and the DoS $attack_{known}$ datasets. The average accuracy and FN rates for the known attacks for both Probing and DoS were 99.87 % and 99.93% for the *no-selection* group and 99.99% and 99.97% for the *selection* group, respectively. The traditional feature selection process (*selection*) improved the classification accuracy for known attacks by an average of 0.08%.

However, in the similar attack datasets, it was possible to observe an increase in the FN rates. The worst classifier was DT with the *selection* method, which showed a 12.57% FN rate

on the Probing database, whereas NB with the *no-selection* method showed a 3.69% FN rate. On average, the FN rate increased by 3.97% and 2.06% with the *selection* and *no-selection* approach, respectively. Thereby, it can be stated that the current approaches in the literature can detect similar attacks with a small increase in the FN rate, 3.01% on average. In most cases, the *selection* approach decreased the FN rate for detecting similar attacks; the only case where the FN rate increased was that of DT on the Probing database, which showed a 9.85% rate. The FP rates remained almost unchanged, with an average increase of 0.09% with both model building methods; thus, it is possible to note that the used services (Table 4) still present the same or similar behavior under different attacks. Finally, for detecting new attacks, the FN rates significantly increased.

The best FN rate was obtained by NB with the *selection* method with a 68.54% FN rate on the Probing dataset, whereas a 48.77% FN rate was obtained by DT with the *no-selection* method on the DoS dataset. The results show the inability of machine learning methods, more specifically batch learning, to detect new attacks in the evaluated scenarios. Neither of the classifiers could maintain the obtained rates during the model testing on known attacks. The ML-based assumption for detecting new attacks was not evidenced during the evaluation tests, where the traditional detection approaches were used: the FN rate as compared to that of the testing phase, was increased by 72.46% and 89.05%, on average, for the *no-selection* and *selection* methods, respectively.

The obtained background detection rates are presented in Table 12. Each batch learner and model building method could detect known services and known services' content, reaching an average FP rate of 0.13% and 0.08%, respectively. The *selection* method improved the FP rate by 0.25% on average for known services' content and by 0.14% for known services.

For detecting new services, the FP rate greatly increased; in general, the *selection* method increased the FP rate significantly: 20.82% on average against 6.03% with the *no-selection* method. However, for detecting similar and new services' content, the *selection* method showed an average FP rate of 0.68% for both similar and new services' content, whereas the *no-selection* method showed an FP rate of 1.50% and 3.06% for similar and new services' content, respectively.

Table 12. Rates obtained for normal detection scenarios.

Atta	***Clas***	***Model Building***	**Dataset**				
			$normal^{known\ content}$	$normal^{similar\ content}$	$normal^{new\ content}$	$normal^{known\ service}$	$normal^{new\ service}$

		Method	Acc. (%)	FP (%)	FN (%)	Acc. (%)	FP (%)	FN (%)	Acc. (%)	FP (%)	FN (%)	Acc. (%)	FP (%)	FN (%)	Acc. (%)	FP (%)	FN (%)
Probing	Decision Tree	no-selection	**99.94**	0.12	0.00	**99.97**	0.05	0.00	**99.98**	0.04	0.00	**99.96**	0.05	0.02	**92.93**	14.12	0.02
		selection	**100.00**	0.00	0.00	**98.89**	2.22	0.00	**98.74**	2.51	0.00	**99.99**	0.01	0.00	**99.34**	1.31	0.00
		multi-objective (attack)	**99.90**	0.20	0.00	**99.81**	0.37	0.00	**99.73**	0.54	0.00	**99.91**	0.14	0.03	**98.50**	2.98	0.03
		multi-objective (normal)	**99.99**	0.01	0.00	**99.99**	0.01	0.00	**99.99**	0.01	0.00	**99.97**	0.01	0.05	**99.92**	0.11	0.05
		multi-objective (general)	**99.92**	0.07	0.09	**98.85**	2.22	0.09	**97.30**	5.31	0.09	**99.86**	0.23	0.05	**86.41**	27.13	0.05
		multi-objective (all)	**99.95**	0.08	0.02	**99.93**	0.11	0.02	**99.96**	0.07	0.02	**99.94**	0.05	0.02	**99.84**	0.27	0.05
	Naive Bayes	no-selection	**99.67**	0.52	0.14	**98.31**	3.23	0.14	**96.90**	6.05	0.14	**99.83**	0.31	0.03	**96.96**	6.05	0.03
		selection	**99.99**	0.02	0.00	**99.81**	0.37	0.00	**96.96**	0.08	0.00	**99.99**	0.01	0.00	**79.43**	41.15	0.00
		multi-objective (attack)	**99.54**	0.93	0.00	**98.08**	3.93	0.00	**96.55**	6.90	0.00	**99.37**	1.27	0.00	**96.06**	7.87	0.00
		multi-objective (normal)	**99.94**	0.04	0.08	**99.92**	0.09	0.08	**99.91**	0.11	0.08	**99.97**	0.02	0.03	**99.46**	1.05	0.03
		multi-objective (general)	**99.56**	0.68	0.22	**97.88**	4.03	0.22	**95.02**	9.74	0.22	**98.73**	2.41	0.13	**98.88**	6.12	0.13
		multi-objective (all)	**99.61**	0.64	0.13	**98.13**	3.60	0.13	**96.55**	6.77	0.13	**98.94**	2.05	0.08	**96.50**	6.92	0.08
DoS	Decision Tree	no-selection	**99.99**	0.03	0.00	**99.98**	0.04	0.00	**99.97**	0.05	0.00	**98.98**	0.02	0.02	**99.78**	0.43	0.02
		selection	**100.00**	0.01	0.00	**99.97**	0.07	0.00	**99.98**	0.04	0.00	**100.00**	0.01	0.00	**98.27**	3.46	0.00
		multi-objective (attack)	**99.96**	0.03	0.04	**99.94**	0.08	0.04	**99.90**	0.15	0.04	**99.93**	0.09	0.04	**93.33**	13.29	0.04
		multi-objective (normal)	**99.98**	0.03	0.02	**99.99**	0.01	0.02	**99.98**	0.02	0.02	**99.99**	0.00	0.02	**99.98**	0.02	0.02
		multi-objective (general)	**99.96**	0.06	0.02	**99.97**	0.05	0.02	**99.98**	0.03	0.02	**99.97**	0.05	0.01	**99.98**	0.04	0.01
		multi-objective (all)	**99.96**	0.03	0.05	**99.92**	0.11	0.05	**99.91**	0.13	0.05	**99.94**	0.07	0.05	**99.92**	0.12	0.05
	Naive Bayes	no-selection	**99.50**	0.36	0.65	**98.33**	2.69	0.65	**96.63**	6.08	0.65	**99.64**	0.20	0.51	**97.99**	3.50	0.51
		selection	**100.00**	0.00	0.00	**99.98**	0.04	0.00	**99.95**	0.09	0.00	**100.00**	0.00	0.00	**81.30**	37.35	0.00
		multi-objective (attack)	**98.91**	0.49	1.69	**98.07**	2.16	1.69	**98.71**	0.89	1.69	**99.35**	0.29	1.02	**92.37**	14.23	1.02
		multi-objective (normal)	**99.99**	0.00	0.03	**99.98**	0.02	0.03	**99.98**	0.02	0.03	**99.99**	0.00	0.03	**99.82**	0.34	0.03
		multi-objective (general)	**98.93**	0.04	2.11	**98.32**	1.26	2.11	**97.20**	3.48	2.11	**98.93**	0.02	2.12	**91.91**	14.06	2.12
		multi-objective (all)	**99.86**	1.20	0.08	**98.98**	1.95	0.08	**98.09**	3.74	0.08	**99.77**	0.13	0.32	**98.73**	1.74	0.32

Finally, Table 13 shows the generalization evaluation performed on DARPA1998. A significant increase in the FP and FN rates can be observed when the model was used in a different scenario. Several observations can be made from Tables 11, 12, and 13 regarding the traditional model building methods:

- Both batch learners, regardless of the model building method used, could detect known events (attacks, services, and services' content). The worst detection rates were 99.75% for known attacks, 99.67% for known services' content, and 99.98% for known services.

Table 13. Rates obtained for generalization evaluation.

Attacks	Classifier	Model Building Method	Accuracy (%)	FP (%)	FN (%)
Probing	Decision Tree	no-selection	**86.97**	15.20	10.85
		selection	**90.38**	8.86	10.36
		multi-objective (attack)	**81.55**	31.80	5.11

		multi-objective (normal)	***86.79***	*15.78*	*10.63*
		multi-objective (general)	***98.42***	*2.89*	*0.27*
		multi-objective (all)	***96.25***	*7.24*	*0.27*
	Naïve Bayes	*no-selection*	***75.93***	*48.12*	*0.00*
		selection	***78.56***	*33.29*	*9.59*
		multi-objective (attack)	***83.77***	*32.43*	*0.00*
		multi-objective (normal)	***68.75***	*7.91*	*54.59*
		multi-objective (general)	***97.29***	*5.43*	*0.00*
		multi-objective (all)	***96.43***	*7.15*	*0.00*
DoS	*Decision Tree*	*no-selection*	***38.98***	*30.47*	*91.57*
		selection	***99.41***	*0.00*	*1.18*
		multi-objective (attack)	***94.56***	*10.89*	*0.00*
		multi-objective (normal)	***85.23***	*29.54*	*0.00*
		multi-objective (general)	***99.90***	*0.20*	*0.00*
		multi-objective (all)	***99.36***	*1.29*	*0.00*
	Naïve Bayes	*no-selection*	***82.71***	*34.56*	*0.01*
		selection	***84.26***	*31.19*	*0.29*
		multi-objective (attack)	***81.82***	*29.28*	*7.09*
		multi-objective (normal)	***93.53***	*12.77*	*0.05*
		multi-objective (general)	***99.66***	*0.52*	*0.16*
		multi-objective (all)	***95.23***	*8.57*	*0.97*

- None of the batch learners could maintain its obtained accuracy on the model building dataset for detecting new attacks. The ML-based assumption for detecting new attacks was not evidenced when the machine learning technique was used.
- The traditional machine learning (through batch learning) technique was able to detect similar attacks (Table 11; 1.97% and 3.95% FP rate increase for the *no-selection* and *selection* methods, respectively), making it a viable approach for detecting possible intrusion attempts, provided that the attacks present a similar behavior.
- In general, there was an FP increase for detecting new services (Table 12; 13.42% average FP rate); however, the accuracy loss was less than that observed for detecting new attacks (Table 11; 80.77% average FN rate).
- A small increase in the FP rate was evidenced for detecting new services' content (Table 12; 2.80% and 0.67% FP rate increase for the *no-selection* and *selection* methods, respectively); however, in most cases, the classifiers were able to correctly distinguish the classes.
- When using the obtained batch learners in a different environment (Table 13), the detection accuracy significantly decreased, even for detecting known attacks; in most cases, the model became scenario-dependent.

The proposed evaluation method (see Section 4.3) allowed the common assumptions presented in the literature to be verified. Moreover, it allowed rich intrusion detection properties to be obtained from the intrusion detection scheme, which helps experts determine whether their systems are reliable for open-world usage or not.

During the evaluation tests using the traditional machine learning (batch) model building techniques it was observed that, when a classifier faced known events, it presented a reasonable accuracy rate. The results showed a decrease in accuracy when a classifier faced similar events; the effect on accuracy further increased when a classifier faced new attacker and client behaviors. The next subsection evaluates the proposed multi-objective feature selection method for building generalization capable models (Section 4.3.5)

6.1.1.3 Multi-objective Feature Selection

The well-known NSGA-II [86] algorithm was used for the multi-objective feature selection for building generalization capable models (Section 4.3.5). As previously described, three objectives must be considered during the model building process: the $attack_{rate}$ (Equation 1), $normal_{rate}$ (Equation 4), and $generalization_{rate}$ (Equation 5). NSGA-II operates by minimizing the objectives; thus, for the tests purposes, the obtained error rate in the evaluation tests was considered. The same set of parameters used by the traditional feature selection process was used (Section 6.1.2): 100 generations and 100 populations for each generation, a mutation probability of 3.3%, and a 60% crossover probability.

As stated in Section 4.3.5, the desired objective must be defined according to the administrator's needs. Thus, for test purposes, four operation points were chosen: attack, normal, generalization, and all. Each chosen operating point presented the lowest error rate related to its objective: attack presented the lowest $attack_{rate}$ (Eq. 1) error rate, normal presented the lowest $normal_{rate}$ (Eq. 4) error rate, generalization presented the $generalization_{rate}$ (Eq. 5) error rate, and finally, all presented the lowest error rate considering all objectives. The obtained objective rates are presented in Table 14. The proposed multi-objective feature selection achieved the best results in all cases for detecting its considered objective.

Table 14. Rates obtained for each considered objective.

Attacks	***Classifier***	***Model Building Method***	$attack_{rate}$	$normal_{rate}$	$generalization_{rate}$
Probing	*Decision Tree*	*no-selection*	*87.76*	*98.56*	*86.97*
		selection	*82.33*	*99.39*	*90.38*
		multi-objective (attack)	***99.66***	*99.57*	*81.55*

		multi-objective (normal)	*91.25*	***99.97***	*86.79*
		multi-objective (general)	*98.63*	*96.47*	***98.42***
		multi-objective (all)	*98.43*	*99.92*	*96.25*
	Naïve Bayes	*no-selection*	*85.39*	*98.33*	*75.93*
		Selection	*88.36*	*95.24*	*78.56*
		multi-objective (attack)	***99.19***	*97.92*	*83.77*
		multi-objective (normal)	*86.95*	***99.84***	*68.75*
		multi-objective (general)	*95.76*	*98.01*	***97.29***
		multi-objective (all)	*96.49*	*97.95*	*96.43*
DoS	*Decision Tree*	*no-selection*	*91.84*	*99.74*	*38.98*
		Selection	*83.69*	*99.64*	*99.41*
		multi-objective (attack)	***97.50***	*98.61*	*94.56*
		multi-objective (normal)	*93.08*	***99.98***	*85.23*
		multi-objective (general)	*91.40*	*99.97*	***99.90***
		multi-objective (all)	*96.66*	*99.93*	*99.36*
	Naïve Bayes	*no-selection*	*84.98*	*98.42*	*82.71*
		Selection	*83.20*	*96.25*	*84.26*
		multi-objective (attack)	***95.57***	*97.48*	*81.82*
		multi-objective (normal)	*83.41*	***99.95***	*93.53*
		multi-objective (general)	*83.92*	*97.06*	***99.66***
		multi-objective (all)	*93.30*	*99.09*	*95.23*

The *multi-objective (attack)* operation point improved the $attack_{rate}$ in all cases. As compared to the traditional *selection* method, it improved the $attack_{rate}$ by 17.33% and 10.83% for the DT and NB classifiers on the Probing dataset, respectively, while improving it by 13.81% and 12.37% for the DT and NB classifiers on the DoS dataset, respectively. On average, the *multi-objective (attack)* improved the $attack_{rate}$ accuracy by 10.49% and 13.59% for the *no-selection* and *selection* methods, respectively. As compared to the other operation points, *multi-objective (attack)* improved the $attack_{rate}$ accuracy by 5.54% on average.

In the individual attack detection accuracy (Table 11), a significant improvement can be observed. The *multi-objective (attack)* operating point significantly improved the detection of similar and new attacks. On average, it improved by 0.77% and 0.27% for similar attacks as compared to the traditional model building methods and the other operating points, respectively. For the detection of new attacks, the *multi-objective (attack)* operating point enabled the detection of new attacks in most cases, improving the detection rate of new attacks by 35.44% as compared to the traditional model building methods and by 16.25%, on average, as compared to the other operating points.

Finally, it is possible to note a tradeoff between the $attack_{rate}$ objective and the other objectives. In most cases, the *multi-objective (attack)* operation point reduced the accuracy of the other objectives; as compared to the other operation points, the only case where the objective

was improved was for the DT classifier on the Probing dataset, showing an improvement of 0.78% on average for the normal objective. On average, the tradeoff between objectives when the attack objective was considered was -0.62% for normal detection and -7.65% for the generalization objective.

The *multi-objective (normal)* operation point slightly improved the $normal_{rate}$ objective as compared to the other techniques in all cases. The *multi-objective (normal)* operation point improved the detection of normal events on average by 1.74% and 1.44% as compared to the traditional model building methods and the other operating points, respectively. These small accuracy improvements occurred as a result of the models' capacity to detect normal events with little or no effect on accuracy (Section 6.1.1.2). It is possible to note a significant tradeoff between the detection of normal events and attack events, and generalization. On average, as compared to the other operating points, when the *multi-objective (normal)* operation point was considered the detection of attacks was reduced by 6.87%, whereas the model generalization capacity decreased by 10.11%.

The *multi-objective (generalization)* operation point significantly increased the $generalization_{rate}$ in all scenarios. As compared to the traditional model building techniques, the *multi-objective (generalization)* operation point presented, on average, a 19.17% higher accuracy rate, while it was increased, on average, by 10.21% as compared to the other operation points. Regarding the *multi-objective (generalization)* operation point tradeoff, on average, it was evidenced that there was a decrease in the attack detection rate of 6.48% and 8.35%, whereas for the normal detection, there were an increase of 0.32% and a decrease of 0.99% for the traditional model building methods and the other operating points, respectively. Thus, to provide generalization, a significant tradeoff between attack and normal detection rates is required. However, the most important point pertaining to generalization was that an old benchmark database was used, because if the generalization rate results are good, it means that the proposed database is equivalent to the old one. Therefore, it is possible to use an updated database to test recent attacks.

Finally, the *multi-objective (all)* operation point was aimed to improve all the considered objectives. When considering all the objectives, an improvement of 9.49% was shown: the average detection rate was 97.42%, whereas the average detection rate for all the objectives using the traditional model building methods was 87.93%; when the other operating points were considered, the average detection rate was 93.68%.

Figure 22 shows the operation points through our method for the detection of probing attacks. It can be noticed that the generalization capacity increases while the attack detection rate decreases. However, the detection of normal events does not significantly decrease the system generalization capacity. Similar behavior was observed for the detection of DoS attacks.

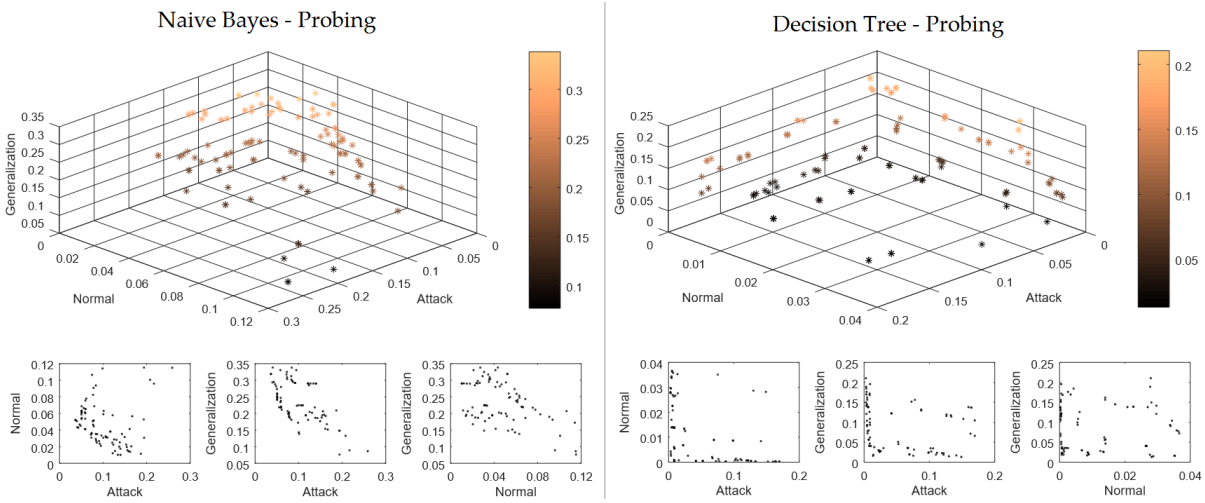


Figure 22. Multi-objective operation points for probing attacks. The operation points are shown in terms of objective error rate; the operation points are chosen according to their lowest error rate.

6.1.2. Reliability in Classifications

Two distinct detection approaches were evaluated for the proposed method to ensuring reliability in classifications. First, a single classifier using the decision tree (DT), naïve bayes (NB) or linear discriminant analysis (LDA) algorithm; second, a combination of the three classifiers using majority voting, as explained in Section 4.4.3 (Figure 14). The C4.5 decision tree algorithm was used with a confidence factor of 0.25. The Fisher's method [87] was used for the LDA classifier. The same stratification procedure adopted for the building of generalization capable models were employed (see Section 6.1.1). The classifiers were evaluated by the means of the Probing scenarios datasets, in the fine-grained intrusion datasets.

The resulting average accuracy in each scenario is shown in Table 15, as measured by the average of both TP and TN. All classifiers presented a reasonably good performance when used in the known scenario testing dataset. The best accuracy rate, 99.97% was obtained with the DT classifier. The worst classifier accuracy rate, 99.44%, was achieved by LDA, although it was only 0.53% lower than DT. When evaluated with the similar scenario, the classifiers were able to detect events with an average accuracy drop of 0.88% compared to the baseline scenario. For new attacks, however, the accuracy decreased on average by 37.64%. The best accuracy rate when detecting new attacks was obtained with the combination/voting classifier, with an

accuracy of 70.29% (Table 15). None of the used classifiers or methods were able to maintain their accuracy obtained in the known scenario when detecting similar or new attacks. If such classifiers were used in a real-world environment, their accuracy would likely drop over time due to changes in the traffic or attack profiles.

Table 15. Accuracy for each Probing scenarios using the obtained classifiers.

Classifier / **Accuracy**	**Probing Scenario**		
	$attack^{known}$	$attack^{similar}$	$attack^{knew}$
DT	99.97 %	98.62 %	64.66 %
NB	99.75 %	99.23 %	57.38 %
LDA	99.44 %	98.39 %	56.00 %
Combination	99.75 %	99.14 %	70.29 %

6.1.2.1 Evaluation of the proposed rejection method

The proposed rejection method aims at maintaining the classifier reliability over time even in the absence of model updates. To achieve this goal, the class assignment (Figure 14) must reject potentially wrong classifications. Therefore, the evaluation tests aim at checking the detection accuracy, while still rejecting as few events as possible.

Due to the number of used features (see Appendix 1), evaluating all possible values of $\propto$ for each feature and profile similarity (Figure 12) is unfeasible. Therefore, two tests for each final class assignment was performed (Figure 14). The first test, named Different Alpha, used different $\propto$ values (Figure 12) for each feature group (*header-based*, *service-based*, and *host-based*, described in Appendix 1), whereas the second test (named Same Alpha) used the same $\propto$ for all features. The profile similarity varied from 0% to 100% in 1% increments. The rejection rate is measured as the ratio between the number of rejected instances and the total number of instances in the test set. The accuracy rate vs rejection rate tradeoff using the combined classifiers for the detection of new attacks is shown in Figure 23.

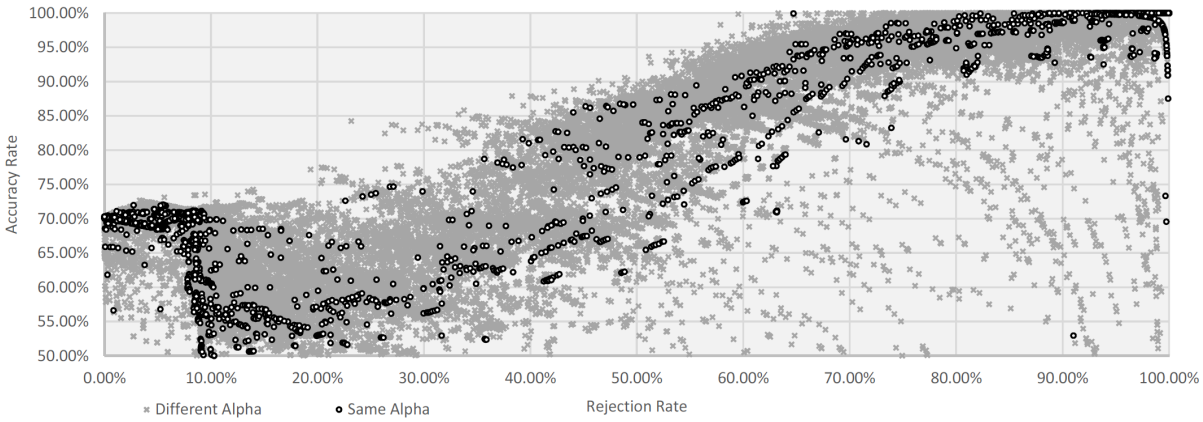


Figure 23. Accuracy-rejection tradeoff for the combination technique while detecting new attacks.

It is possible to note that in most cases there is a direct relation between accuracy and rejection, regardless of the ∝ technique (Figure 23, Different Alpha and Same Alpha). One may notice that different distributions of feature values allowed to improve the accuracy rate while rejecting fewer instances. The classifier combination scheme was able to reach an accuracy rate close to 100% while rejecting 59.52% of instances in the new attack dataset. All evaluated classifiers were able to reach an $attack^{new}$ accuracy of 100% at the cost of a 60% rejection rate. The combination classifier provided the best accuracy with a minimum rejection rate considering all scenarios, outperforming the best single classifier (DT) by about 5% in the low rejection rate setting, and by about 10% in the average rejection setting (Table 16).

In the real world, it is not possible to choose a different set of thresholds for each event, because the classifier is unable to determine whether an event is a known attack, a similar attack, or a new one. Therefore, the choice of a set of thresholds must be made taking into account the tradeoff between accuracy and rejection rate. Figure 24 shows the accuracy-reject tradeoff between the accuracy in detecting new attacks and the rejection rate for known and similar attacks, using the same set of thresholds during the detection. The graph shows that it is possible to maintain the accuracy for the detection of new attacks, but at the cost of an increased rejection rate for known and similar attacks. For instance, it is possible to maintain the accuracy rate at 95% in a scenario with new attacks, at the cost of rejecting 31% in average of the events in the other two scenarios (known and similar attacks).

The set of thresholds should be established according to the user goals. If certain lenience for accuracy is acceptable, fewer events will be rejected, but the class assignment will be more susceptible to errors. Table 16 shows the accuracy-reject relationship for each dataset, for the rejection settings highlighted in Figure 25, using the same set of thresholds. Four rejection rate settings (*no rejection*, *low rejection*, *average rejection*, and *high rejection*) were selected for the new attacks dataset. The same set of thresholds were used in the other scenarios to investigate the rejection rate impact for the known and similar attack datasets. The obtained results are shown in Table 16.

Table 16. Accuracy-rejection tradeoff for each dataset using the points marked in Figure 25.

Rejection Rate	Classifier (Normal Alpha, Attack Alpha)	Probing Dataset					
		$attack^{known}$		$attack^{similar}$		$attack^{new}$	
		Acc. (%)	Rej. (%)	Acc. (%)	Rej. (%)	Acc. (%)	Rej. (%)
No Rejection	DT (n.a., n.a.)	99.97	-	98.62	-	64.66	-
	NB (n.a., n.a.)	99.75	-	99.23	-	57.38	-
	LDA (n.a., n.a.)	99.44	-	98.39	-	56.00	-

Rejection Rate	Classifier (Normal Alpha, Attack Alpha)	Probing Dataset					
		$attack^{known}$		$attack^{similar}$		$attack^{new}$	
		Acc. (%)	Rej. (%)	Acc. (%)	Rej. (%)	Acc. (%)	Rej. (%)
	Combination (n.a., n.a.)	99.75	-	99.14	-	70.29	-
Low Rejection	DT (0.55, 0.18)	99.97	0.21	98.70	1.05	**67.37**	4.82
	NB (0.62, 0.27)	99.75	0.14	99.23	0.25	59.66	4.07
	LDA (0.64, 0.29)	99.44	0.14	98.48	0.31	58.15	3.98
	Combination (0.53, 0.16)	99.75	0.01	99.14	0.00	**72.66**	3.31
Average Rejection	DT (0.83, 0.37)	99.98	5.40	98.80	5.77	**74.90**	24.14
	NB (0.88, 0.51)	99.75	5.41	99.21	5.88	67.94	23.50
	LDA (0.88, 0.51)	99.65	5.71	99.65	6.00	66.37	24.99
	Combination (0.81, 0.25)	99.76	0.25	99.14	0.15	**84.27**	23.16
High Rejection	DT (0.90, 0.51)	99.99	37.27	99.87	25.79	**99.92**	59.57
	NB (0.90, 0.51)	99.96	37.34	99.92	25.42	99.92	59.61
	LDA (0.90, 0.51)	100.00	37.38	99.87	25.57	99.86	59.62
	Combination (0.90, 0.51)	99.95	37.27	99.92	25.38	**99.86**	59.52

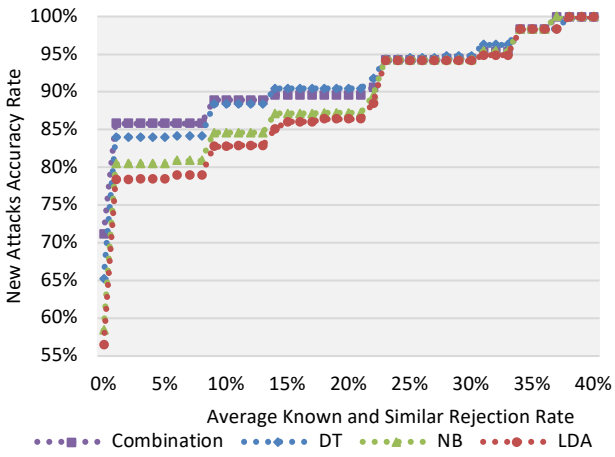


Figure 24. Tradeoff between the accuracy improvement for new attacks and the rejection of known and similar attacks.

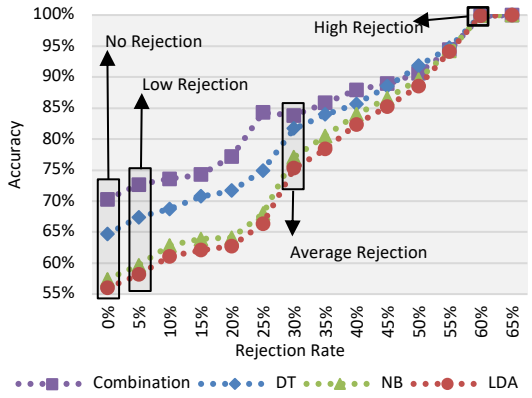


Figure 25. Tradeoff between accuracy and rejection rate, for each classifier in new attacks dataset.

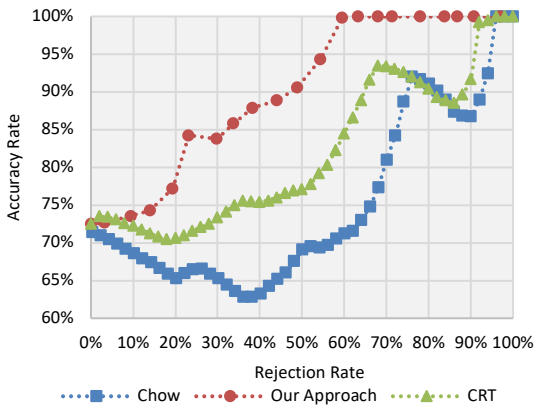


Figure 26. Accuracy-rejection tradeoff, for the combination classifier in the new attacks dataset, using the evaluated rejection techniques.

The *Average Rejection* setting presented the best accuracy-reject tradeoff. The rejection method was able to improve the classification accuracy by 13.98% for new attacks while rejecting only 0.25% known attacks and 0.15% similar attacks, using the combination classifier (*Average Rejection*, Table 16). The combination classifier produced, in average, the best results when compared to the single classifiers at the same rejection rate interval. In summary, the proposed rejection method allowed the detection of new attacks while maintaining the classifier's overall reliability.

6.1.2.2 Comparison with other rejection approaches

Finally, two commonly used rejection approaches that rely on class probabilities, the Chow's rule [37] and the Class-related Reject Threshold (CRT) [38], were compared to the proposed method. Chow's rule defines a single rejection threshold for all classes, whereas CRT uses a different threshold for each class. For evaluation purposes, the combination classifier was used because it presented the best results (Table 16). The three approaches – CRT, Chow and the proposed approach – were evaluated using the Probing New Attacks dataset. The rejection rates from 0% to 100% were evaluated. Figure 26 shows the accuracy-reject tradeoff comparison for the evaluated approaches.

The proposed approach outperformed both existing techniques, CRT and Chow's rule. The traditional rejection approaches were not able to identify behavior changes and increased the classification confusion; the assigned class probabilities were high even for misclassified instances. In contrast, the proposed approach was able to operate with fewer misclassifications in the presence of traffic behavior changes, reaching 100% accuracy while rejecting 60% of the events.

6.1.3. Discussion

A machine learning classifier works by identifying similar behaviors and must have representative instances from all of the considered classes [6]. This section made experiments in order to evaluate the common assumptions in the literature regarding network-based intrusion detection. As the evaluation tests have shown, the assumption that a classifier will be able to detect new attacks only holds when their behavior is similar to the one used during the classifier training. Thereby, a machine learning (batch) detection system becomes unreliable when the traffic behavior changes.

In the light of this, the first evaluation aimed at providing generalization capable batch learning models. The goal was to build a batch learner model that is able to generalize the behavior from the training dataset (limited set) to a wider one, represented as the similar and

new scenarios (i.e. production environment). The proposed approach, by the means of a multi-objective feature selection technique, have enabled the building of generalization capable models, when compared to their *no-selection* or *selection* counterpart. However, although the model is able to generalize the behavior from the training dataset, the detection of new behaviors (e.g. $attack^{new}$) was still a problem to be addressed. This because the accuracy rate of new behaviors did not meet the accuracy rate from the known or similar ones.

Thereby, the proposal has dealt with such issue by the means of rejecting potentially non-reliable classifier decisions. Although, in the literature, such effect is often ignored [6, 7]. In which, in general, the network traffic is considered static, and the common machine learning evaluation schemes are adopted without taking into account its dynamic characteristics, which demands constant classifier updates over time.

To ensure the classification reliability, the use of simple lower and upper thresholds was proposed, obtained from the feature distributions observed during the model training phase. Due to the large number of used features, the contribution of distinct features according to their feature group was analyzed. The proposed rejection method was able to guarantee the system reliability with a low accuracy-reject tradeoff, improving the accuracy in 13.98% for new attacks while rejecting only 0.25% of known behaviors using a classifier combination scheme.

6.2 *Stream Learning*

In order to deal with the constant network behavior changes over time, one must update its detection scheme periodically. However, the model update is not easily feasible in high-speed networks. In the light of this, to provide ongoing updated classification models, the reliable intrusion detection model relies in the usage of stream learning detection schemes. The proposed approach considers the usage of an unsupervised stream learning approach, as it enables to provide ongoing updated classification models, but, on the other side, it is prone to adversarial attacks.

6.2.1. Stream Learning – Resilience to Adversarial Attacks

This section evaluates the proposed approach for providing unsupervised stream learning schemes resilient to adversarial attacks (Section 4.5). For the tests purposes the well-known Micro-cluster-based Continuous Outlier Detection (MCOD) [87] algorithm has been considered in the proposed method (Section 4.5). For comparison purposes two other approaches were considered: Traditional Batch Learning (TBL) and Traditional Stream Learning (TSL). For evaluation purposes the fine-grained intrusion dataset DoS known scenario was considered (see Section 5.1).

6.2.1.1 Model Obtainment Process

For the proposed method (Section 4.5), two classes were considered: normal and attack. Thereby, for each test, two outlier detectors were used, one for normal and one for attack. A sliding window of 10,000 events was considered. A total of 50 events was established as neighbor (k) parameter. Each class outlier detection has its own radius parameter. A series of tests were conducted to establish the radius parameters; the choosing criteria was to minimize the fitness value in Equation 10.

$$fitness = error_{rate} + rejection_{rate} \quad \text{Equation (10)}$$

The $error_{rate}$ and $rejection_{rate}$ were defined through the detection of the initial 10,000 normal events followed by the detection of the 10,000 further attack events from the training dataset (Table 6, *DoS (Known)* scenario). The radius values for each class outlier detector, normal and attack, was varied in a 0.01 interval from zero to 2.00.

The k-Nearest Neighbor (kNN) classifier was used for the TBL. To allow comparison, a total of 5,000 events for each class, normal and attack, are used for the classifier neighbor computation. The 5,000 events of each class are defined by the k-means clustering algorithm [89], using the training dataset (25% of randomly chosen events from Table 6). The kNN neighbors set are not updated during the classification process. Finally, for the TSL, the MCOD is used. However only the normal class is considered, as commonly performed in related works [88], whilst the radius obtainment process was established only by the $error_{rate}$ minimization.

6.2.1.2 Traditional Evaluation

Initially, the traditional evaluation process was considered for the evaluated approaches. In the traditional evaluation, the adversarial settings (Section 2.2) are not considered.

For the kNN classifier, the dataset was divided into: training, validation and testing, containing, 25%, 25% and 50%, respectively of the whole dataset (Table 6, *DoS (Known)* scenario). Due to the adaptive nature of the considered stream learning algorithm, the whole dataset is used for the traditional evaluation. The events are replayed in the exact same order as they appear in the original dataset. Table 17 shows the accuracy rates regarding each of the evaluated approaches, where the method column refers to the used approach during the detection stage. Each approach is tested with a different set of attacks used during the training stage, shown in brackets in the method column.

Table 17. Proposed stream learning resilient to adversarial attacks and traditional evasion evaluation.

Method	*Accuracy (Reject)*

(Attack used for training)	*Normal* *Acc. (Rej.)*	*SYNFlood* *Acc. (Rej.)*	*UDPFlood* *Acc. (Rej.)*	*ICMPFlood* *Acc. (Rej.)*
Proposed Approach MCOD (SYNFlood)	100.00% (0.04)	100.00 (0.00)	- (100.00)	- (100.00)
Traditional kNN (SYNFlood)	99.83 (N.A.)	100.00 (N.A.)	100.00 (N.A.)	0.01 (N.A.)
Proposed Approach MCOD (UDPFlood)	100.00 (0.98)	- (100.00)	100.00 (0.10)	- (100.00)
Traditional kNN (UDPFlood)	99.93 (N.A.)	49.97 (N.A.)	100.00 (N.A.)	0.01 (N.A.)
Proposed Approach MCOD (ICMPFlood)	100.00 (0.97)	- (100.00)	- (100.00)	100.00 (0.12)
Traditional kNN (ICMPFlood)	100.00 (N.A.)	3.23 (N.A.)	100.00 (N.A.)	100.00 (N.A.)
Traditional Stream Learning MCOD	99.19 (N.A.)	0.81 (N.A.)	0.69 (N.A.)	0.22 (N.A.)

One can notice that both the proposed approach and the traditional batch learning (kNN) can detect the same set of attacks, with a significantly high accuracy rate. Regarding the detection of attacks, both the proposed approach and the kNN presented a FN rate of zero percent, when detecting the same set of attacks the system has been trained with. Considering the FP rate, the kNN classifier achieved 0.17, 0.07 and zero percent when trained with SYNFlood, UDPFlood and ICMPFlood attacks, respectively. The proposed approach achieved a FP rate of 0.00 percent in all tested cases. However, the proposed approach rejected potentially wrong classifications. In such a case, 0.04, 0.98 and 0.97 percent of normal events were rejected for SYNFlood, UDPFlood and ICMPFlood attacks, respectively. It can be observed that the proposed approach presents a similar detection accuracy when compared to the traditional batch learning approach. However, the proposed approach rejects potentially wrong classifications, which can be observed by comparing the kNN FP rate and the proposed approach rejection rate.

Considering the traditional stream learning approach, it was possible to notice that when events are replayed in the exact same order as they appear in the original dataset (Table 6, *DoS (Known)* scenario), the method can detect only the initial attacks – when the sliding window is almost fully populated with normal events. However, as the attacks occurrence increases, the further attack events are classified as inlier (normal). Such a property occurs due to the adaptive nature of stream learning algorithms, which allows that an event, initially classified as outlier (attack), to be added in the sliding window, hence, allowing that an attack to become an inlier over time, perverting the outlier detector behavior. In this manner, the traditional stream learning algorithms, must consider such property – in the intrusion detection field, which is dealt in this work by considering the *immutable behavior* (Section 4.5).

6.2.1.3 Adversarial Settings – Exploratory Attacks

Two types of attacks were evaluated in this experiment: the *traditional evasion* and the *window interval exploit*.

a. Traditional Evasion

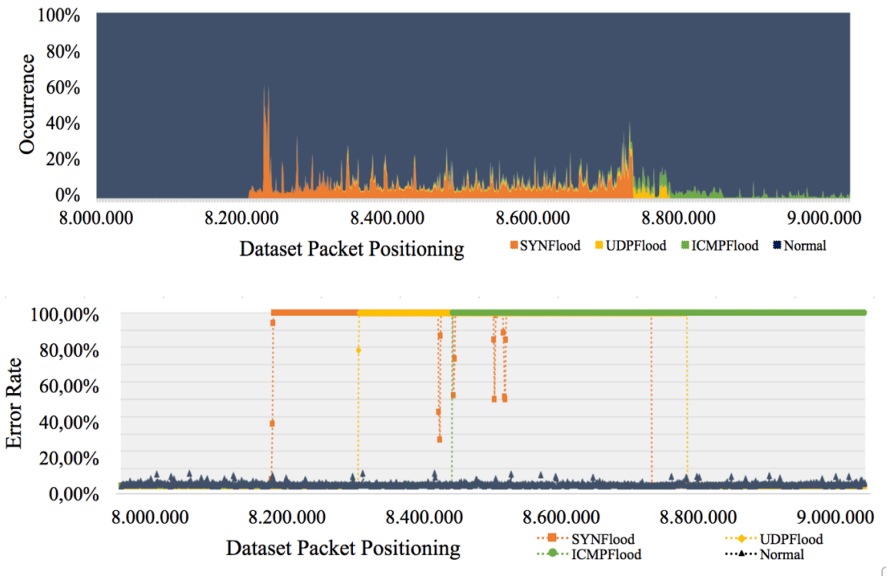
The *traditional evasion* refers to the detection of attacks with a different kind of behavior to the attack that the system was trained with, however with the same or similar outcome. For example, attacks aiming at generating a significant amount of network traffic at the targeted victim, regardless of the considered protocol, e.g. UDP, TCP or ICMP floods. In this way, each of the considered approaches were also tested with a different flood attack than the system was trained with. The obtained accuracy is shown in Table 17.

Regarding the traditional batch learning (kNN), the attacker could evade the system, while generating an attack that produces the same or similar outcome. When the kNN was used, the evasion possibility was evidenced for all evaluated attacks: SYNFlood, UDPFlood and ICMPFlood. For instance, when the system was trained with SYNFlood attacks, the attacker is still able to evade the detection system by generating ICMPFlood attacks. Moreover, the tested approach accepted only the classifications outputs regarding the attacks the system was trained with. Such a high rejection rate, and in this case reliability increase, due to the possible increase in the error rate, occurred due to the lack of decision unanimity between the outlier detectors, and thereby rejecting the assigned class.

b. Window Interval Exploit

The second evaluated exploratory attack is called as *sliding window exploit*. The *sliding window exploit* attack aims at evaluating the traditional stream learning accuracy according to the attack occurrence in a sliding window. The increase in the attack frequency in the sliding window renders the stream learning algorithm unreliable over time, this because an attacker is able to render an outlier as an inlier, as he increases the attack occurrence over time. Figure 27 (bottom chart) shows the error rate regarding each of the evaluated attacks, during the 8 to 9 million packets in the created dataset. The error rate is evaluated in a 1,000 packets interval.

Figure 27. Traditional stream learning approach behavior under network traffic intensive attacks, upper chart shows the network packet classes occurrence while bottom chart shows the related error rate. Attack detection error rate increases according to the occurrence of attacks in the *sliding window*.



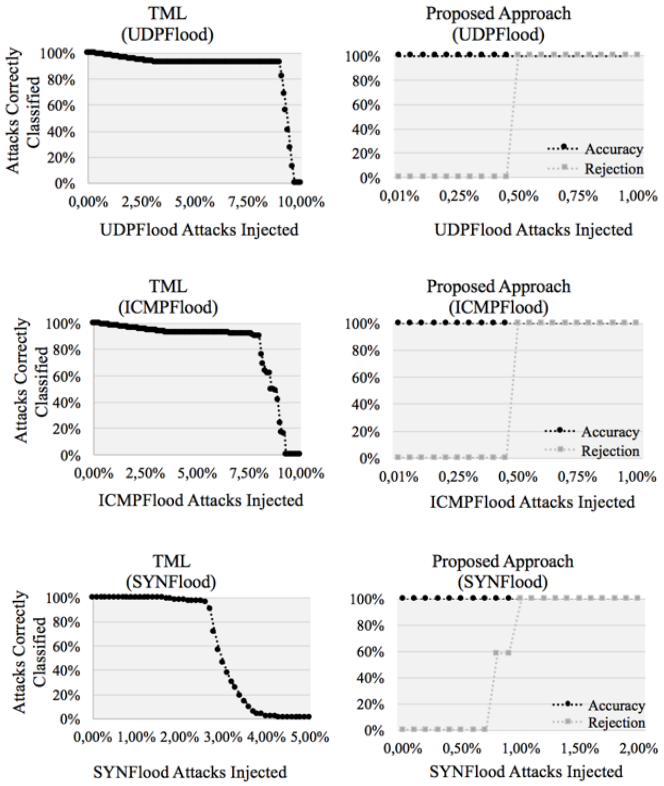
It is possible to observe that during the normal events detection, the traditional stream learning algorithm error rate remains similar to the rate obtained during the traditional evaluation (Section 6.3.2, 0.81 percent). However, as the attacks begins to occur (around the 8.2 millionth packet), the attack detection error rate increases, due to the increasing in the attack occurrence. In this manner, the attacker can exploit the traditional stream learning algorithm sliding window, by increasing the attack occurrence (Figure 27, upper chart), causing the attacks to be classified as inlier (normal) due to their frequency increase in the sliding window.

The *sliding window exploit* does not occur in the proposed approach due to the *immutable behavior* (Section 4.5.1) and the class-specific single class detection mechanism (Section 4.5). The results are shown in Table 17. The attacker is not able to add attacks in the normal outlier detector sliding window due to the *immutable behavior*. Whilst, if the detection mechanism wrongly classifies an event, and thereby add it in its sliding window, the event will be rejected, because it will not be possible to establish an unanimity between the others outliers' detectors, given the outliers decision have a non-null intersection.

6.2.1.4 Adversarial Settings – Causative Attacks

Finally, to evaluate the causative attacks resilience, a training dataset poisoning approach was adopted. The traditional batch learning (TBL) and the proposed approach were evaluated regarding the influence that attacks, initially injected into the training dataset as normal events, have in the resulting accuracy. Thereby, the goal was to evaluate each of the considered methods, regarding their resilience to dataset poisoning attacks. Figure 28 shows the relation between the attack detection rate and the attacker control percentage over the training dataset, while successfully injecting attacks labeled as normal activity.

Figure 28. Traditional Batch Learning (TML, Traditional Machine Learning) and Proposed Approach resilience to *causative attacks* (training dataset poisoning attacks). The horizontal axis shows the rate of attacks injected into the training dataset labeled as normal activities. The vertical axis shows the accuracy and rejection rate impact while detecting such attacks, having the infected training dataset.



Regarding the TBL, it is possible to note that the three evaluated attacks can evade the detection mechanism when injected in the training dataset as normal events. The accuracy rate for SYNFlood attacks dropped for 50% when only 3% of normal events were SYNFlood injected attacks. Whilst, for ICMPFlood and UDPFlood the attacker could evade the detection system when 9% of attacks were injected. On the other side, the proposed approach could detect when attacks were injected into the training dataset and reject further classifications. Such a characteristic occurred due to class specific outlier detector, the attacks injected into the training dataset as normal events incurred in a lack of outliers unanimity in the classification decision process, thereby, the events were rejected.

6.2.2. Discussion

In order to enable production usage, intrusion detection schemes must be easy to update (see Section 6.1). One approach that stand out amongst others to this end is unsupervised stream learning techniques. By the means of a sliding window, unsupervised stream learning techniques enables the ongoing identification of network anomalies without the assistance of

an expert. The premise is that an outlier (attack) presents a behavior significantly different from the other events in the sliding window.

However, although being easy to update, such techniques are prone to adversarial attacks. To this end, this section has evaluated the proposed approach to provide resiliency to adversarial attacks at both training and testing time. The proposed approach outcomes both traditional batch and stream learning techniques to provide resiliency to adversarial attacks. However, as a tradeoff to provide resiliency, the proposed technique further increases the system rejection rate.

6.3 *Reliable Learning*

This section presents the evaluation regarding the building of the reliable intrusion detection model. First, batch and stream learning techniques are evaluated regarding their accuracy and reliability using the *MAWIFlow* dataset (Section 5.2). Second, the proposed conformal evaluator is evaluated in order to provide reliability even in the absence of model updates. Finally, the proposed reliable intrusion detection model is evaluated regarding its accuracy and reliability over time.

6.3.1. Accuracy Degradation of Machine Learning Classifiers

This section evaluates if ML-based approaches can maintain their reliability over time while processing network traffic from real networks. For evaluation purposes both batch and stream learning classifiers that are frequently employed in intrusion detection were considered. For the Batch Learning the Random Forest (RF) [70] was considered, on the other side, the OzaBoosting (OZA) [90] was used as Stream Learning classifier.

For each evaluated classifier, five different views (feature sets) were tested: Viegas [5], Nigel [79], Orunada [78], Moore [80], and All, which comprises all prior feature sets. The evaluation employs a single training step using the data of *MAWIFlow* from the first two months of 2007, and then employs the built model for the remainder of the time (March 2007 to December 2016), without updates, as often made in the literature.

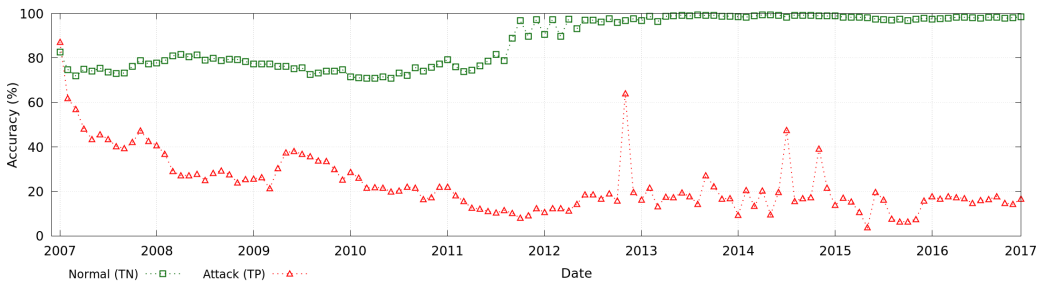
The Weka API [82] version 3.8.0 was used for the implementation and evaluation of the Batch Learning classifier, while the MOA API [91] version 2017.06 was used for the Stream Learning classifier. The RF is composed of 50 decision trees, while the OZA is composed of 50 Hoeffding Trees [44] as their base-learner classifiers. The accuracy rates regarding true

negative (TN, normal events correctly classified) and true positive (TP, attack events correctly classified) during the 10 years of *MAWIFlow* are shown in Figure 29 and 30[6].

Both considered ML approaches, Batch and Stream Learning, have shown significant accuracy impact over time. Several observations can be made from Figures 29 and 30, regarding traditional model building process:

- *Accuracy*: regardless of the considered classification approach, either Batch or Stream Learning, an increase in the error rates can be seen only months after training. The increase in the error rates can be evidenced up to four years after that of training time;
- *Feature Sets*: despite presenting different outcomes, models become unreliable over time regardless of their used feature sets. However, there is a difference on how each model performs, according to their view. For instance, models with both Viegas and Moore views significantly increases their TP rates after 2012 (despite not reaching their accuracy obtained at training time in 2007), while models with Nigel and Orunada views does not significantly changes after such time;
- *Attack Events Detection*: the detection of attacks is more challenging than the detection of normal events over time, similar findings to that obtained in the fine-grained dataset evaluation (see Section 6.1). Such property can be noted due to the difference of TN and TP rates during the 10 years of *MAWIFlow*;

(a) RF (*Viegas View*)

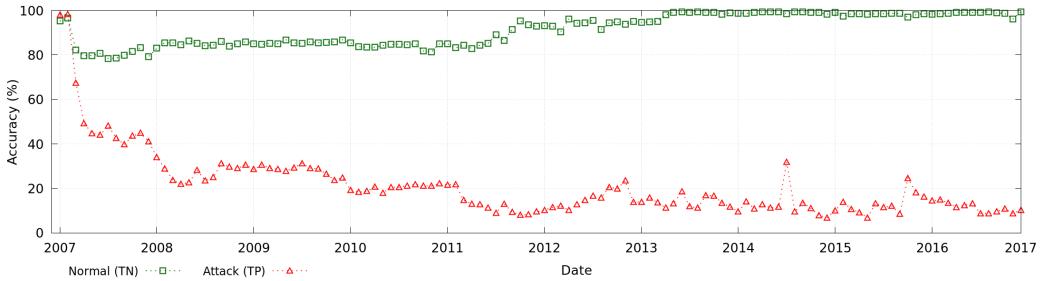


(b) RF (*Nigel View*)

[6] The results obtained using other classifiers can be foun in Appendix 3

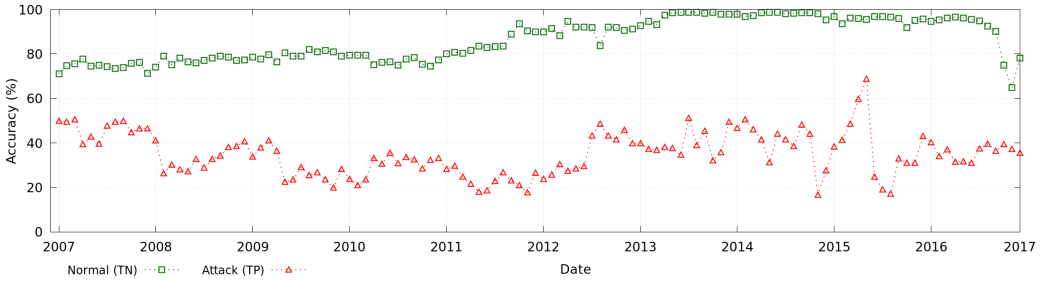
(c) RF (*Orunada View*)

(d) RF (*Moore View*)



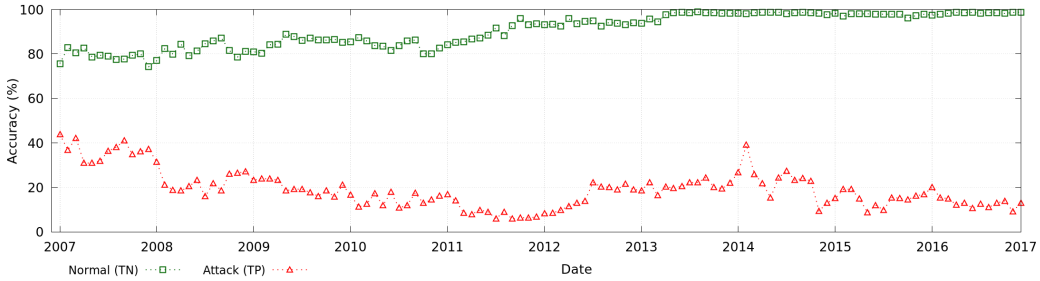
(e) RF (*All View*)

Figure 29. *Batch Learning* (Random Forest) classifier monthly accuracy according to its used feature view throughout 10 years of network traffic anomalies, only the first 60 days of 2007 are used for training. The system is not updated throughout time. Similar results are found when other *Batch Learning* classifiers are evaluated.



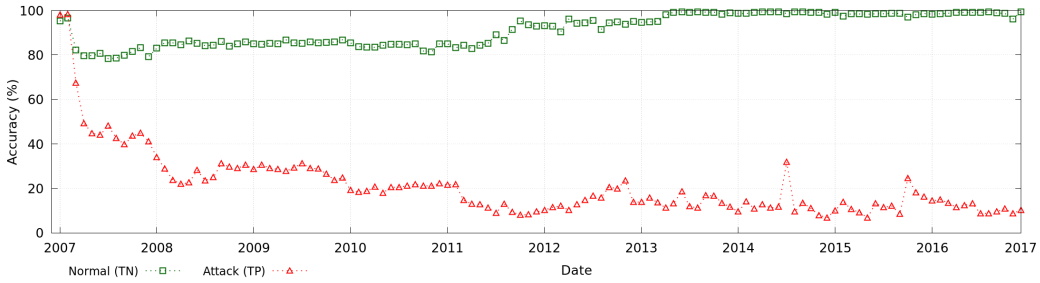
(a) OzaBoosting (*Viegas View*)

(b) OzaBoosting (*Nigel View*)



(c) OzaBoosting (*Orunada View*)

(d) OzaBoosting (*Moore View*)



(e) OzaBoosting (*All View*)

Figure 30. *Stream Learning* (OzaBoosting) classifier monthly accuracy according to its used feature view throughout 10 years of network traffic anomalies, only the first 60 days of 2007 are used for training. The system is not updated throughout time. Similar results are found when other *Stream Learning* classifiers are evaluated.

- *Normal Events Detection*: the TN rates does not significantly change over time, even increasing after 2012. However, it is important to note that this detection rate change between 2011 and 2012 occurs due to the increase of MAWI transmission link, from megabit to gigabit. Thus, significantly increasing the number and rate of normal events (Figures 21.a and 21.b). One must note that despite increasing the TN rates after such period, it does not mean that the classifiers are more reliable, but rather that there was a significant change in the normal and attack event's behavior;
- *Batch versus Stream Learning*: both batch and stream learning algorithms have presented similar outcomes. In this sense, both approaches become unreliable over time, in the absence of model updates;

In summary, this experiment gives evidence that in production, ML-based IDSs must be updated periodically, otherwise its outputs become unreliable over time. However, the

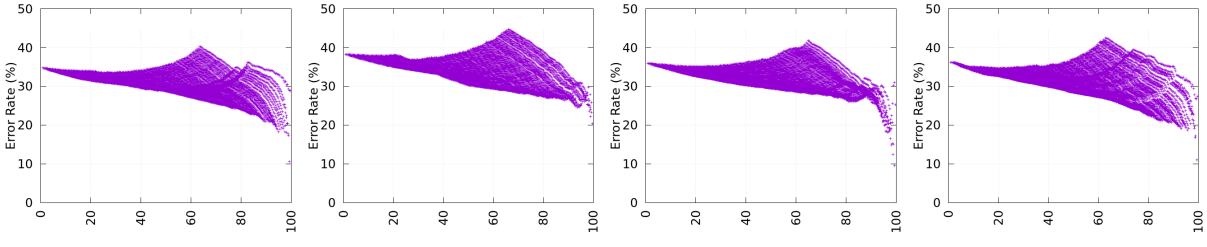
regular update of the classifier is a challenging task, because the network activity must be stored for further analysis and needs to be labeled accordingly, often demanding time and expert assistance.

6.3.2. The Problem with Classification Confidence in Intrusion Detection

This section evaluates how a traditional classification assessment approach performs when applied to *MAWIFlow*. More specifically, a class related threshold (CRT) [38] technique is evaluated, which uses a class-specific threshold to reject or not a decision according to the classifier output confidence and the assigned class.

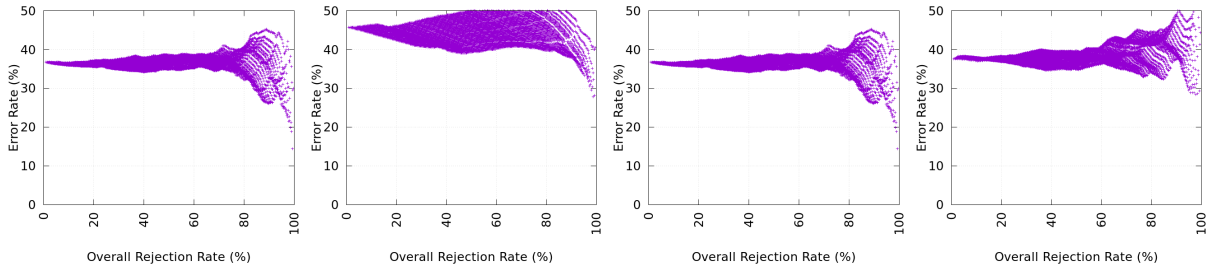
Similarly to the experiments conducted previously, the evaluated classifiers are trained using the first two months of 2007 (January and February). However, the error-reject tradeoff when CRT is applied considers only the remainder of 2007. The thresholds for both Normal and Attack classes was varied from 0.00 to 1.00 in a 0.01 basis. The error-reject tradeoff during 2007 for all of the evaluated thresholds on *MAWIFlow* is shown in the Figures 31 and 32 for the Batch and Stream Learning classifiers respectively.

It can be noted that the CRT approach fails at providing the desired level of reliability regardless of the considered classifier and view. For instance, in the best case (Nigel View with the Random Forest classifier), when a 50% rejection rate is considered, only 8% of error reduction is reached. In this sense, the behavior change over time, as occurs in MAWIFlow, renders traditional classification assessment approaches, as made by CRT, unable to provide reliability in intrusion detection, regardless of the considered rejection rate.



(a) RF (*Viegas View*) (b) RF (*Nigel View*) (c) RF (*Orunada View*) (d) RF (*Moore View*)

Figure 31. *Batch Learning* (Random Forest) classifier class related threshold (CRT) error-reject tradeoff for 2007, thresholds for both Normal and Attack thresholds were varied from 0.00 to 1.00 in a 0.01 basis, all operation points are shown. Traditional classification assessment approach, using CRT, fails at providing reliability when new network traffic behavior is occurring.



(a) OZA (*Viegas View*) (b) OZA (*Nigel View*) (c) OZA (*Orunada View*) (d) OZA (*Moore View*)

Figure 32. Stream Learning (OzaBoosting) classifier class related threshold (CRT) error-reject tradeoff for 2007, thresholds for both Normal and Attack thresholds were varied from 0.00 to 1.00 in a 0.01 basis, all operation points are shown. Traditional classification assessment approach, using CRT, fails at providing reliability when new network traffic behavior is occurring.

6.3.3. Discussion

The findings in the evaluation tests performed over *MAWIFlow* suggests that current ML-based approaches for intrusion detection lacks reliability to face production environments, regardless of the employed ML algorithm or their used feature set. Nonetheless, when a traditional reliability assessment approach is employed, such as the CRT, the error-reject tradeoff does not meet the desired level of reliability improvement.

Surprisingly, the evaluated approaches, which were found to be unreliable, due to the lack of model updates and a proper reliability assessment approach, are commonly used in the literature. Thereby, this indicates the reason that there have been many studies presenting detection schemes with low error rates, but, on the other hand, a lack of usage of such systems on production [6, 7].

Current approaches to provide classification reliability fail to reach a desired error-reject tradeoff. Such issue is mainly caused due to the network behavior changes seen in production environments, and the lack of model updates. The model update in production environment is a challenging task, which often demands expert assistance. Thereby, it demands time, not only to identify the network behavior changes, but also to rebuild the model after such change occurs, incurring in a delay of days or even weeks to build an updated classification model. Thus, it is not possible to always have an up-to-date model. In such context, to achieve reliable intrusion detection, one must first address classification reliability, regardless of the current network behavior, even in the absence of model updates.

To this end, to provide classification reliability, one must reject unreliable instances. However, the rejected instances must be stored, to either be manually inspected (not usually feasible) or further processed to identify their classes (often through an unsupervised detection scheme). To this end, the storage of such instances may become unfeasible, in a high-speed

network context. Nonetheless, as the number of rejected instances may be high, process them in real-time may not be feasible.

In such context, the model rebuilding in production becomes a challenging task when this kind of network environments is considered, such as the one present in *MAWIFlow*. The main difficulty is regarding model updates because it is a computational-expensive process, which demands time and human assistance.

6.3.4. Conformal Evaluator

In order the provide classification reliability over time, the reliable intrusion detection model relies in the use of a conformal evaluator (Section 4.6).

For evaluation purposes, the OzaBoosting stream learning algorithm was considered, using the same parameters used previously.

For the computation of the *Credibility* and *Confidence* values in the conformal evaluator, a Random Forest made of 100 decision trees were used, similarly to [24]. As a *conformity* measure (see Figure 16), the ratio of trees which classified the instance as belonging to a given class was used[7]. Thereby, *Credibility* and *Confidence* for a given classification can be computed according to Equation 11 and 12.

$$Credibility(x,c) = \frac{\sum_{i=0}^{N} \begin{cases} 0, conformity(x,c) < conformity(x_i,c) \\ 1, conformity(x,c) \geq conformity(x_i,c) \end{cases}}{N} \quad \text{Equation (11)}$$

$$Confidence(x,c) = 1 - Credibility\left(x, f(c) = \begin{cases} c = Normal, Attack \\ c = Attack, Normal \end{cases}\right) \quad \text{Equation (12)}$$

In which, x denotes the instance, c its assigned class, *conformity* a function which computes the instance conformity for the given class c, and n the number of instances with class c in a given test set. Finally, function f returns the opposite of the assigned class by the classifier, which can be either normal or attack.

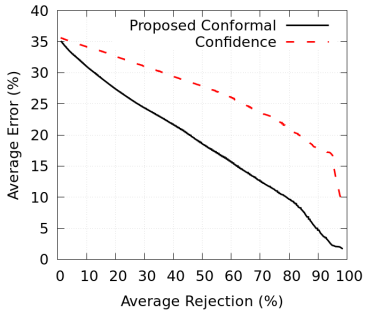
At training time, the conformal evaluator receives as input the training set, used for the classifier training (OzaBoosting). Afterwards, the training set is split in two parts, in a k-fold manner. The first split, is used to train the Random Forest classifier, while the second split is used to compute the *conformity* values, used during the *credibility* and *confidence* values computation (Equations 11 and 12).

[7] Note that, by default, random forest classifiers compute the classification confidence as the product of individual tree's confidence values

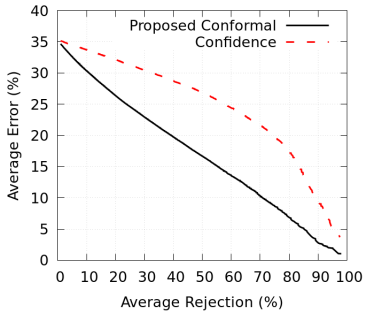
For evaluation purposes, a single view was used during the tests, made of all views used in Section 6.3.1. In addition, a complete model retraining was made every 6 months, best interval established after a series of evaluation tests. Thereby, throughout the 10 years of *MAWIFlow* dataset, both conformal evaluator and OzaBoosting classifiers were retrained 20 times. For each system retraining, OzaBoosting and conformal evaluator, two months of data was used, both classifier and conformal evaluator are not updated after such period.

The proposed conformal evaluator was evaluated regarding its error-reject tradeoff throughout the 10 years of *MAWIFlow* dataset, according to the model lifetime interval (6 months). For this purpose, conformal evaluator was compared with the class-related threshold (CRT), as measured by the OzaBoosting confidence values. Similarly to CRT, the conformal evaluator also used different class thresholds (t_{class}) for each class. Figure 33 shows the relation between the average error rate and the average rejection rate for the OzaBoosting for each model lifetime, in the first three years of *MAWIFlow* dataset (see Appendix 2, for the complete evaluation of the conformal evaluator). 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.

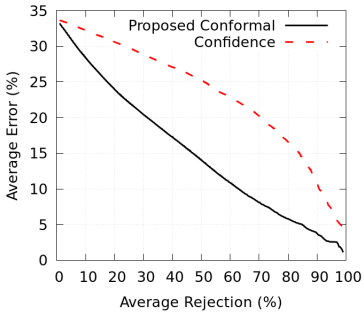
The proposed conformal evaluator, significantly improved the error-reject tradeoff when compared to the CRT approach. For all evaluated model lifetimes (20 time intervals in the complete *MAWIFlow* dataset), the conformal evaluator enabled to reduce the error rate, with a significantly lower rejection rate, when compared to the CRT approach.



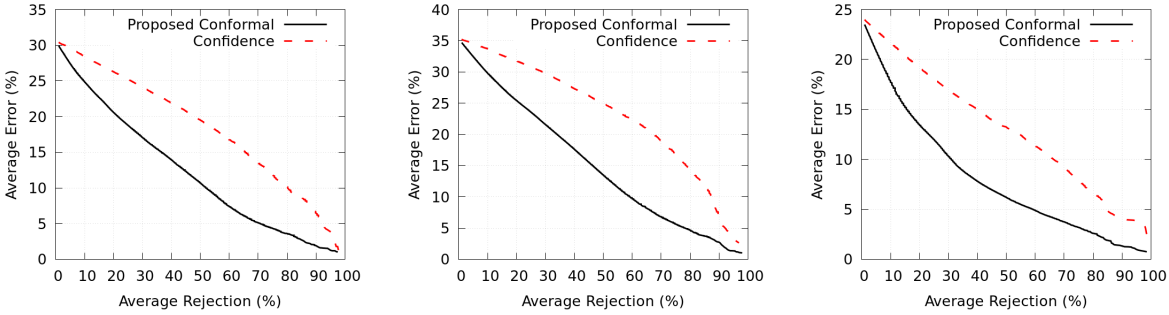
(a) Jan. 2007 to Jun. 2007



(b) Jul. 2007 to Dec. 2007



(c) Jan. 2008 to Jun. 2008



(d) Jul. 2008 to Dec. 2008 (e) Jan. 2009 to Jun. 2009 (f) Jul. 2009 to Dec. 2009

Figure 33. Conformal evaluator error-reject tradeoff, compared to class-related threshold (CRT) as measured by OzaBoosting classifier, in the first three years of the *MAWIFlow* dataset.

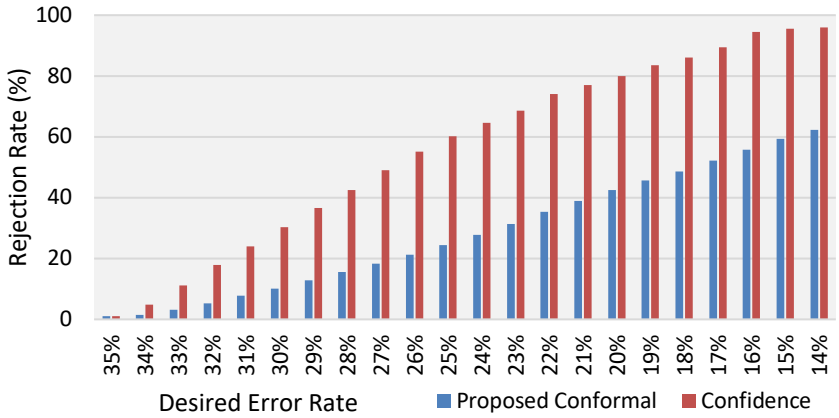


Figure 34. Conformal evaluator error-reject tradeoff, compared to class-related threshold (CRT) as measured by OzaBoosting classifier, in Jan. 2007 to Jun. 2007 in *MAWIFlow* dataset.

It becomes possible to note that the proposed conformal evaluator enables to maintain the system reliability, even in the absence of model updates. Figure 34 shows a comparison regarding the model lifetime in Jan. 2007 to Jun. 2007 (Figure 33-a) with the CRT approach, according to a desired error rate and the necessary rejection rate. For instance, the proposed conformal evaluator was able to reach 20 percent of error rate while rejecting 42 percent of instances, as opposed to 80 percent of rejection, to reach the same error rate level for the CRT approach.

Finally, the proposed conformal evaluator enabled to measure the classification reliability, even in the absence of model updates (for up to four months). Such property, is desired to enable reliable usage of ML-based schemes for network-based intrusion detection. In such context, due to the high network throughput, and the need to ongoing update the intrusion detection mechanism, the conformal evaluator may aid the system administrator at establishing the reliability of his system. Moreover, conformal evaluator may assess individual classifications regarding their reliability, i.e. whether the classification should be accepted or

not, but also, to establish the detection mechanism lifespan. For instance, a high level of rejection rate may indicate that a system retraining is needed.

6.3.5. Adapting to Network Behavior Changes

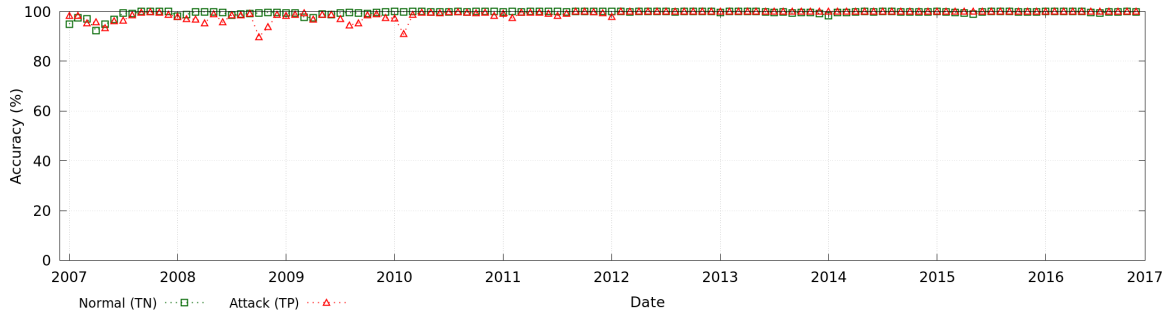
Finally, the proposed reliable intrusion detection model was evaluated regarding its capacity to adapt to network behavior changes over time. For evaluation purposes, the same parameters used for the evaluation of the proposed conformal evaluator was considered (Section 6.3.4).

In addition, for the periodic label request process, the uncertainty sampling [92] algorithm was used as a ranking method. However, the uncertainty value is computed by the means of the proposed conformal evaluator, thus, using the $Reliability_{degree}$, rather than the classifier confidence values. Finally, the label propagation process is made labeling nearby instances according to their Euclidean distance.

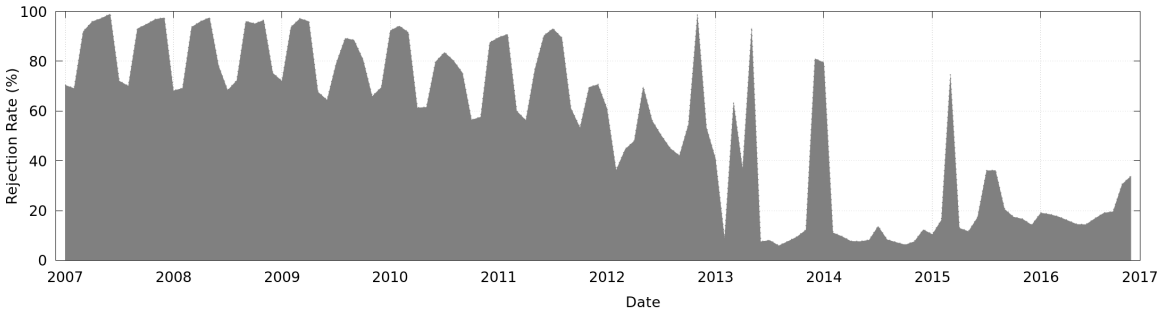
The first evaluation aim to evaluate the proposed reliable intrusion detection model without updates, with conformal evaluator, and with a complete retraining every 6 months. For the sake of simplicity, the results are shown using, in all cases, a daily 1% of label request upon the rejected instances. For the label propagation, a 0.5 radius was used, which have yield the best results, note that the used feature set comprises all features (158). During the evaluation tests a single rejection threshold was used (t_{class}), as the rejection rate, and reliability improvement, must be established according to the administrator needs. Thereby, a 0.7 t_{class} value was used as a threshold for the conformal evaluator for both classes.

Figure 35 shows the accuracy and rejection rate throughout the 10 years of *MAWIFlow* without performing updates. It is possible to note that, despite the high rejection rate, the monthly accuracy rates, during 10 years, does not significantly change over time. In the worst case, the accuracy for attack events (TP), drops to 89%, as opposed to 96% at training time, in October 2008. Regarding the detection of normal events (TN), the worst accuracy was observed in April 2007, yielding 92% of TN.

Regarding the rejection rate, it is possible to note, that the conformal evaluator rejects less events, during the period used for training time. However, in general, the rejection rate increases after such period. Thereby, serving as an indicator that the model must be either retrained or updated. Moreover, in specific periods of time, the rejection rate does not significantly increase, for instance from 2014 to first semester of 2015, thus, one could also use the conformal evaluator rejection rate as an indicator of the model lifespan.



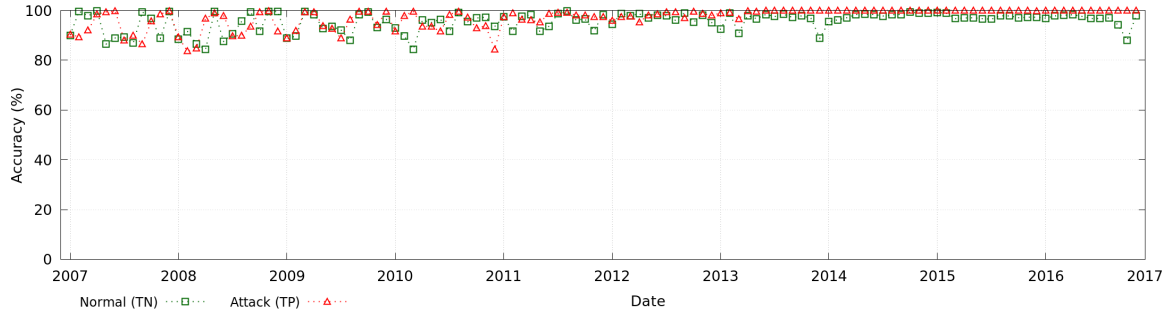
(a) Accuracy Rates



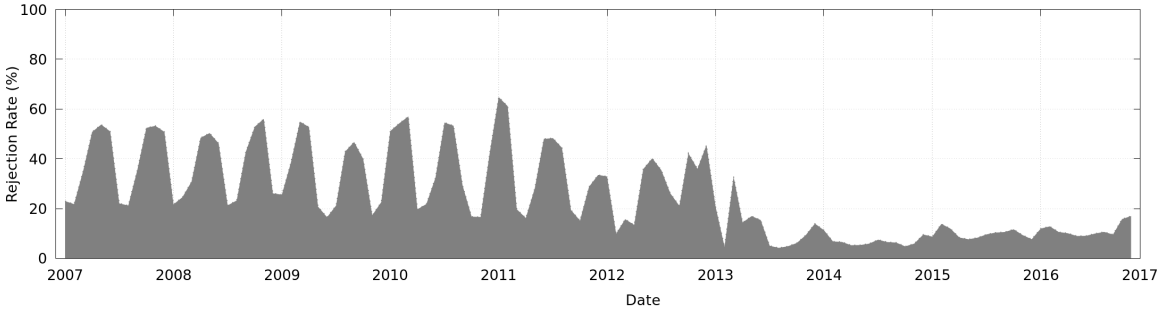
(b) Rejection Rate

Figure 35. Reliable intrusion detection model performance throughout the 10 years of *MAWIFlow*. System was updated every 6 months, and not incrementally updated afterwards. For the sake of simplicity, a 0.7 t_{class} for both classes was used in all cases.

The second evaluation aimed to measure the impact that incremental model updates yield to accuracy and rejection rates. To this end, several tests, were performed to measure the impact that lowering the t_{class} value from the conformal evaluator yield to the system accuracy, when incremental model updates are performed. Figure 35 shows the accuracy and rejection rate when incremental model updates are performed, using a t_{class} value of 0.4 for both classes.



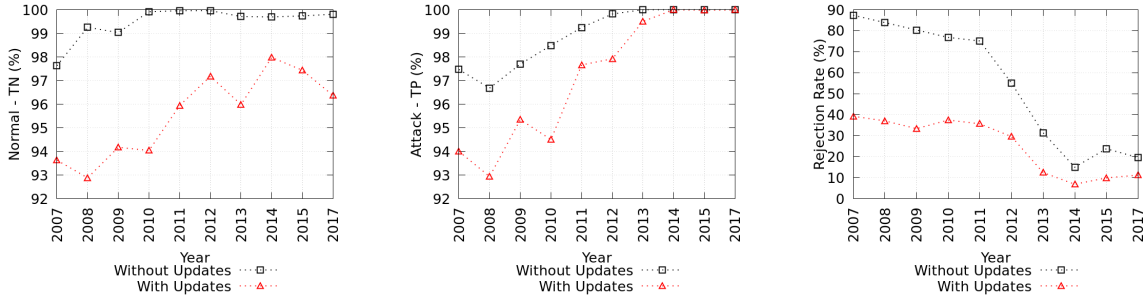
(a) Accuracy Rates



(b) Rejection Rate

Figure 36. Reliable intrusion detection model performance throughout the 10 years of *MAWIFlow*. System was updated every 6 months, and incrementally updated with 1% of rejected instances every day. For the sake of simplicity, a 0.4 t_{class} for both classes was used in all cases.

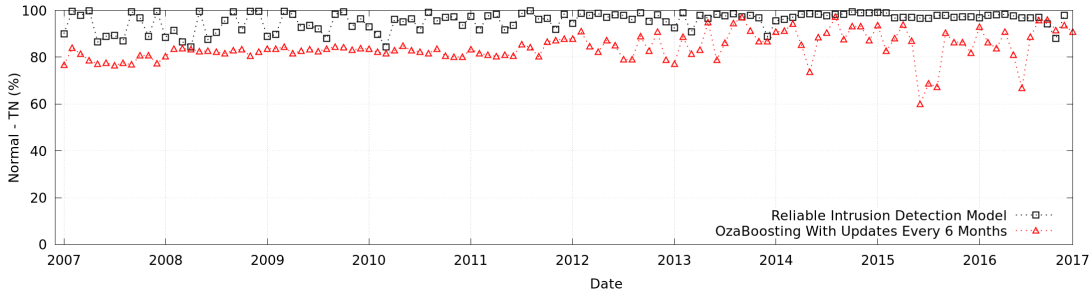
It is possible to note a significant decrease in the system rejection rate, while presenting similar detection rates. Thereby, the task of performing incremental model updates, 1% of rejected instances in this evaluation, enables to significantly reduce the t_{class} values, and, as a consequence, decrease the rejection rate, without significant impact on accuracy. Figure 36 shows a comparison regarding the system rejection and accuracy rates when model is incrementally updated and when it is not.



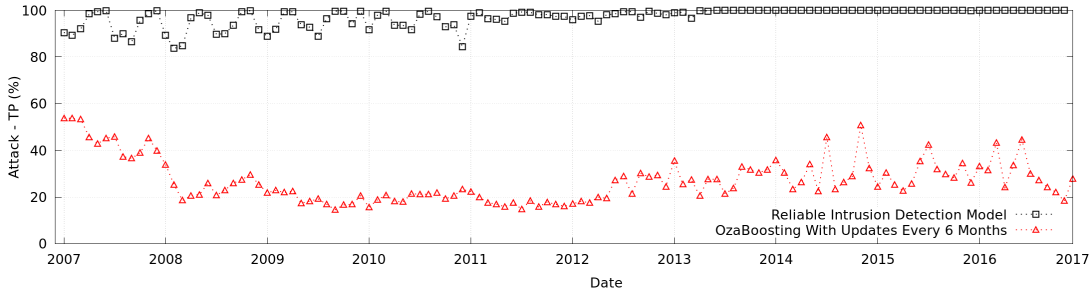
(a) Normal Accuracy Rates (b) Attack Accuracy Rates (c) Rejection Rates

Figure 37. Yearly accuracy and rejection rates comparison as obtained by the proposed reliable intrusion detection model with and without incremental model updates.

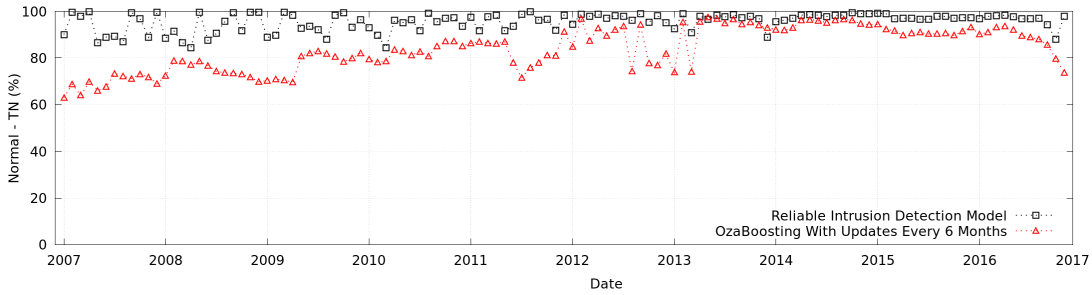
When the model is incrementally updated using only 1% of rejected instances, the proposed approach is able to decrease the rejection rate up to a ratio of 1.17. However, as a tradeoff, in average it incurs in the reduction of TN rates by 3.91% and TP rates by 1.75%. It is important to note, that the accuracy and rejection rates, should be established according to the administrator needs. For instance, if a higher instance label request was used, the accuracy rates could be further increased, while keeping the rejection rates lower.



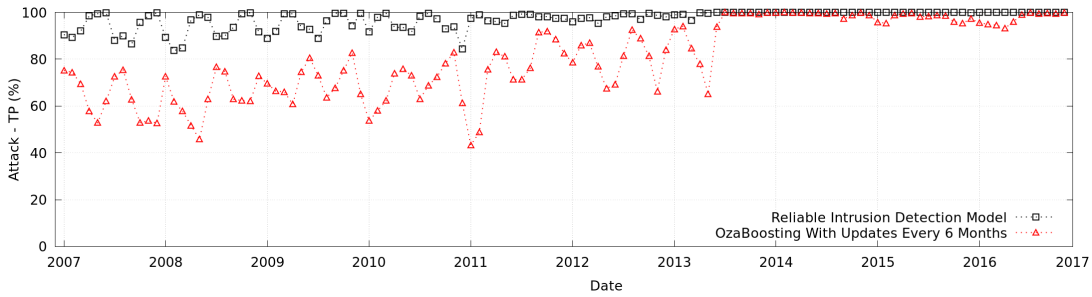
(a) Reliable intrusion detection model normal accuracy (TN) versus not updated OzaBoosting



(b) Reliable intrusion detection model attack accuracy (TP) versus not updated OzaBoosting



(c) Reliable intrusion detection model normal accuracy (TN) versus OzaBoosting updated every semester



(d) Reliable intrusion detection model attack accuracy (TP) versus OzaBoosting updated every semester

Figure 38. Monthly accuracy and rejection rates comparison as obtained by the proposed reliable intrusion detection model with and without incremental model updates.

Figure 38 shows a comparison with the reliable intrusion detection model, with 1% of instance label request (Figure 35), and the system accuracy rates without incremental model nor the conformal evaluator.

6.3.6. Discussion

Although the proposal was able to significantly decrease the average rejection rate by the means of incremental model updates (Figure 35), 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 a high rejection rate point (0.7 t_{class} values). 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 the proposal was able to significantly reject less instances when the incremental model updates are performed. This indicates that, the average rejection rate will significantly decrease over time if an expert provides a subset of event's label over time. 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 proposal rejection rate can be even further decreased. Nonetheless, the labeling process can be made automatically, if a labeling delay can be tolerated for instance. This approach can be used in production as the conformal evaluator enables to maintain the system reliability despite the model being updated or not.

The most important benefit of the 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, since the conformal evaluator is used. The action the administrator will perform is under her/his discretion. 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.

6.4 *BigFlow - Dealing with High-speed Networks*

Finally, the proposal was implemented in a distributed environment for the evaluation of its throughput. To this end, a *BigFlow* prototype was implemented and deployed in a distributed environment, as shown in Figure 39. The prototype takes as input network packet

headers from MAWI [16], 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.

The prototype was implemented on top of Apache Flink stream processing framework [30], 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 *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.

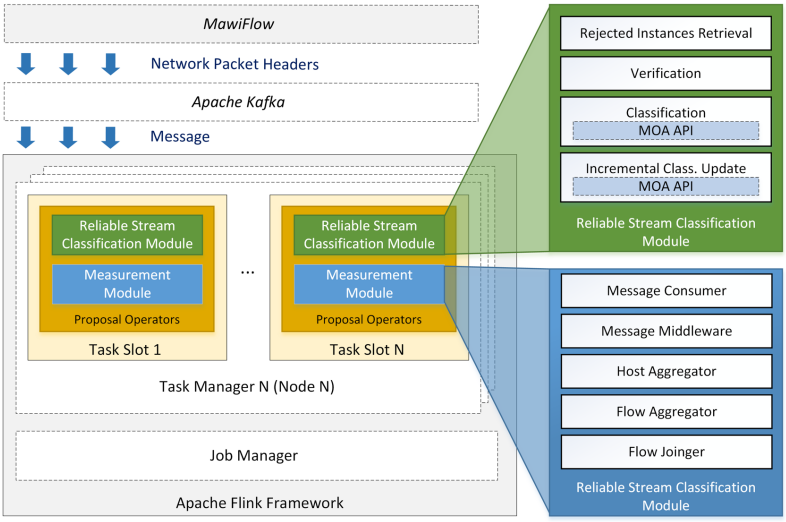


Figure 39. *BigFlow* prototype architecture.

For the evaluation, only the reliable stream learning classification module was implemented using the massive online analysis (MOA) library [91], release 16.04 (further details over such classification mechanism can be found in [60]). 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 39), which retrieved the rejected instances through Kafka. The PE parallelism level was set according to the number of worker nodes used in the experimental evaluation.

For evaluating the scalability of the prototype, a 12-node cluster in a single rack was set-up, connected through a 10 GbE interface. Each node has a 4-core CPU with 8 GB of memory. In all considered experiments, the BigFlow prototype (Figure 39) was set-up 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,

only the feature set from Viegas [5] view 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 40 shows the throughput and performance breakdown. 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.

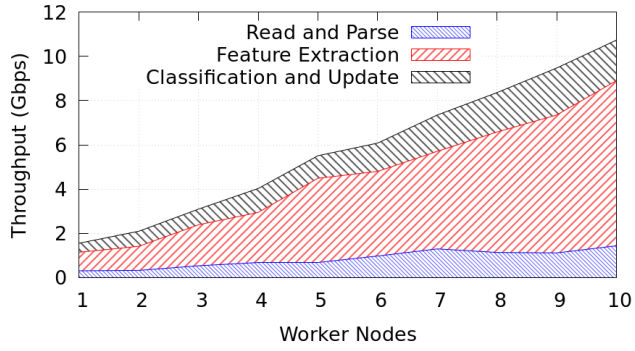


Figure 40. *BigFlow* throughput performance according to the number of worker nodes.

Figure 41 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).

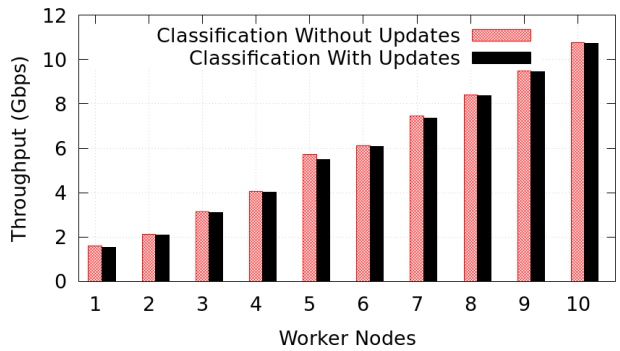


Figure 41. *BigFlow* prototype architecture.

Finally, Table 18 shows the weekly training time and required storage for several evaluated classifiers, 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 Decision Tree, Random Forest, Gradient Boosting, Ensemble, and Hoeffding tree classifiers.

Table 18. 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**

6.5 *Final Considerations*

In this chapter a series of evaluations were presented concerning the methods implemented in this work. The experiments regarding traditional model building methods revealed that the assumptions commonly used in the literature does not hold when ML techniques are considered. The experiments presented in Section 6.1 contemplates the methods regarding the building of reliable batch learning models. In these experiments, the lack of generalization and reliability in face of new network traffic behavior of current approaches in the literature was evidenced. The results regarding the building of generalization capable models showed that it was possible to build batch learning models able to generalize the behavior from the training dataset. In addition, the method for ensuring reliability in face of unknown network traffic behavior, allowed to ensure classification reliability even in the presence of unknown network traffic. The experiments presented in Section 6.2 contemplate the evaluation of a resilient to adversarial attacks stream learning approach. The proposed model was able to provide resiliency to both *causative* and *exploratory* attacks. In Section 6.3, the proposed reliable intrusion detection model was evaluated regarding its classification reliability over time. The findings in the evaluation tests, showed that the proposed approach was able to provide reliable classification of new network traffic, even in the absence of model updates. In addition, when only 1% of rejected instances are used for incremental model update, the proposed approach significantly decreases the rejection rate. Finally, Section 6.4 shows the prototype and evaluation of *BigFlow*. The proposed approach for near real-time network traffic classification was scalable. In the next chapter the final conclusions alongside with future works are presented.

Chapter 7

Conclusion

Network-based intrusion detection have been extensively studied in the literature over the last years. In this sense, a popular approach often consists of detecting intrusion attempts by the means of machine learning algorithms. However, current approaches for network traffic classifications are not able to meet the desired throughput neither deal with the evolving behavior of network traffic in a reliable manner.

This work has addressed each of the challenges of building reliable intrusion detection schemes by the means of machine learning techniques for production usage. To this end, the approach proposed in this work, namely reliable intrusion detection model, relies in the use of both batch and stream learning algorithms coped together, in which, each learner overcomes a specific challenge.

A novel approach for building generalization capable batch learning models enabled to build models able to generalize the behavior from the training dataset. The proposed approach tackles the challenge of obtaining realistic intrusion datasets for model building process, by generalizing the behavior from a limited training set to a wider one, according to each of the production environment characteristics.

In addition, to overcome the challenge of providing reliable classifications in face of new network traffic behavior, a new lightweight rejection approach has been proposed. The proposed approach ensures that the classified instance presents a similar behavior to that of the training dataset. Thereby, tackling the challenge of providing reliability of classifications in face of unknown traffic behavior.

Regarding the classification of new network traffic over time. First, a new unsupervised stream learning model was proposed to provide resiliency to adversarial attacks. The proposed approach was able to provide resiliency to both *causative* and *exploratory* attacks targeted at the model. Thereby, enable the system to reliably adapt to changes when unsupervised stream learning algorithms are used for the classification of network traffic.

Afterwards, an approach for providing classification reliability even in the absence of model updates was presented. The approach, by the means of a conformal evaluator, enable to ensure the classification reliability for up to 4 months since the last model update. Through the proposed conformal evaluator, the reliable intrusion detection model was able to provide reliable classifications throughout 10 years of network traffic. In addition, the proposed approach was able to significantly decrease the system rejection rate, when incremental model updates are performed, with only 1% of rejected instances.

Finally, the challenge of performing network-based intrusion detection at high-speed networks was addressed. The proposed approach named *BigFlow* and the experimental evaluation showed that *BigFlow* is feasible: the prototype could reach up to 10.72 Gbps throughput in a 40-core cluster.

6.1 Future Works

This work presents the following future works:

- Through the built datasets, evaluate common approaches in the literature for network-based intrusion detection, and establish a set of guidelines towards reliability;
- Evaluate common active learning approaches for stream learning in the *MAWIFlow;*
- Design a new method coped with conformal evaluator to assess the model lifespan, in order to establish whether a model should be rebuilt or not;
- Design a method to use a multiple classifier system in order to aid at providing reliability in the absence of model updates;
- Design a method for building models aiming to increase their lifespan in *MAWIFlow;*

6.2 Publications

In this section a list of papers either published or under review are listed according to the challenge addressed by the reliable intrusion detection model.

6.2.1. Generalization Capable Models

Conference: **VIEGAS, E. K.**; SANTIN, A. O. ; ABREU, V. ; OLIVEIRA, LUIZ S. *Enabling Anmaly-based Intrusion Detection Through Model Generalization*. Accepted at: IEEE Symposium on Computers and Communications 2018 (ISCC). **(Brazilian qualis A2)**

Journal: **VIEGAS, E. K.** ; SANTIN, A. O. ; OLIVEIRA, L. E. S. Toward a Reliable Anomaly-Based Intrusion Detection in Real-World Environments. Elsevier Computer Networks, vol. 127, no. 1, pp. 200-216, 2017 **(Brazilian qualis A1)**

Journal: **E. VIEGAS**, A. SANTIN, A. FRANÇA, R. JASINSKI, V. PEDRONI, AND L. OLIVEIRA, "Towards an Energy-Efficient Anomaly-Based Intrusion Detection Engine for Embedded Systems," IEEE Transactions on Computers, vol. 66, no. 1, pp. 1–14, 2017. **(Brazilian qualis A1)**

6.2.2. Reliability in Batch Learning Classification

Journal: **E. VIEGAS**, A. SANTIN, L. OLIVEIRA , A. FRANÇA, R. JASINSKI, and V. PEDRONI, "A reliable and Energy-Efficient Classifier Combination Scheme for Intrusion Detection in Embedded Systems". Accepted at: Computers & Security **(Brazilian qualis A2)**

6.2.3. Resiliency to Adversarial Attacks

Conference: **VIEGAS, E. K**. ; SANTIN, A. O. ; ABREU, V. ; OLIVEIRA, L. E. S. Stream Learning and Anomaly-based Intrusion Detection in the Adversarial Settings. In: IEEE Symposium on Computers and Communications (ISCC), 2017, Creete, Greece. In Proceddings of IEEE ISCC. Los alamitos: IEEE, 2017. p. 1-6. **(Brazilian qualis A2)**

Conference: **VIEGAS, E. K.**; SANTIN, A. O. ; ABREU, V. ; OLIVEIRA, LUIZ S. . Detecção de Intrusão Através de Aprendizagem de Fluxo no Ambiente do Adversário. In: Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg), 2017, 2017. p. 1-14. **(Brazilian qualis B3)**

Conference: **VIEGAS, E**.; SANTIN, A.; NEVES, N.; BESSANI, A.; ABREU, V.; A Resilient Stream Learning Intrusion Detection Mechanism for Real-time Analysis of Network Traffic. In. proc. of IEEE GLOBECOM, Singapore, Singapore. Proceeding of Globecom 2017. Los Alamitos: IEEE, 2017. **(Brazilian qualis A1)**

6.2.4. High-speed Networks

Journal: **VIEGAS, E.;** SANTIN, A.; NEVES, N.; BESSANI, A.; "BigFlow: Real-time and Reliable Anomaly-based Intrusion Detection for High-speed Networks." Major Review at an Special Issue in: Future Generation Computer Systems, Elsevier (FGCS) **(Brazilian qualis A2)**

6.2.5. Conformal Evaluator and Reliable Intrusion Detection Model

Journal: **E. VIEGAS**, A. SANTIN, BESSANI, A., N. NEVES, "Reliable Intrusion Detection: Dealing with 10 years of Network Traffic Anomalies". Will be submitted to: IEEE Transactions on Information Forensics and Security. **(Brazilian qualis A1)**

6.2.6. All

Patent: **E. VIEGAS**, A. SANTIN, “Mecanismo de Detecção de Intrusão Confiável Baseada em Machine Learning em Redes de Alta Velocidade”, Patente de Invenção, Patente BR 10 2018 011016 0, Data de depósito 30/05/2018,

References

[1] CISCO. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016 – 2021, 2017

[2] P802.3cd – Standard for Ethernet Amendment [Online]. Available: http://ieeexplore.ieee.org/document/8115318/

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

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

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

[6] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," *2010 IEEE Symp. Secur. Priv.*, vol. 0, no. May, pp. 305–316, 2010.

[7] C. Gates and C. Taylor, "Challenging the Anomaly Detection Paradigm: A Provocative Discussion," *Proc. 2006 Work. New Secur. Paradig.*, pp. 21–29, 2007.

[8] R. Fontugne, P. Abry, K. Fukuda, D. Veitch, K. Cho, P. Borgnat, and H. Wendt, "Scaling in Internet Traffic: A 14 Year and 3 Day Longitudinal Study, with Multiscale Analyses and Random Projections," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2152–2165, 2017.

[9] P. Borgnat, G. Dewaele, K. Fukuda, P. Abry, and K. Cho, "Seven years and one day: Sketching the evolution of internet traffic," *Proc. - IEEE INFOCOM*, pp. 711–719, 2009.,

[10] M. Tavallaee, N. Stakhanova, and A. A. Ghorbani, "Toward credible evaluation of anomaly-based intrusion-detection methods," *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 40, no. 5, pp. 516–524, 2010.

[11] Snort. Snort Open-source Intrusion Prevention System. [Online]. Available: https://www.snort.org/

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

[13] R. Fontugne, J. Mazel, and K. Fukuda, "Hashdoop: A MapReduce framework for network anomaly detection.," *INFOCOM Work.*, pp. 494–499, 2014.

[14] HDFS. HDFS, Hadoop Distributed Filesystem [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[15] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking," *Proc. 6th Int. Conf. - Co-NEXT '10*, pp. 1–12, 2010.

[16] MAWI. MAWI Working Group Traffic Archive. [Online]. Available: http://mawi.wide.ad.jp/mawi/

[17] G. Creech and J. Hu, "Generation of a new IDS test dataset: Time to retire the KDD collection," *IEEE*

Wirel. Commun. Netw. Conf. WCNC, pp. 4487–4492, 2013.

[18] A. M. White, A. R. Matthews, K. Z. Snow, and F. Monrose, "Phonotactic reconstruction of encrypted VoIP conversations: Hookt on fon-iks," *Proc. - IEEE Symp. Secur. Priv.*, pp. 3–18, 2011.

[19] K. Kendall, "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems". *Thesis Submitted to the Department of Electrical Engineering and Computer Science*, 1999.

[20] S. Brugger and J. Chow, "An assessment of the DARPA IDS Evaluation Dataset using Snort," *UCDAVIS Dep. Comput. Sci.*, pp. 1–19, 2007.

[21] 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.*, vol. 3, no. 4, pp. 262–294, 2000.

[22] E. K. Viegas, A. O. Santin, and L. S. Oliveira, "Toward a reliable anomaly-based intrusion detection in real-world environments," *Comput. Networks*, vol. 127, 2017.

[23] J. D. Tygar, "Adversarial Machine Learning", *IEEE Internet Comput*, vol. 15, no. 5, pp. 4-6, 2011

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

[25] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion detection and Big Heterogeneous Data : a Survey," *Journal of Big Data*, pp. 1-41, 2015.

[26] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mob. Networks Appl.*, vol. 19, no. 2, pp. 171–209, 2014.

[27] Hadoop. Apache Hadoop. [Online]. Available: http://hadoop.apache.org/

[28] MapReduce. MapReduce Programming Model. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

[29] Storm. Apache Storm. [Online]. Available: http://storm.apache.org/

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

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

[32] J. Gama, "A survey on learning from data streams: current and future trends," *Prog. Artif. Intell.*, vol. 1, no. 1, pp. 45–55, 2012.

[33] Y. Fu, X. Zhu, and B. Li, "A survey on instance selection for active learning," *Knowl. Inf. Syst.*, vol. 35, no. 2, pp. 249–283, 2013.

[34] Apache MLib. Apache MLib. [Online]. Available: https://spark.apache.org/mllib/

[35] StreamDM. StreamDM [Online]. Available: http://huawei-noah.github.io/streamDM/

[36] A. Shiravi, H. Shiravi, M. Tavallaee, and A. a. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Secur.*, vol. 31, no. 3, pp. 357–374, 2012.

[37] C. K. Chow, "On Optimum Recognition Error and Reject Tradeoff," *IEEE Trans. Inf. Theory*, vol. 16, no. 1, pp. 41–46, 1970.

[38] Giorgio Fumera, Fabio Roli, and Giorgio Giacinto, "Multiple reject thresholds for improving classification reliability". *Advances in Pattern Recognition: Joint IAPR International Workshops, SSPR 2000 and SPR 2000*. Alicante, Spain, pp. 863.

[39] E. Viegas, A. Santin, V. Abreu, and L. S. Oliveira, “Stream learning and anomaly-based intrusion detection in the adversarial settings,” in *Proceedings - IEEE Symposium on Computers and Communications*, 2017.

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

[41] V. K. C. Bumgardner and V. W. Marek, “Scalable hybrid stream and hadoop network analysis system,” *Proc. 5th ACM/SPEC Int. Conf. Perform. Eng. - ICPE ’14*, pp. 219–224, 2014.

[42] Apache Metron. Apache Metron Real-time Big Data Security. [Online]. Available: http://metron.apache.org/

[43] E. Viegas, A. Santin, N. Neves, A. Bessani, and V. Abreu, “A Resilient Stream Learning Intrusion Detection Mechanism for Real-time Analysis of Network Traffic”. *In. proc. of IEEE GLOBECOM, Singapore, Singapore. Proceeding of Globecom 2017*. Los Alamitos: IEEE, 2017

[44] G. Hulten, L. Spencer, and P. Domingos, “Mining Time-changing Data Streams,” in Proc. 7th ACM *SIGKDD Int. Conf. Knowl. Discovery Data Mining,* pp. 97–106, 2001.

[45] D. E. Denning, “An intrusion-detection model,” Proc. - IEEE Symp. Secur. Priv., no. 2, pp. 118–131, 2012.

[46] D. Canali, M. Cova, G. Vigna, and 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., pp. 197–206, 2011.

[47] A. I. Abubakar, H. Chiroma, S. A. Muaz, and L. B. Ila, “A review of the advances in cyber security benchmark datasets for evaluating data-driven based intrusion detection systems,” Procedia Comput. Sci., vol. 62, no. Scse, pp. 221–227, 2015.

[48] V. Paxson and S. Floyd, “Wide-Area Traffic: The Failure of Poisson Modeling,” IEEE/ACM Trans. Netw., vol. 3, no. 3, pp. 226–244, 1995.

[49] L. S. Oliveira, R. Sabourin, F. Bortolozzi, and C. Y. Suen, “Automatic recognition of handwritten numerical strings: A Recognition and Verification strategy,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 24, no. 11, pp. 1438–1454, 2002.

[50] G. D. C. Cavalcanti, L. S. Oliveira, T. J. M. Moura, and G. V. Carvalho, “Combining diversity measures for ensemble pruning,” Pattern Recognit. Lett., vol. 74, pp. 38–45, 2016.

[51] B. Nelson, M. Barreno, F.J. Chi, A.D. Joseph, B.I.P. Rubinstein, U. Saini, C. Sutton, J.D. Tygar, K. Xia, "Exploiting Machine Learning to Subvert Your Spam Filter", Proc. First Workshop Large-Scale Exploits and Emergent Threats, pp. 1-9, 2008.

[52] L. Huang, A.D. Joseph, B. Nelson, B. Rubinstein, and J.D. Tygar, "Adversarial Machine Learning," Proc. Fourth ACM Workshop Artificial Intelligence and Security, pp. 43-57, 2011.

[53] N. Srndic and P. Laskov, “Practical evasion of a learning-based classifier: A case study,” Proc. - IEEE Symp. Secur. Priv., pp. 197–211, 2014.

[54] G. Wang, S. Barbara, T. Wang, H. Zheng, and B. Y. Zhao, “Man vs . Machine : Practical Adversarial Detection of Malicious Crowdsourcing Workers,” 23rd USENIX Secur. Symp., pp. 239–254, 2014.

[55] B. I. P. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S. Lau, S. Rao, N. Taft and J. D. Tygar, "ANTIDOTE : Understanding and Defending against," SIGCOMM, no. November, pp. 1–14, 2009.

[56] B. Hanczar and E. R. Dougherty, "Classification with reject option in gene expression data," *Bioinformatics*, vol. 24, no. 17, pp. 1889–1895, 2008.

[57] D. P. P. Mesquita, L. S. Rocha, J. P. P. Gomes, and A. R. Rocha Neto, "Classification with reject option for software defect prediction," *Appl. Soft Comput. J.*, vol. 49, pp. 1085–1093, 2016.

[58] J. R. Navarro-Cerdan, J. Arlandis, R. Llobet, and J. C. Perez-Cortes, "Batch-adaptive rejection threshold estimation with application to OCR post-processing," *Expert Syst. Appl.*, vol. 42, no. 21, pp. 8111–8122, 2015.

[59] L. S. Oliveira, R. Sabourin, F. Bortolozzi, and C. Y. Suen, "Automatic recognition of handwritten numerical strings: A Recognition and Verification strategy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 11, pp. 1438–1454, 2002.

[60] E. Viegas, A. Santin, N. Neves, and A. Bessani. "BigFlow: Real-time and Reliable Anomaly-based Intrusion Detection for High-speed Networks". *Under Major Review at: Future Generation Computer System*

[61] A. Kantchelian, S. Afroz, L. Huang, A. C. Islam, B. Miller, M. C. Tschantz, R. Greenstadt, A. D. Joseph, and J. D. Tygar, "Approaches to adversarial drift," *Proc. 2013 ACM Work. Artif. Intell. Secur. - AISec '13*, pp. 99–110, 2013.

[62] F. Maggi, W. Robertson, C. Kruegel, and G. Vigna, "Protecting a moving target: Addressing web application concept drift," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5758 LNCS, pp. 21–40, 2009.

[63] M. Gudadhe, Prakash Prasad, and Kapil Wankhade. "A new data mining based network intrusion detection model," *Int. Conf. on Computer & Com. Technology (ICCCT)*, pp. 0–4, 2010.

[64] D. P. Gaikwad and R. C. Thool, "Intrusion detection system using Bagging with Partial Decision Tree base classifier," *Procedia Comput. Sci.*, vol. 49, no. 1, pp. 92–98, 2015.

[65] N. F. Haq, A. R. Onik, and F. M. Shah, "An ensemble framework of anomaly detection using hybridized feature selection approach (HFSA)," *IntelliSys 2015 - Proc. 2015 SAI Intell. Syst. Conf.*, pp. 989–995, 2015.

[66] J. Jabez and B. Muthukumar, "Intrusion Detection System (IDS): Anomaly Detection Using Outlier Detection Approach," Procedia Comput. Sci., vol. 48, no. Iccc, pp. 338–346, 2015.

[67] I. Syarif, A. Prugel-Bennett, and G. Wills, "Data mining approaches for network intrusion detection: from dimensionality reduction to misuse and anomaly detection," J. Inf. Technol. Rev., vol. 3, no. 2, pp. 70–83, 2012.

[68] S. Y. Ji, B. K. Jeong, S. Choi, and D. H. Jeong, "A multi-level intrusion detection method for abnormal network behaviors," J. Netw. Comput. Appl., vol. 62, pp. 9–17, 2016.

[69] C. K. Olivo, A. O. Santin, and L. S. Oliveira, "Obtaining the threat model for e-mail phishing," Appl. Soft Comput. J., vol. 13, no. 12, pp. 4841–4848, 2013.

[70] L. Breiman, "Random forests," Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.

[71] D. Meyer, "Support Vector Machines: interface to libsvm in e1071," Engineering, vol. 1, no. December, pp. 1–8, 2009.

[72] J. Yang, Y. Qiao, X. Zhang, H. He, F. Liu, and G. Cheng, "Characterizing user behavior in mobile internet," IEEE Trans. Emerg. Top. Comput., vol. 3, no. 1, pp. 95–106, 2015.

[73] A. Milenkoski, M. Vieira, S. Kounev, A. Avritzer, and B. D. Payne, "Evaluating Computer Intrusion

Detection Systems: A Survey of Common Practices," ACM Comput. Surv., vol. 48, no. 1, pp. 1–41, 2015.

[74] D. Darmon, J. Sylvester, M. Girvan, and W. Rand, "Understanding the Predictive Power of Computational Mechanics and Echo State Networks in Social Media," Human, vol. 2, no. 1, pp. 13–24, 2013.

[75] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015-2020 White Paper, 2017, Document ID:1454457600805266

[76] HoneyD. [Online] Available: http://www.honeyd.org/

[77] T. Heimann, P. Mountney, M. John, and R. Ionasec, "Learning without labeling: Domain adaptation for ultrasound transducer localization," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 8151 LNCS, no. PART 3, pp. 49–56, 2013.

[78] J. Dromard, G. Roudiere, and P. Owezarski, "Online and Scalable Unsupervised Network Anomaly Detection Method," IEEE Trans. Netw. Serv. Manag., vol. 14, no. 1, pp. 34–47, 2017.

[79] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 5, p. 5, 2006.

[80] A. Moore, D. Zuev, and M. Crogan, "Discriminators for use in flow-based classification," *Queen Mary Westf. Coll. Dep. Comput. Sci.*, no. August, 2005.

[81] Weka. Weka Data Mining Software. [Online] Available: https://www.cs.waikato.ac.nz/ml/weka/

[82] U. M. Fayyad and K. B. Irani, "Multi-Interval Discretization of Continuos-Valued Attributes for Classification Learning," Proceedings of the International Joint Conference on Uncertainty in AI. pp. 1022–1027, 1993.

[83] M. S. Hoque, M. A. Mukit, M. A. N. Bikas, and M. Sazzadul Hoque, "An Implementation of Intrusion Detection System Using Genetic Algorithm," Int. J. Netw. Secur. Its Appl., vol. 4, no. 2, pp. 109–120, 2012.

[84] D. S. Kim, H. Nguyen, and J. S. Park, "Genetic Algorithm to Improve SVM Based Network Intrusion Detection System," 19th Int. Conf. Adv. Inf. Netw. Appl. Vol. 1 (AINA Pap., vol. 2, pp. 155–158, 2005.

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

[86] S. Mika, G. Ratsch, J. Weston, B. Schölkopf, and K.-R. Muller, "Fisher discriminant analysis with kernels," Ieee, pp. 41–48, 1999.

[87] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos, "Continuous Monitoring of Distance-Based Outliers over Data Streams," 2011

[88] K. Alsabti, S. Ranka, and V. Singh, "An efficient k-means clustering algorithm," Electical Eng. Comput. Sci., 1997.

[89] R. Pelossof, M. Jones, I. Vovsha, and C. Rudin, "Online coordinate boosting," 2009 IEEE 12th Int. Conf. Comput. Vis. Work. ICCV Work. 2009, pp. 1354–1361, 2009.

[90] MOA. MOA Machine Learning for Streams. [Online] Available: https://moa.cms.waikato.ac.nz/

[91] Y. Fu, X. Zhu, and B. Li, "A survey on instance selection for active learning," Knowl. Inf. Syst., vol. 35, no. 2, pp. 249–283, 2013.

Appendix 1

This appendix provides a description regarding the feature sets used throughout this document. Two different feature sets were extracted according to the considered intrusion dataset.

The fine-grained intrusion dataset (see Section 5.1) extracts the features from [5], which are listed below.

Table 19. Fine-grained intrusion dataset feature set.

#	Description
1	Header-based, ip type
2	Header-based, ip len
3	Header-based, ip id
4	Header-based, ip offset
5	Header-based, ip reserved flag
6	Header-based, ip don't fragment flag
7	Header-based, ip more fragments flag
8	Header-based, ip protocol
9	Header-based, ip checksum
10	Header-based, udp source port
11	Header-based, udp destination port
12	Header-based, udp length
14	Header-based, udp checksum
15	Header-based, icmp type
16	Header-based, icmp code
17	Header-based, icmp checksum
18	Header-based, tcp source port
19	Header-based, tcp destination port
20	Header-based, tcp sequence number
21	Header-based, tcp acknowledgment number
22	Header-based, tcp fin flag
23	Header-based, tcp syn flag
24	Header-based, tcp reset flag
25	Header-based, tcp push flag
26	Header-based, tcp ack flag
27	Header-based, tcp urg flag
28	Header-based, frame length
29	Host-based, number of packets from source to destination
30	Host-based, number of packets from destination to source
31	Host-based, number of bytes sent from source to destination
32	Host-based, number of bytes sent from destination to source
33	Host-based, number of packets with push flag set sent from source to destination
34	Host-based, number of packets with push flag set sent from destination to source
35	Host-based, number of packets with syn and fin flag set sent from source to destination
36	Host-based, number of packets with syn and fin flag set sent from destination to source

37	Host-based, number of packets with fin flag set sent from source to destination
38	Host-based, number of packets with fin flag set sent from destination to source
39	Host-based, number of packets with ack flag set sent from source to destination
40	Host-based, number of packets with ack flag set sent from destination to source
41	Host-based, number of packets with rst flag set sent from source to destination
42	Host-based, number of packets with rst flag set sent from destination to source
43	Host-based, if is the first packet seen in flow-based communication
44	Service-based, number of packets from source to destination
45	Service-based, number of packets from destination to source
46	Service-based, number of bytes from source to destination
47	Service-based, number of bytes from destination to source
48	Service-based, it is the first packet seen in flow-based communication
49	Connection status
50	Class {normal, attack}

The extraction of the feature set for the fine-grained intrusion dataset occurred for each network packet. For each network packet read from the Network Interface Card (NIC), a set of predetermined features was extracted and sent to a classifier engine for classification. All feature values were obtained by analyzing the packet header values. The header-based category of features was extracted directly from the network packet header. The host-based and service-based categories of features were extracted by analyzing the communication history between two hosts or services.

A 2 seconds time window was used to compute the time-based type of features. The feature extractor engine was implemented using the C++ language following the PCAP API using the libpcap (www.tcpdump.org) library; the implementation details are further explained in [5]. From each network packet in the fine-grained intrusion databases, the set of features are extracted, and the feature vector is written in a separate dataset. Each feature vector entry was automatically labeled as normal or attack, on the basis of the source IP address. It is important to note that features that were scenario-specific were not considered, e.g., TTL and IP address source or destination.

In contrast, *MAWIFlow* had its features extracted by the means of *BigFlow* feature extraction module, in which four different feature sets were extracted: Orunada [78], Nigel [79], Moore [80], and Viegas [5]. Those feature sets refer to the evaluation process of the conformal evaluator (see method in Section 4.6) and the evaluation of the adapting to network changes over time (see method in Section 4.7).

Bellow each table provides a description of each feature for each feature set.

Table 20. Orunada feature set.

#	**Description**
1	Percentage of packets seen between host/host communication with TCP SYN flag set.

2	Percentage of packets seen between host/host communication with TCP ACK flag set.
3	Percentage of packets seen between host/host communication with TCP RST flag set.
4	Percentage of packets seen between host/host communication with TCP FIN flag set.
5	Percentage of packets seen between host/host communication with TCP CWR flag set.
6	Percentage of packets seen between host/host communication with TCP URG flag set.
7	Average packet size seen between host/host communication
8	Average TTL values seen between host/host communication
9	Percentage of packets seen between host/host communication with ICMP redirect flag set.
10	Percentage of packets seen between host/host communication with ICMP time exceeded flag set.
11	Percentage of packets seen between host/host communication with ICMP unreachable flag set.
12	Percentage of packets seen between host/host communication with ICMP other types flag set.
14	Number of unique hosts sending network packets to host
15	Number of unique services sending network packets to host

Table 21. Nigel feature set.

#	**Description**
1	Minimum network packet length during time interval sent
2	Mean network packet length during time interval sent
3	Maximum network packet length during time interval sent
4	Standard deviation of network packet length during time interval sent
5	Minimum network packet length during time interval received
6	Mean network packet length during time interval received
7	Maximum network packet length during time interval received
8	Standard deviation of network packet length during time interval received
9	Minimum network packet arrival sent time during time interval
11	Mean network packet arrival sent time during time interval
12	Maximum network packet arrival sent time during time interval
13	Standard deviation of network packet arrival sent time during time interval
14	Minimum network packet arrival received time during time interval
15	Mean network packet arrival received time during time interval
16	Maximum network packet arrival received time during time interval
17	Standard deviation of network packet arrival received time during time interval
18	Protocol (UDP, TCP or ICMP)
19	Number of network packets sent
20	Number of bytes received
21	Number of network packets received

Table 22. Viegas feature set.

#	**Description**
1	Number of Packets
2	Number of Bytes
3	Average Packet Size
4	Percentage of Packets (PSH Flag)
5	Percentage of Packets (SYN and FIN Flags)
6	Percentage of Packets (FIN Flag)
7	Percentage of Packets (SYN Flag)
8	Percentage of Packets (ACK Flag)
9	Percentage of Packets (RST Flag)
10	Percentage of Packets (ICMP Redirect Flag)
12	Percentage of Packets (ICMP Redirect Flag)

13	Percentage of Packets (ICMP Time Exceeded Flag)
14	Percentage of Packets (ICMP Unreachable Flag)
15	Percentage of Packets (ICMP Other Types Flag)
16	Average Packet Size, Throughput in Bytes
17	Protocol
18	Number of Packets – Both directions
19	Number of Bytes – Both directions
20	Average Packet Size – Both directions
21	Percentage of Packets (PSH Flag) – Both directions
22	Percentage of Packets (SYN and FIN Flags) – Both directions
23	Percentage of Packets (FIN Flag) – Both directions
24	Percentage of Packets (SYN Flag) – Both directions
25	Percentage of Packets (ACK Flag) – Both directions
26	Percentage of Packets (RST Flag) – Both directions
27	Percentage of Packets (ICMP Redirect Flag) – Both directions
28	Percentage of Packets (ICMP Redirect Flag) – Both directions
29	Percentage of Packets (ICMP Time Exceeded Flag) – Both directions
30	Percentage of Packets (ICMP Unreachable Flag) – Both directions
31	Percentage of Packets (ICMP Other Types Flag) – Both directions
32	Throughput in Bytes – Source to Destination
33	Number of Packets – Source to Destination
34	Number of Bytes – Source to Destination
35	Average Packet Size – Source to Destination
36	Percentage of Packets (PSH Flag) – Source to Destination
37	Percentage of Packets (SYN and FIN Flags) – Source to Destination
38	Percentage of Packets (FIN Flag) – Source to Destination
39	Percentage of Packets (SYN Flag) – Source to Destination
40	Percentage of Packets (ACK Flag) – Source to Destination
41	Percentage of Packets (RST Flag) – Source to Destination
42	Percentage of Packets (ICMP Redirect Flag) – Source to Destination
43	Percentage of Packets (ICMP Redirect Flag) – Source to Destination
44	Percentage of Packets (ICMP Time Exceeded Flag) – Source to Destination
45	Percentage of Packets (ICMP Unreachable Flag) – Source to Destination
46	Percentage of Packets (ICMP Other Types Flag) – Source to Destination
47	Throughput in Bytes – Destination to Source
48	Number of Packets – Destination to Source
49	Number of Bytes – Destination to Source
50	Average Packet Size – Destination to Source
51	Percentage of Packets (PSH Flag) – Destination to Source
52	Percentage of Packets (SYN and FIN Flags) – Destination to Source
53	Percentage of Packets (FIN Flag) – Destination to Source
54	Percentage of Packets (SYN Flag) – Destination to Source
55	Percentage of Packets (ACK Flag) – Destination to Source
56	Percentage of Packets (RST Flag) – Destination to Source
57	Percentage of Packets (ICMP Redirect Flag) – Destination to Source
58	Percentage of Packets (ICMP Redirect Flag) – Destination to Source
59	Percentage of Packets (ICMP Time Exceeded Flag) – Destination to Source
60	Percentage of Packets (ICMP Unreachable Flag) – Destination to Source
61	Percentage of Packets (ICMP Other Types Flag) – Destination to Source
62	Throughput in Bytes – Destination to Source

Table 23. Moore feature set.

#	Description
1	Minimum inter arrival time
2	First quartile of inter arrival time
3	Median inter arrival time
4	Average inter arrival time
5	Third quartile of inter arrival time
6	Maximum inter arrival time
7	Variance of inter arrival time
8	Minimum inter arrival time from source to destination
9	First quartile of inter arrival time from source to destination
10	Median inter arrival time from source to destination
11	Average inter arrival time from source to destination
12	Third quartile of inter arrival time from source to destination
13	Maximum inter arrival time from source to destination
14	Variance of inter arrival time from source to destination
15	Minimum inter arrival time from destination to source
16	First quartile of inter arrival time from destination to source
17	Median inter arrival time from destination to source
18	Average inter arrival time from destination to source
19	Third quartile of inter arrival time from destination to source
20	Maximum inter arrival time from destination to source
21	Variance of inter arrival time from destination to source
22	Minimum packet length
23	First quartile of packet length
24	Median packet length
25	Average packet length
26	Third quartile of packet length
27	Maximum packet length
28	Variance of packet length
29	Minimum packet length from source to destination
30	First quartile of packet length from source to destination
31	Median packet length from source to destination
32	Average packet length from source to destination
33	Third quartile of packet length from source to destination
34	Maximum packet length from source to destination
35	Variance of packet length from source to destination
36	Minimum packet length from destination to source
37	First quartile of packet length from destination to source
38	Median packet length from destination to source
39	Average packet length from destination to source
40	Third quartile of packet length from destination to source
41	Maximum packet length from destination to source
42	Variance of packet length from destination to source
43	Total network packets from source to destination
44	Total network packets from destination to source
45	Total network packets with TCP ACK flag set from source to destination
46	Total network packets with TCP ACK flag set from destination to source
47	Total network packets with only TCP ACK flag set from source to destination
48	Total network packets with only TCP ACK flag set from destination to source

49	Total network packets with TCP SYN flag set from source to destination
50	Total network packets with TCP SYN flag set from destination to source
51	Total network packets with TCP FIN flag set from source to destination
52	Total network packets with TCP FIN flag set from destination to source
53	Total network packets with TCP PSH flag set from source to destination
54	Total network packets with TCP PSH flag set from destination to source
55	Total network packets with TCP URG flag set from source to destination
56	Total network packets with TCP URG flag set from destination to source
57	Total ICMP packets from source to destination
58	Total ICMP packets from destination to source
59	Throughput from source to destination
60	Throughput from destination to source

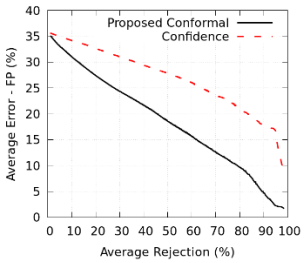
In addition to each extracted feature set, each network flow also contains features regarding their labeling process. Those features are shown in Table 24. It is important to note that such features are removed before the classification process, their only purposes are either for debugging or flow labeling.

Table 24. Features regarding the labeling process.

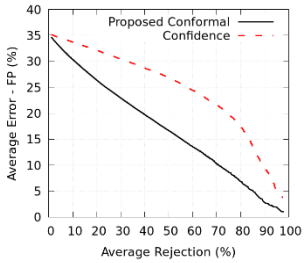
#	**Description**
1	MAWILAB, attack taxonomy
2	MAWILAB, anomaly distance
3	MAWILAB, number of detectors that detect the anomaly
4	MAWILAB, label {normal, anomalous, suspicious, notice}
5	Class {normal, attack}

Appendix 2

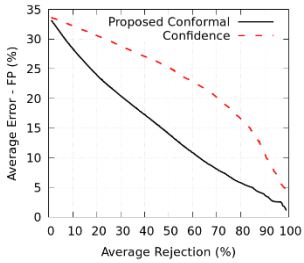
This appendix provides the complete conformal evaluator results compared to CRT throughout 10 years of *MAWIFlow* dataset.



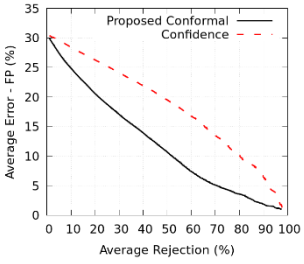
(a) Jan. 2007 to Jun. 2007



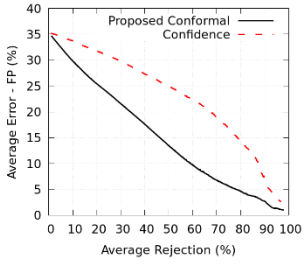
(b) Jul. 2007 to Dec. 2007



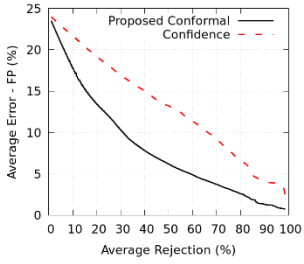
(c) Jan. 2008 to Jun. 2008



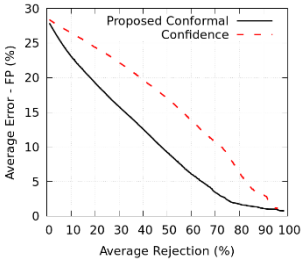
(d) Jul. 2008 to Dec. 2008



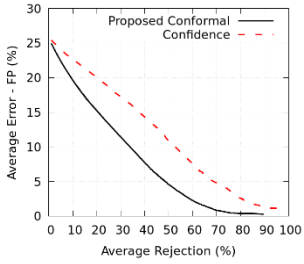
(e) Jan. 2009 to Jun. 2009



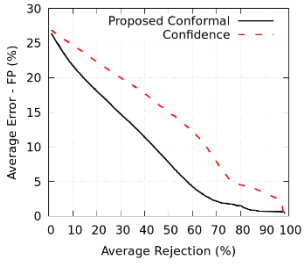
(f) Jul. 2009 to Dec. 2009



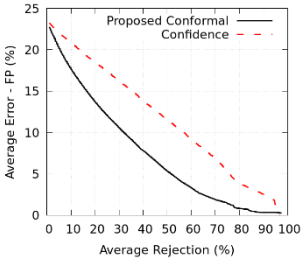
(g) Jan. 2010 to Jun. 2010



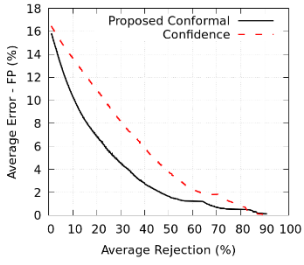
(h) Jul. 2010 to Dec. 2010



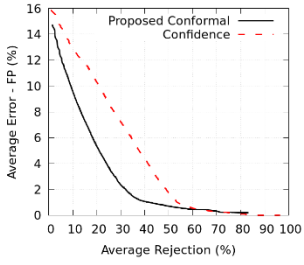
(i) Jan. 2011 to Jun. 2011



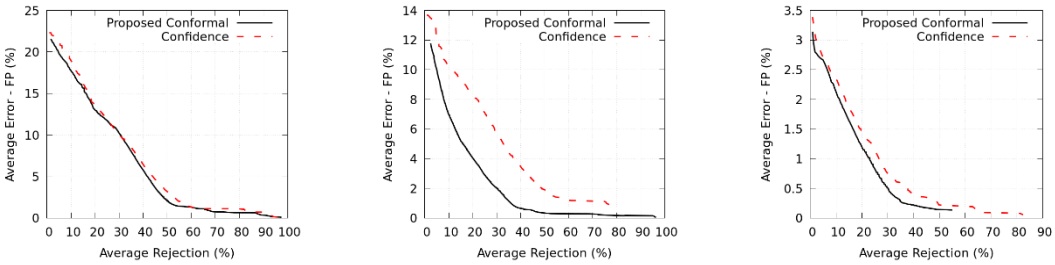
(j) Jul. 2011 to Dec. 2011



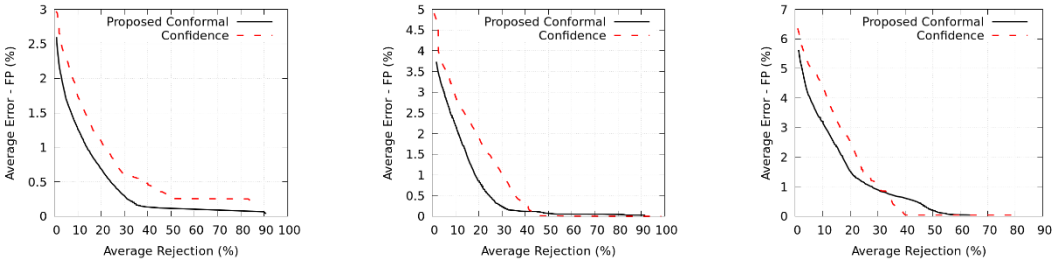
(k) Jan. 2012 to Jun. 2012



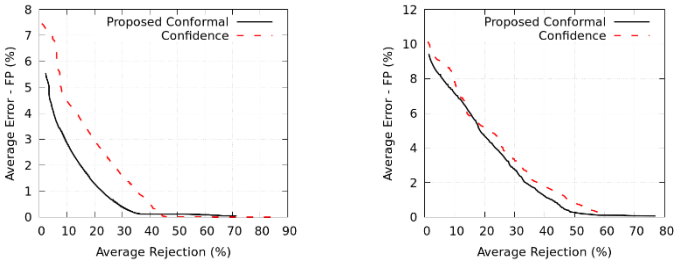
(l) Jul. 2012 to Dec. 2012



(m) Jan. 2013 to Jun. 2013 (n) Jul. 2013 to Dec. 2013 (o) Jan. 2014 to Jun. 2014



(p) Jul. 2014 to Dec. 2014 (q) Jan. 2015 to Jun. 2015 (r) Jul. 2015 to Dec. 2015

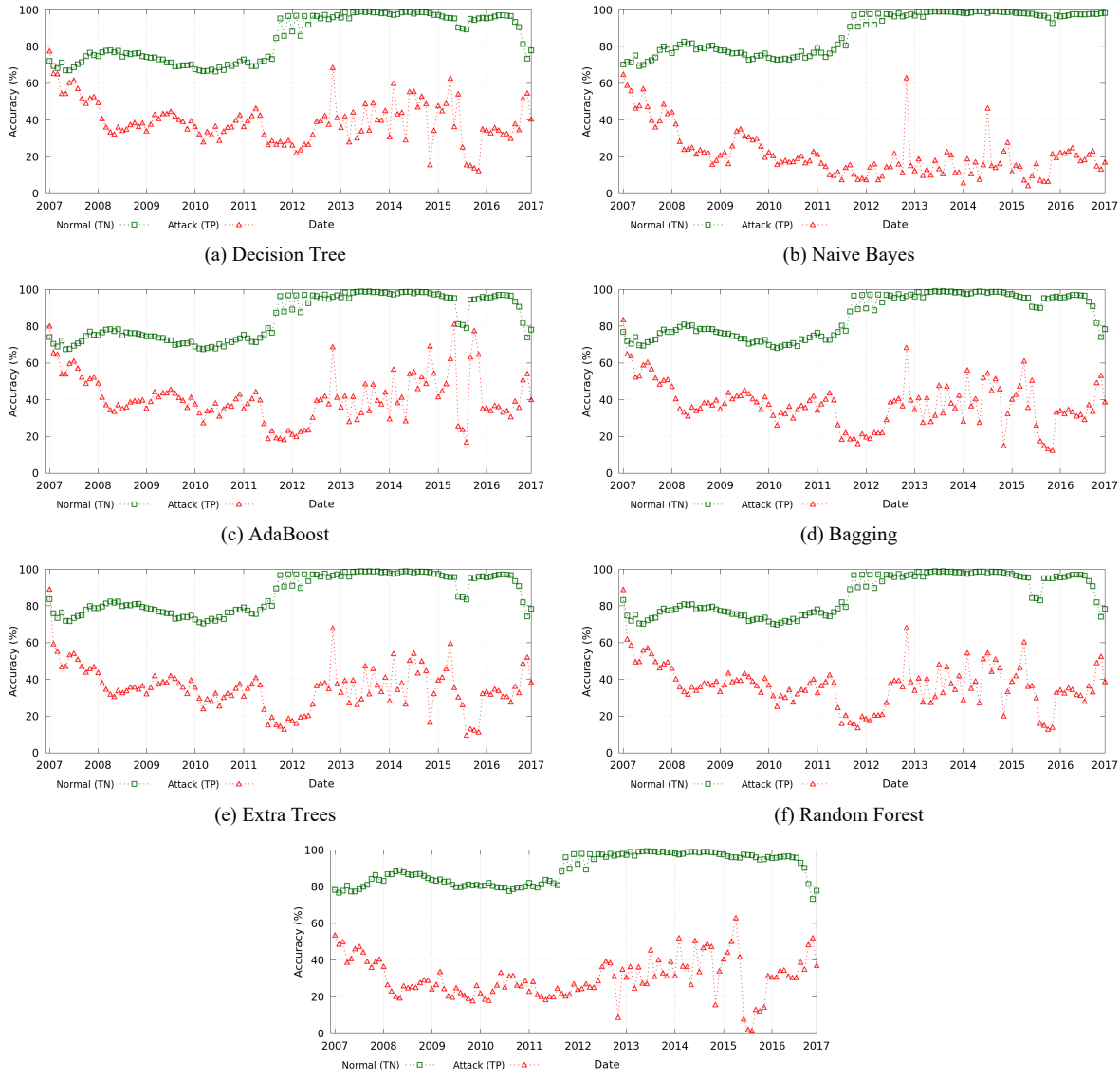


(s) Jan. 2016 to Jun. 2016 (t) Jul. 2016 to Dec. 2016

Figure 42. Conformal evaluator performance throughout 10 years of *MAWIFlow* dataset.

Appendix 3

This appendix shows the performance of several classifiers in *MAWIFlow* dataset according to their used feature set. Only for this appendix, classifiers were trained using the data from Jan. 2007, and not updated afterwards.



(a) Decision Tree

(b) Naive Bayes

(c) AdaBoost

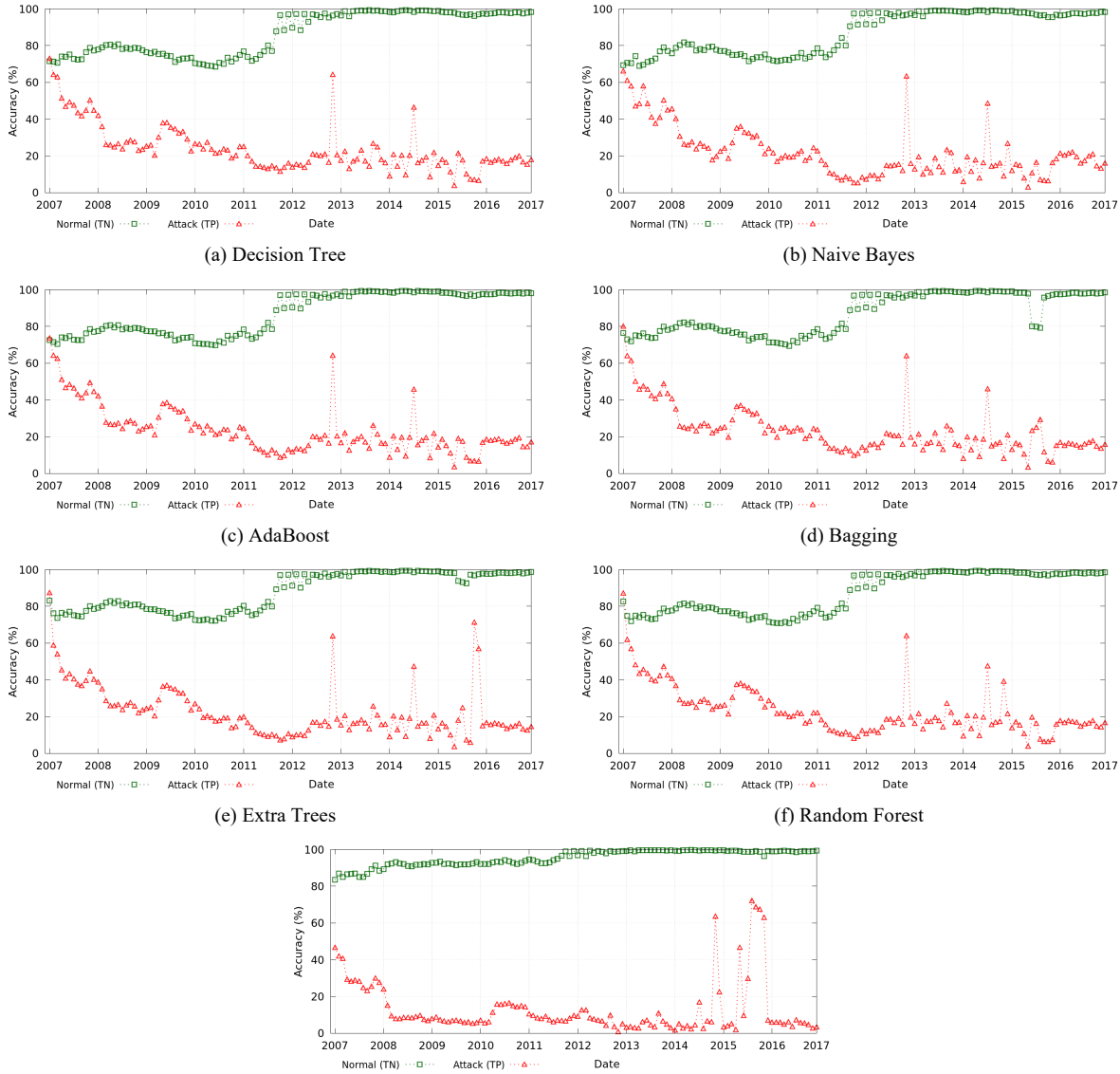
(d) Bagging

(e) Extra Trees

(f) Random Forest

(g) Hoeffding Tree (Weka API)

Figure 43. Performance of several classifiers using MOORE feature set throughout 10 years of *MAWIFlow* dataset.



(a) Decision Tree

(b) Naive Bayes

(c) AdaBoost

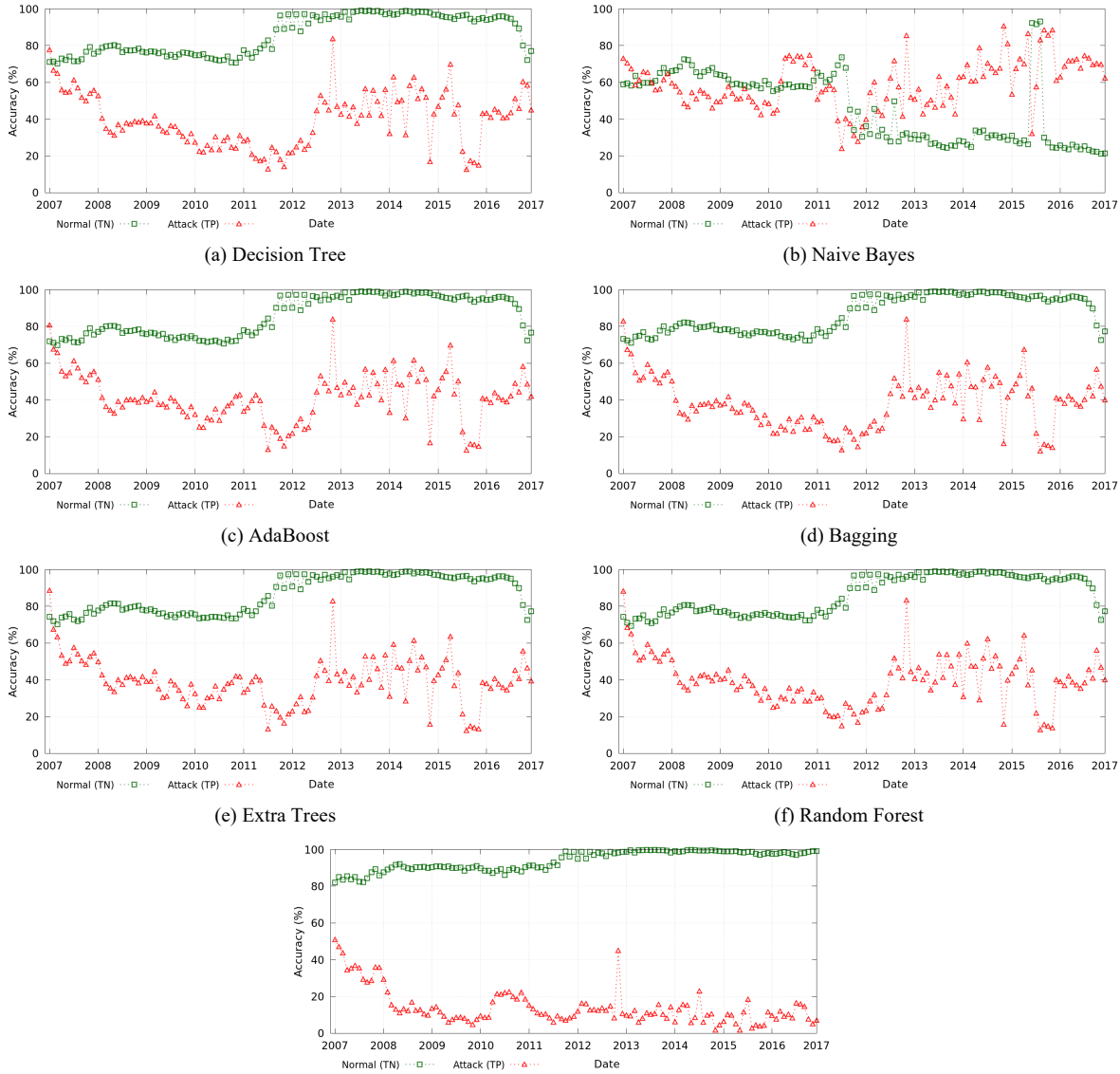
(d) Bagging

(e) Extra Trees

(f) Random Forest

(g) Hoeffding Tree (Weka API)

Figure 44. Performance of several classifiers using NIGEL feature set throughout 10 years of *MAWIFlow* dataset.



(a) Decision Tree

(b) Naive Bayes

(c) AdaBoost

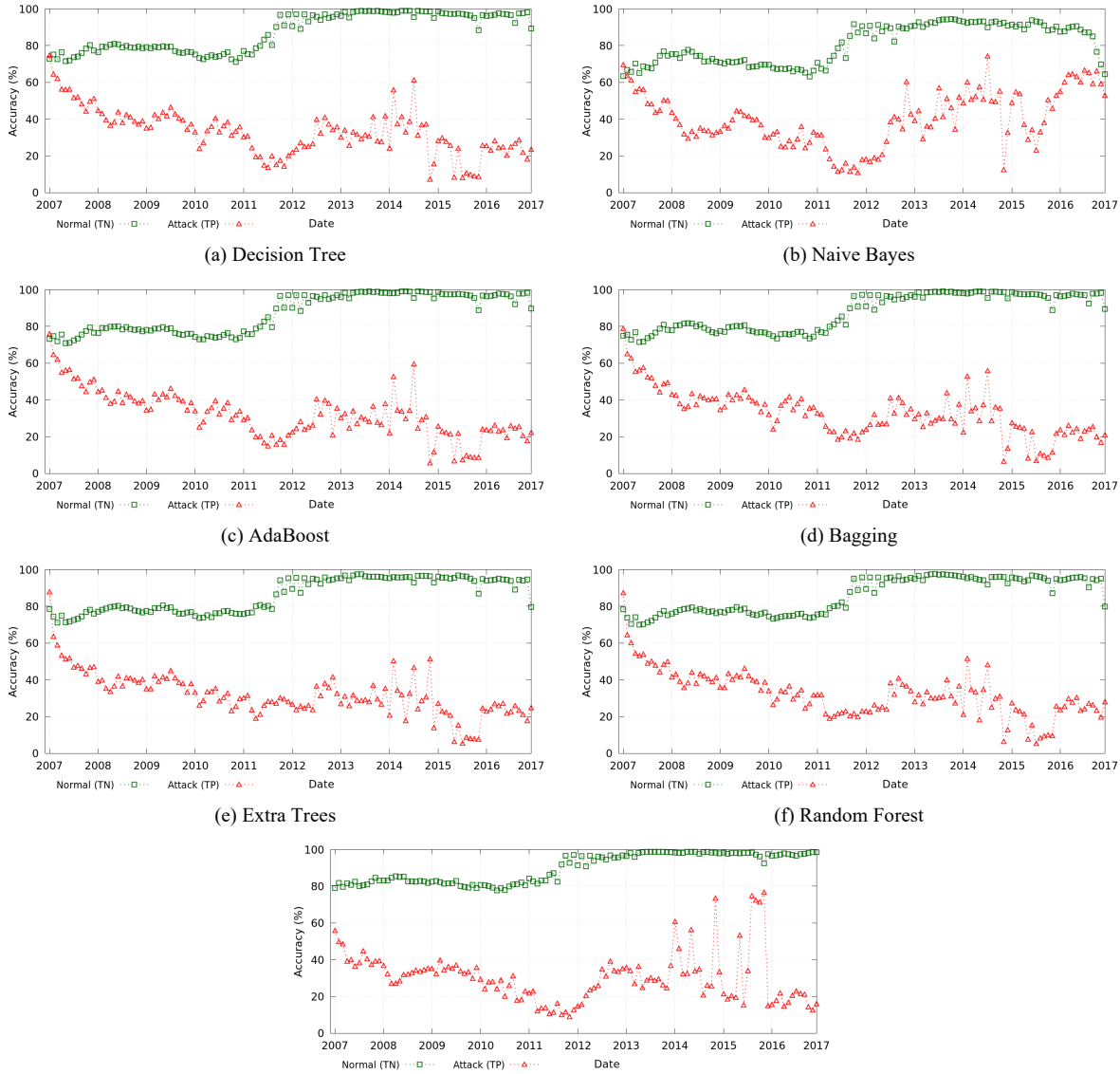
(d) Bagging

(e) Extra Trees

(f) Random Forest

(g) Hoeffding Tree (Weka API)

Figure 45. Performance of several classifiers using VIEGAS feature set throughout 10 years of *MAWIFlow* dataset.



(a) Decision Tree

(b) Naive Bayes

(c) AdaBoost

(d) Bagging

(e) Extra Trees

(f) Random Forest

(g) Hoeffding Tree (Weka API)

Figure 46. Performance of several classifiers using ORUNADA feature set throughout 10 years of *MAWIFlow* dataset.