Learning From Network Data Changes for Unsupervised Botnet Detection

Bruno Henrique Schwengber, Andressa Vergütz, *Student Member, IEEE*, Nelson G. Prates, Jr., and Michele Nogueira

Abstract—**The networks of infected devices (a.k.a., botnets) threaten network security due to their dynamic nature and support to different attacks (e.g., Distributed Denial of Services and personal data theft). Detecting botnets is a challenging task because the infected devices (bots) are numerous, widely and geographically spread. Significant attention has been given to improve the efficiency, robustness and adaptability of network security approaches. However, in the literature, botnet detection techniques usually ignore fast changes in statistical data distribution, performing over static windows, i.e., fixed intervals of time or fixed quantity of flows. Changes in statistical data distribution are known as concept drifts and they make the classification models obsolete. Furthermore, those works employing approaches aware of concept drift use supervised machine learning, which is slow, costly, and prone to error. Therefore, this article presents TRUSTED, a system for online and unsupervised botnet detection aware of concept drifts. Unlike other works, the TRUSTED system improves the learning process for botnet detection, applying concept drift in an online and unsupervised classification. Evaluations comprise offline and online scenarios. Results show that the TRUSTED system detects botnets using concept drift identification, reaching 87% to 95% accuracy, precision, recall, and F1-scores.**

Index Terms—**Security management, concept drift, botnet detection, attacks.**

I. Introduction

BOTNETS comprise a set of network devices infected with malicious software (a.k.a., malware). These infected devices (bots) receive remote commands from a botmaster to carry out attacks, such as Distributed Denial of Service (DDoS), massive spamming, or personal data theft [1]. DDoS attacks overload the computational and network resources from the victim, causing financial and reputation losses. The rise of the Internet of Things (IoT) has brought several vulnerabilities (e.g., faults in software design and authentication failures) [2] to the networks, intensifying the potential of botnets. Data volume and data transmission rate in IoT have increased [3], requiring rapid adaptation and security management redesign.

Internet traffic carries the form of high-speed streams. Hence, it is necessary to process and analyze it in an online and adaptive way to prevent the spread of attacks [4]. Online processing means an ongoing input of instances into a system, the system processes them as they arrive. However, processing, analyzing, and adapting become a challenge for online learning, given the increasing volatility in network statistical properties. Traditional machine learning (ML) methods applied to network security management (train and test) assume, in general, a static behavior for network traffic, i.e., they assume that test data will always follow the same distribution as the data employed for training. These techniques are restrictive when dealing with dynamic and adversarial environments (e.g., network traffic) [5]. Nevertheless, high-speed streams evolve and change data statistical distributions. These changes are called concept drifts [6]. When a concept drift happens, the classification model becomes obsolete due to statistical distribution changes. This affects negatively the classification model performance [7].

The high level of adaptability expected by network security management reveals limitations in traditional ML methods (e.g., offline learning) given the high traffic velocity, volume and variety [3]. Existing works related to botnet detection ignore the volatility and rapid changes in network traffic behavior [4], [8], [9], [10]. Concept drift has been applied to network security [5]. Casas *et al.* have employed the concept drift detection in an online and supervised environment for malicious traffic detection, highlighting the improvement in performance by adapting the model to new data traffic distributions [5]. Nevertheless, the use of supervised classifiers is in general slow, costly, and prone to error for the malicious traffic detection given the high speed and data volume [3], [11]. Thus, since concept drift deteriorate the classification performance [5] and the supervised approach presents limitations [3], an unsupervised online detection system aware of concept drifts is essential to identify changes in data distribution and detect botnets without supervision.

This article introduces **TRUSTED**, an online sys**T**em fo**R** **U**nsupervi**S**ed bo**T**n**E**t **D**etection. The TRUSTED system applies the unsupervised identification of concept drifts over a sliding window to improve the input for the unsupervised

Manuscript received January 7, 2021; revised May 23, 2021; accepted July 27, 2021. Date of publication August 31, 2021; date of current version March 11, 2022. This work is supported by the FAPESP/MCTIC MENTORED project under the grants #20/05884-8 and #18/23098-0, by CAPES/Brazil under grants #88882.381944/2019-01 and #88882.461738/2019-01, and by CNPq/Brazil under the grants #309129/2017-6 and #432204/2018-0. The associate editor coordinating the review of this article and approving it for publication was C. Kamhoua. *(Corresponding author: Michele Nogueira.)*

Bruno Henrique Schwengber, Andressa Vergütz, and Nelson G. Prates, Jr., are with the Computer Science Department, Federal University of Paraná, Curitiba 81530160, Brazil (e-mail: bhschwengber@inf.ufpr.br; avergutz@inf.ufpr.br; ngpjunior@inf.ufpr.br).

Michele Nogueira is with the Computer Science Department, Federal University of Paraná, Curitiba 81530160, Brazil, and also with the Computer Science Department, Federal University of Minas Gerais, Belo Horizonte 31270-901, Brazil (e-mail: michele@dcc.ufmg.br).

Digital Object Identifier 10.1109/TNSM.2021.3109076

online botnet detection. Thus, by identifying concept drifts and analyzing data similarity, the system provides unsupervised learning and identifies infected devices in the network. Being unsupervised implies that the system does not need prior knowledge of the network flow, attack signatures, or prior training of the classification models to detect botnets. The major contributions of this article are as follows.

- TRUSTED, a three-phased online system for unsupervised botnet detection aware of concept drifts. It is protocol independent and resilient against zero-day attacks.
- D3N, a non-linear discriminative drift detector adapted from D3 [6], changing the discriminative classifier to detect non-linear changes in data.

The TRUSTED system evaluation follows a trace-driven approach under two datasets: the Botnet 2014 [12] and the CTU-13 dataset [13]. These datasets contain network traffic collected from periods with and without attacks. The datasets include network traffic from Peer-to-Peer (P2P), Internet Relay Chat (IRC), and HyperText Transfer Protocol (HTTP). They encompass different botnets, such as *Rbot, Virut and Zeus*. From these datasets, network flow features are extracted, composing the instances of analysis. In this work, an instance means a set of values from related features extracted from the network traffic flow. Thus, the extracted features build a data stream to emulate an online environment for evaluation. Having as input a data stream, the TRUSTED system delimits the data window, identifies concept drifts, and detects botnets. The results demonstrate that the system reaches 87% - 95% accuracy, detecting botnet using concept drift identification to provide an optimized input for online learning. When compared to systems following a traditional fixed window strategy, the TRUSTED system improves the results in 1% - 5% in accuracy, precision, recall, and F1-score.

This article proceeds as follows. Section II introduces the main challenges related to network security management. Section III discuss the related works. Section IV introduces concept drift in online learning and details the TRUSTED system. Section V presents the performance evaluation methodology and Section VI discusses the results and the system limitations. Section VII concludes the article and discusses future works.

II. Challenges on Network Security

This section enlightens network security main issues related to data acquisition, operational costs, attack detection, device heterogeneity and concept drifts, aspects that can influence the performance of security solutions.

In general, network security solutions capture packets in a stream because data extraction techniques employ their data. However, packet payload acquisition presents two main problems: privacy and encryption. Inspecting the packet payload violates data user privacy since it accesses private information. Several applications employ payload data encryption to hamper disclosure [14]. The network information employed to classify the traffic must gather data from the packet header to overcome these limitations. Using data from the packet header is possible to build a large set of statistical properties from the network traffic [12].

Another challenge related to network security solutions involves operational costs [4], [5]. The operational cost increases with the large scale of the network since it is necessary to process and store many heterogeneous network data. Hence, in online learning using sliding or fixed windows, keeping just a portion of old data is possible to prioritize the newest data. The sliding widowing follows the rule first-in/first-out, adding and dropping the instances as the new arrives. The fixed windowing gathers a fixed number of instances to process and then drop all of them to gather new one [15].

Network security solutions rely on attacks and traffic signatures or anomaly detection [14], [16]. However, many of these solutions depend on a labeled dataset to train models, resulting in three main problems. First, modeling requires large storage to keep several instances from malicious and benign traffic. Second, it requires an expert to define and label the traffic as malicious or benign. Third, supervised machine learning can not accurately detect botnet attacks because new botnets launch several zero-day attacks every day [3]. Network security solutions must employ online unsupervised algorithms to address these limitations since these algorithms do not require labeled training data or previous knowledge about attack signatures to perform traffic classification. Also, they follow a windowing strategy to avoid large data storage.

The statistical distribution of real-world data stream (e.g., network traffic) might evolve and change over time. These changes are known as concept drift [6]. When a concept drift happens, the classification model becomes outdated because data distribution has changed. Then, it can not correctly and efficiently detect botnets anymore [7]. Supervised and unsupervised approaches exist to identify concept drift. Although the supervised approaches present only a few false-positive drifts, they need labels. However, unsupervised concept drift identifiers work bypassing the need for labels once it uses the statistical data properties to identify drifts [17].

III. Related Works

Different works address and discuss the consequences of concept drift in data stream classification [6], [7], [17], and network attack detection [4], [5], [18]. Concept drifts affect the learning model, usually decreasing the classification performance once data behavior (i.e., statistical properties) changes and makes the classification model obsolete. This section presents the main works related to botnet detection and concept drift identification.

A. Botnet Detection Approaches

Several works address botnet detection [4], [5], [8], [9], [10], [18], [19], [20]. These works generally use ML methods (supervised or unsupervised) and follow online or offline approaches. Supervised algorithms employ the traditional approach for classification models (train, then test). However, the supervised methods require a labeled dataset to train [4], [18], presenting challenges such as storage and human intervention to label data, i.e., making them costly and slowly [3]. Unsupervised ML methods classify data without training or prior knowledge. These methods follow offline and online approaches. Some works apply offline unsupervised approaches to detect botnets through network traffic capture and cluster analysis [8], [19]. Other works employ online unsupervised approaches to detect botnets by timely analyzing network traffic. Online approaches need to be faster than offline due to timely data analysis, improving the decision making process [9], [10], [20]. Moreover, some works follow hybrid approaches using unsupervised and supervised learning to detect botnets [21], [22].

Mainly, the botnet detection approaches in the literature uses flow-based features (e.g., flow duration and amount of traffic bytes) [5], [9], [10], [20], [21], [23], [24], [25]. In [25], the authors highlight privacy issues related to the use of packet payload, encouraging the use of only the header. In [26], the authors employed privileged information as features in a supervised approach. Moreover, Beigi *et al.* [12] and Haddadi and Zincir-Heywood [27] have evaluated different features and protocol for botnet detection. Beigi *et al.* presented a set of best features, including the best amount of traffic and timestamp-based features (e.g., the moment of flow start and end). This work evaluated 16 different traffic features and selected a set of four best features based on flow characteristics. Haddadi and Zincir-Heywood explored five different traffic flow exporters, each with a different set of flow features, using two different protocol as filters, Hypertext Transfer Protocol (HTTP) and Domain Name System (DNS), and five different classifiers. They concluded that a flow exporter and a protocol filter indeed affects the performance of botnet traffic classification.

Other works related to botnet detection can be classified as pursuing detection in early or advanced stages and classified based on their dependence or independence on encryption [10]. There is another classification related to their perception or not of concept drifts. Among the related works, only in [5], the authors employed the concept drift perception, achieving 99% of accuracy. Such a result emphasizes the relevance and performance improvement of classifiers when using concept drift identification techniques. The perception of concept drifts updates obsolete classification models by identifying the changes in the statistical data distribution [6]. However, the use of concept drift perception is new, and it has been applied only in a supervised environment. The online unsupervised botnet detection approaches have a lack of concept drift perception [9], [10].

For instance, Yu *et al.* [9] presented an online and unsupervised detection technique for centralized botnets. The authors performed a traffic transformation into a stream of instances with network features. These instances represent the stream features. They processed the stream by keeping a data window, dropping old instances and replacing them with new ones. The window serves as input to K-Means clustering algorithm. The proposed technique analyzes the similarity between the instances in the same cluster to detect bots. When a cluster exceeds a certain similarity threshold, i.e., when it has similar instances, the group is labeled as suspect. However, this technique does not consider concept drifts related to data distribution, which can negatively affect its performance.

Yahyazadeh and Abadi [10] presented an online unsupervised botnet detection system supported by a reputation model and similarity analysis. The system tracks the history of coordinated activity between devices on the network and calculates each device reputation based on their participation in network tasks between pairs of devices (e.g., sending files and messages). This system uses incremental clustering and similarity analysis to label traffic. It uses fixed-size windows based on time and does not consider concept drifts in the predefined window. Concept drifts affect the system performance negatively because of the changes in data statistical distribution makes the detection model obsolete.

Nevertheless, more recently, Khanchi *et al.* [21] and Siloa and Soniva [20] proposed data stream based approaches. The former proposed the use of genetic programming considering a label budget, e.g., request only a half part of labels to keep the evolutionary model updated. It considered an instance-based sliding window to store only recent data following a first-in-first-out rule. However, it still needs some labels to keep the evolutionary model in the loop. The latter used micro and macro clusters with fading weights to detect botnets. The microclusters are formed by flows in regular time intervals considering a fixed radius nearest neighbor algorithm. In parallel to micro clustering, the work contemplates macro clusters assembled by analyzing the flow density between two near microclusters. The weights emphasize new data as time goes on. This is the closest work to achieve an unsupervised concept drift aware approach in the literature.

B. Concept Drift Identification Approaches

In online learning, there are two types of concept drifts, real and virtual. The real concept drift refers to the change in the classification model performance and happens when the data distribution $p(X)$ affects the classification result $p(y, X)$. In contrast, the virtual concept drift consists of changes only in data $p(X)$, abstaining from the classification result $p(y, X)$ [7]. Moreover, real or virtual concept drift uses supervised and unsupervised methods, respectively [17]. Thus, it is necessary to use an incremental classifier to continually adapt the classification model for the classification and concept drift identification [6].

The concept drift identification follows supervised or unsupervised approaches. The supervised approaches rely on labeled data to identify data changes by monitoring performance metrics (e.g., accuracy). In contrast, the unsupervised approaches rely on data statistical properties [17] (e.g., network data stream) to identify data concept drifts. In the literature, the main supervised concept drift identification method is the Adaptive Windowing (ADWIN) [15]. This method monitors the classification model performance based on a history of previous classification results. The history comprises of two sets of data (also called blocks) *W0* and *W1*, containing the old and recent/new classification results, respectively. After each classification, the ADWIN method compares both data sets using Hoeffding Bound to measure if they are large and distinct enough. While there is no drift, the window keeps growing. If a drift happens, the window shrinks, keeping only the recent (new) data.

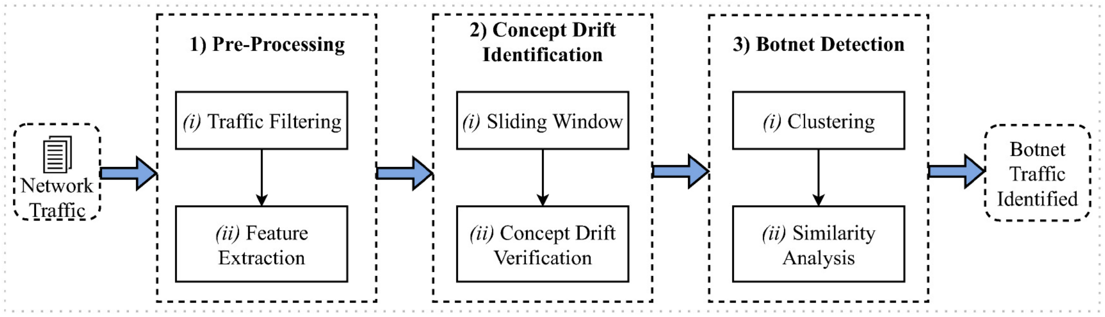


Fig. 1. The TRUSTED System Architecture.

Unsupervised concept drift approaches follow two strategies (i) novelty detection, and (ii) multivariate distribution monitoring [17]. The novelty detection relies on distance and density of information to identify data distribution patterns. It identifies suspicious samples and defines it as an additional class, i.e., a new pattern in data distribution [28], [29], [30]. Although the detection strategy has resulted in a successful identification of concept drifts, it suffers from the curse of dimensionality because it depends on distance measurement [17]. The multivariate distribution monitoring strategy observes data distribution from the perspective of statistical properties, such as mean and standard deviation. In general, this strategy organizes data in two sets (blocks) called old and new data. Then, it compares both sets of data analyzing their statistical data properties [6], [31], [32], [33], [34]. Although multivariate distribution monitoring is prone to false alarms, it presents a fast answer to the system. Thus, it works better than detection approaches when dealing with multi-dimensional streams (e.g., network flows) [17]. Furthermore, in multivariate distribution detection, the Discriminative Drift Detector (D3) [6] works building a sliding window divided into two sets (old and new data). This method identifies drift labeling old and new data with 0 or 1, respectively, and submitting data to a train and test classification. If the classification gets an Area Under the Curve (AUC) score higher than 70%, it means that data is a drift. Otherwise, it means that there is no drift in data properties. This article applies this method to build a sliding window and identify drifts.

IV. The TRUSTED System

This section describes TRUSTED, an online sys**T**em fo**R** **U**nsuperviSed boTnEt Detection. Its goal is to detect botnets online without supervision. The TRUSTED system is protocol-agnostic and resilient against zero-day attacks since it uses unsupervised learning and detects new patterns as they appear. The following subsection details the proposed system operation. Differently from our previous work [35], this work applies the non-linear discriminative drift detector (D3N) in a full unsupervised environment.

A. The TRUSTED Specification

The TRUSTED system comprises of three phases: (*i*) pre-processing, (*ii*) concept drift identification, and (*iii*) botnet detection, illustrated in Fig. 1. The system captures network traffic and requires the network interface in a promiscuous mode on the system hardware. The first phase comprises the data pre-processing, i.e., transforming data into flows and extracting features. The second phase builds the sliding window and identifies possible concept drift in data behavior. The third phase detects malicious behavior by clustering data stream and using similarity analysis.

1) Pre-Processing: is responsible for collecting network traffic, filtering it by flows, and extracting the statistical features to organize it in flow instances. First, the TRUSTED system splits the collected traffic into flows $F = \{p_1, p_2, p_3, \ldots, p_t\}$ following the tuple $\langle$Source IP, Destination IP, Source Port, Destination Port and Communication Protocol$\rangle$. In a flow *F*, *p* is a packet and *t* is an index indicating the packet capture sequential order. From *F*, the TRUSTED system extracts the statistical network features (e.g., the amount of sent packets, traffic volume, and packet size), defined as an instance $X^F = \{x_1, x_2, x_3, \ldots, x_i\}$, where *x* is an extracted feature and *i* is an index associated to each feature. Thus, the next phase creates the data window $W = \{X_1^F, X_2^F, X_3^F, \ldots, X_z^F\}$ that contains the flow instances.

2) Concept Drift Identification: This phase receives as input a set of flow instances from the pre-processing phase. It first builds the sliding window and then identifies concept drifts. The sliding window comprises data of type X^F organized following the first-in/first-out order. Fig. 2 illustrates a sliding window *W* of size 8 (just as example). The sliding window contains old instances (in blue) and new instances (in green). Considering the first-in/first-out policy, new instances mean those recently added to *W*.

The full window has a variable size *K* defined as $K = \delta * (1 + \sigma)$, containing δ old instances and a percentage σ of new instances of network data. While the window is not full, it keeps adding new instances X^F to fill it. When the window gets full, the TRUSTED system runs the concept drift identification. If a concept drift happens, the δ old instances are all removed, keeping only the new ones (the most recent ones) in *W*. Otherwise, if no concept drift happens the system removes only a predefined small amount of old instances. The δ old instances in *W* are labeled by “0”, whereas the new instances are labeled by “1”.

The concept drift identification uses a non-linear discriminative drift detector (D3N) proposed in a previous work [35] and adapted from the (D3) method [6]. The motivation for using this drift detector lies in the fact that network traffic

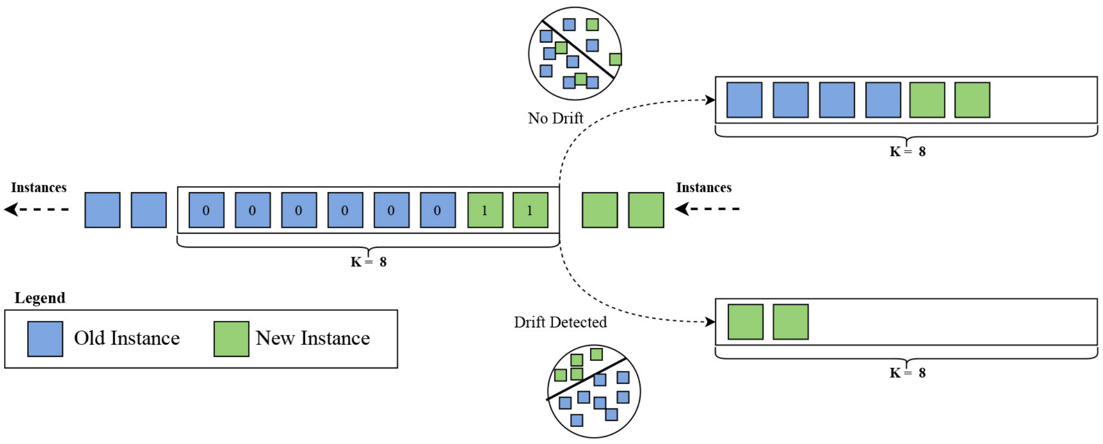


Fig. 2. Concept Drift Detection Workflow. Adapted from [6].

Algorithm 1 D3N - Non-linear Discriminative Concept Drift Identification

```
Input: $stream, \delta, \sigma, \beta$
Output: Drift or not
 1: Initialize window $W$ where $|W| = \delta * (1 + \sigma)$
 2: Decision tree Classifier C
 3: for $X$, $y$ in stream do
 4:     if $W$ is not full then
 5:         $W \leftarrow W \cup X$ {add X to the head of W}
 6:     else
 7:         $S$ is vector of $s$, $|W| = |S|$ {s is a slack variable}
 8:         $s = 0$ for $W[1, \delta]$ {label for old (0) and new (1)}
 9:         $s = 1$ for $W[\delta + 1, end]$
10:         Train C($W, S$) { train C with S as labels of W}
11:         if AUC(C, S) $\geq \beta$ then
12:             Drop $\delta$ elements from the tail of the $W$
13:             return True
14:         else
15:             Drop $\delta\sigma$ elements from the tail of $W$
16:             return False
17:         end if
18:     end if
19: end for
```

statistical distributions change inversely related to the features (e.g., more bytes and fewer packets per flow). Moreover, the addressed scenario needs an unsupervised and multivariate distribution drift detector given its multidimensional stream. D3N is an unsupervised and multivariate distribution detector.

Algorithm 1 details the abstract steps of the D3N method. It receives as inputs a data stream of type X^F, the number of old (δ) instances and a value in the range between 0 and 1 meaning the percentage allowed of new instances (σ). It also receives a threshold (β), a value between 0 and 1. As a result, the algorithm returns true or false (i.e., drift or not), pointing out the concept drift occurrence or not. In order to identify concept drift, the data stream of type X^F arrives sequentially for verification. In line 1, the algorithm initializes data window with size $K = \delta(1 + \sigma)$. Note that it only performs the concept drift identification when the data window is full. Hence, in line 4, the algorithm verifies the data window size. If it is not full, it adds new instances to the data window (as displayed in Fig. 2). If the window is full, the system applies labels to the instances in the window (label 0 for old instance and label 1 for new instance). Formally, in lines 8 and 9, the algorithm attributes label 0 to δ instances and the label 1 to the remaining instances in the window, composing the slack variable *S* (vector of labeled instances *s*). In line 10, the system trains and tests a decision tree classifier with the labeled instances in *S*. In line 11, the function *AUC*(*C*, *S*) calculates a score that indicates the performance of the classifier *C* the classification of the instances in *S*. If the score is higher than β (meaning the performance of the classifier was higher than the predefined threshold β), it notifies a drift occurrence. Otherwise, it returns false for drift occurrence. Note that the training is related to the drift, and not to the botnet detection.

As the TRUSTED system works with streaming data, network data comes continuously. The system identifies a concept drift when the decision tree classifier separates the instances with an AUC performance higher than β threshold. TRUSTED uses AUC as a measure of separability since it represents the percentage degree of the model on distinguishing two classes. Supposing the discriminative classifier model can correctly classify both old and new data as two distinct classes (with 70-100% of AUC), it means that drift occurs. In line 11 of the algorithm, the system compares the AUC result from the decision tree classifier with the β threshold. If the AUC is higher than β it points out that a new data distribution exists, i.e., there is a concept drift occurrence. Otherwise, the algorithm points out no drift occurrence. Hence, when the system detects a concept drift, it removes a set of old δ instances from the window tail, updating the window for new changes (lines 12 and 13 from the algorithm). Moreover, if there is no concept drift occurrence, the TRUSTED system replaces a set of $K * \sigma$ instances for new ones, keeping the data window always updated (lines 15 and 16 from the algorithm).

3) Botnet Detection: This phase receives as input the data window created in the concept drift identification phase. Particularly, the input is a data window W containing the flow instances X^F, ranging from 1 instance to $\delta(1 + \sigma)$ instances. The system groups the flow instances incrementally, i.e., one by one, as long as the stream exists, by the clustering algorithm. Then, it performs the similarity analysis in each cluster (analyzing the flow instances) to detect botnets.

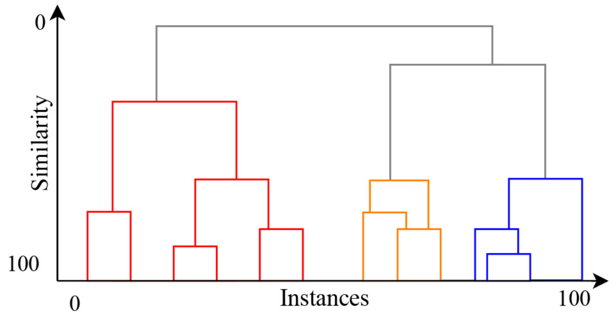


Fig. 3. Hierarchical Tree (dendrogram).

Algorithm 2 HAC: Hierarchical Agglomerative Clustering

```
Input: stream, $\gamma$
Output: Hierarchical Tree T
 1: A ← $\oslash$ {Initialize the group of cluster as empty}
 2: for all $(X, y)$ in stream do
 3:     A ← A ∪ stream {Add each instance to the group}
 4: end for
 5: T ← A {Add the group of clusters to the tree}
 6: while | A | > 1 do
 7:     $G_1^*$, $G_2^*$ ← min DIST($G_1$, $G_2$) {Clusters the closest pairs}
 8:     A ← (A \ { $G_1$ }) \ {$G_2$} {Remove the old clusters}
 9:     A ← A ∪ {$G_1^*$ ∪ $G_2^*$} {Add the new clusters to the group}
10:     T ← T ∪ $A$} {Add the new group of clusters to the tree}
11: end while
```

The system uses the Hierarchical Agglomerative Clustering (HAC) to cluster the data window *W*, i.e., to cluster the flow instances X^F. In each cluster, the algorithm groups the flow instances using Euclidean distance γ that computes the distance between the features values of each X^F. Hence, smaller the distance γ between the flow instances, more clusters the algorithm creates. According to [36], distance metrics assist in achieving better results since the relationships of the observed data (e.g., clusters) are better represented following usual distance metrics, such as the Euclidean distance. While similar data requires small distance, sparse data needs large distance to be well represented in the model. Thus, similarity and distance depend on each other and their values are set according to the data context and environment. For instance, according to [9], [10], the botnet flow traffic is highly similar with each other since the cluster instances even overlap; hence, botnet flows can be detected by setting a high similarity threshold and small distance threshold in the clusters. This information from [9], [10] guided us to set a high similarity threshold of 95% due to our context. Finally, by using distance metric, the construction of the learning models becomes easier and the accuracy usually improves [36]. Moreover, HAC uses a tree data structure to manage the clusters [37]. For this, it initializes each flow instance as clusters. Then, for each iteration, the algorithm groups the two closest clusters building a hierarchical tree (dendrogram), as shown Fig. 3. Algorithm 2 details HAC. In line 1, it initializes an empty tree. In line 3, the algorithm defines *A* as a group of clusters, where each flow instance is a cluster, and, in line 5, it adds the clusters to the hierarchical tree *T* (to manage the clusters easily in a data structure). As long as the number of clusters is greater than 1, in line 7, the algorithm groups the closest pairs (G_1, G_2) maintaining the maximum Euclidean distance γ and adding to the new groups (G_1^*, G_2^*). In lines 8 and 9, the algorithm removes the old clusters from *A* and adds the new groups to the clusters group *A*. In line 10, the algorithm adds the group of clusters *A* to the hierarchical tree *T*.

From the clusters created by Algorithm 2, the system performs the similarity analysis. It calculates the Euclidean distance between all flow instances of the same cluster following Eq. (1). Specifically, the equation evaluates a cluster *S* calculating the Euclidean distance *d*(*x*, *y*) between two instances (*x* and *y*) in the same cluster *S*. To identify the bot clusters, i.e., the botnets, the TRUSTED system compares the Euclidean distance result $\Delta(S)$ (equation output) with a threshold χ defined as 95% of similarity. This threshold determines the critical distance between the flow instances to detect botnets. A botnet cluster has its flows 5% of γ distance from each other, presenting high similarity. Hence, if the distance value $\Delta(S)$ is smaller than the threshold χ, it defines the cluster as infected because, according to [9], [10], a infected cluster has high instances similarity, i.e., $\Delta(S) < \chi$. Botnets have high similarity due to the bot performs similar activities. Therefore, the Euclidean distance between flow instances of the cluster determines when there is a botnet in the network.

$$\Delta(S) = \frac{1}{|S| * (|S| - 1)} * \sum\{d(x, y)\}. \tag{1}$$

V. Performance Evaluation

The TRUSTED system performance evaluation followed a trace-driven approach in online and offline environments. The TRUSTED system implementation is available to public.[1] The datasets employed for performance evaluation involve the Botnet 2014[2] (**dataset 1**) from University of New Brunswick [12], and CTU-13[3] (**dataset 2**) from Stratosphere Lab [13]. Evaluations follow these datasets for both environment settings since they contain real botnet attacks traffic. Moreover, the dataset reduces complexity and enables reproducibility. In the environment settings, the network data capture has used network tools such as TCPdump and Tshark. Furthermore, we have compared the system with state-of-the-art works [5], [9], [10] to deeply evaluate the performance. The next subsections detail the datasets characteristics, the data preparation, and the evaluation methodology.

A. Datasets Characteristics and Data Preparation

The two datasets employed in this analysis are Botnet 2014 (**dataset 1**) and CTU-13 (**dataset 2**). This subsection describes the main datasets characteristics and data preparation. **Dataset 1** (Botnet 2014) represents generalism, realism, and representativeness from botnet attacks. It achieves the generalism by using 16 botnets variations based on 3 application protocols

[1]The source code is available on: https://github.com/ccsc-research/trusted-system.

[2]Botnet 2014, https://www.unb.ca/cic/datasets/botnet.html, Last accessed on Nov. 2020.

[3]CTU-13 Dataset, https://www.stratosphereips.org/datasets-ctu13, Last accessed on Nov. 2020.

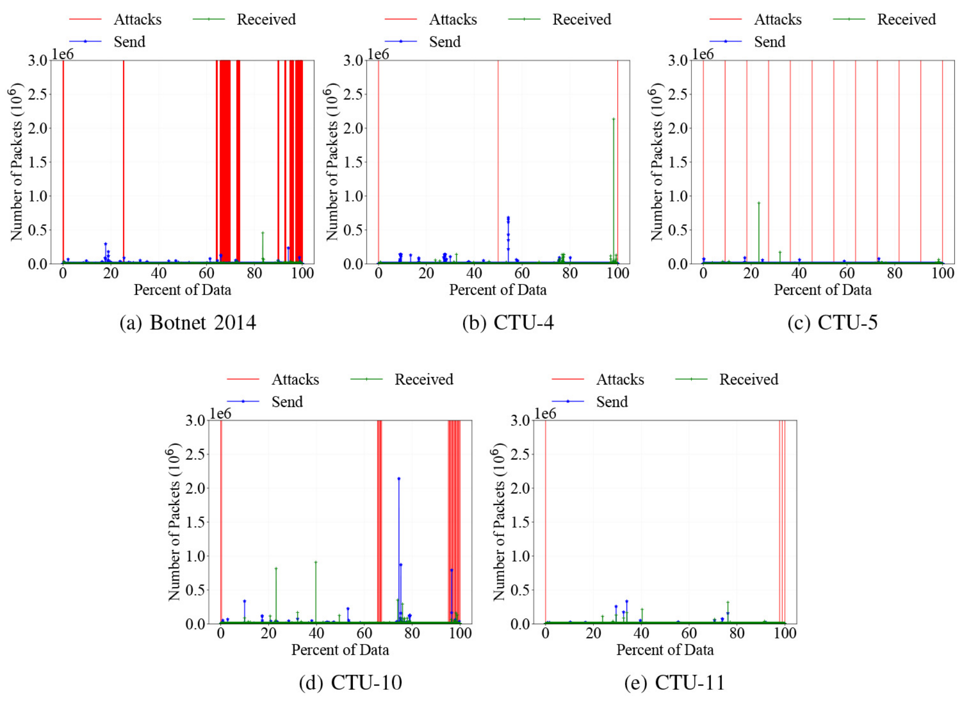


Fig. 4. Datasets packet distribution and attacks.

TABLE I
BOTNET 2014 DATASET

Botnet Name	Type	Data %
Neris	IRC	5,67
Rbot	IRC	0,018
Menti	IRC	0,62
Murlo	IRC	1,06
Tbot	IRC	0,283
Sogou	HTTP	0,019
Virut	HTTP	12,80
NSIS	P2P	0,165
Zeus	P2P	0,109
SMTP Spam	P2P	4,72
UDP Storm	P2P	9,63
Zero Access	P2P	0,221
Weasel	P2P	9,25
Smoke Bot	P2P	0,017
Zeus Control	P2P	0,006
ISCX IRC Bot	P2P	0,387

types. Table I presents the botnets variation (name), application protocol type, and the data percentage generated from each botnet. The realism is provided by mixing real traffic collected from an uncontrolled environment and generated traffic from a controlled environment. The dataset presents representativeness by creating a set of collections made with non-overlapping data of real and artificial traffic [12].

Dataset 2 (CTU-13) contains botnet and normal traffic captured in the Czech Technical University. It has 13 attack scenarios. It uses different botnet malware and amount of bots [13]. This work employs the scenarios 4, 5, 10, and 11, called as CTU-4, CTU-5, CTU-10, and CTU-11, respectively. Table II presents the malware name, duration, and the number of bots for each considered scenario.

In each experiment, the TRUSTED system receives as input dataset 1 or dataset 2. The system groups the network traffic into F flow instances following the 5-tuple $\langle$ Source IP, Destination IP, Source Port, Destination Port, and communication protocol $\rangle$. Then, it extracts statistical features from the flow instances. The features involve the number of bytes sent and received, the start and end time of the packets sending, the start and end time of packet receiving, the interval between the start and end time of sending packets, and the interval between the start and end of receiving packets. Fig. 4 presents the sent and received packet distribution and the attack occurrences in the datasets. Particularly, Fig. 4(a) shows the Botnet 2014 dataset packet distribution, and Figs. 4(b), 4(c), 4(d), 4(e) show the CTU-13 dataset distribution (4, 5, 10, and 11 scenarios). The green, blue, and red lines represent the received packets, sent packets, and attack traffic packets, respectively.

Although both datasets presented attack traffic, they had different behaviors. For instance, the Botnet 2014 and CTU-10 present dense attack traffic, around the 70% and 90% percent of data, while the other datasets show a disperse attack traffic. Thus, the TRUSTED system aims at detecting all data traffic changes and then the attacks.

B. Environment Settings

An **empirical setup** evaluation aims at determining the ideal setting parameters since they influence in the performance results and system operation. These parameters involve the window size, the percent of new instances to consider, concept drift threshold, and clustering distance. Thus, the parameter values vary along the evaluations to select the best setup. Afterward, two environments settings, (i) **offline** and (ii) **online**, employ the best setup obtained through the empirical setup evaluation (see the results in Section VI). Both environments receive as input the datasets. As the TRUSTED system works in online environments, data traffic streams

TABLE II
CTU-13 DATASET

CTU-13 Scenarios	Botnet	Duration (h)	# bots	Botnet flows	Normal flows	C&C flows	Background flows
Scenario 4 (CTU-4)	Rbot	4.21	1	0.15%	2.25%	0.004%	97.58%
Scenario 5 (CTU-5)	Virut	11.63	1	0.53%	3.6%	1.15%	95.7%
Scenario 10 (CTU-10)	Rbot	4.75	10	8.11%	1.2%	0.002%	90.67%
Scenario 11 (CTU-11)	Rbot	0.26	3	7.6%	2.53%	0.002%	89.85%

using the datasets have been created to detect the concept drifts and botnets. The data traffic streams contain the flow instances. Based on the streams, sliding windows are incrementally composed and analyzed. The implementation used Python-based libraries: Scapy[4] v2.4.4, scikit-learn[5] v0.23, scikit-multiflow[6] v0.5.3, and Pandas[7] v1.0.5. Next we detail the offline and online environments.

The **offline** environment was implemented and evaluated on Ubuntu 18.04 LTS with an Intel Core i7 8750H processor and 16GB of RAM. It used the datasets as input to build data stream, i.e., flow instances sequence, arriving one by one in the system. Moreover, again empirical setups have assisted in the definition of the best system parameters (e.g., window size). Finally, we have compared tree instances of the TRUSTED system (state-of-the-art comparison): TRUSTED w/ HAC (the system with unsupervised learning algorithm, similarity analysis, and concept drift identification), FIXED w/ HAC (the system with unsupervised learning algorithm, similarity analysis, and fixed windows, but ignoring concept drift identification) [9], [10], and TRUSTED w/ ARF (the system with supervised Adaptive Random Forest classifier and concept drift identification) [5].

The **online** environment followed an experimental evaluation on an end-to-end wide area network supported by the National Education and Research Network[8] (RNP) from Brazil (partnership with MENTORED project). The RNP spread its backbone through all Brazil states called computational resource islands. Our environment considered a total of two islands, one in the state of *Rio Grande do Sul* (RS) and another in the state of *Minas Gerais* (MG). Each island contains a border router and a virtualization server with Intel Xeon Gold, model 5118, and 256 GB of DDR4 RAM. The RS and MG islands represent the client (target network) and server, respectively. Hence, the RS virtualization server emulated a local area network with Ubuntu Docker containers for each client to reproduce the client traffic. The CTU-13 scenarios have been reproduced in this environment, simulating a real Internet connection between client and server. The client and server send and receive traffic by Scapy Python script following the CTU-13 scenarios. In the client border router, the botnet detection happens. Finally, in this environment, TRUSTED w/ ARF and TRUSTED w/ HAC are evaluated, ignoring the FIXED w/ HAC because of its static properties and the relevant drop detection capacity noted in the offline environment results (see Section VI-B).

In order to evaluate the TRUSTED system, different performance metrics were employed. These metrics quantify the classification models performance. They are accuracy, precision, recall, and F1-score. To obtain these metrics the equations require the values of true positive (TP), true negative (TN), false positive (FP), and false negative (FN). Thus, the accuracy (Eq. (2)) refers to the proportion of flow instances classified correctly concerning all traffic samples. The precision (Eq. (3)) estimates the percentage of true positives among all flow instances classified as positive. The recall (Eq. (4)) consists of the true positives percentage out of all flow instances whose expected class is positive. The F1-Score (Eq. (5)) makes a relationship between the precision and recall by estimating the harmonic mean. Therefore, results follow these metrics.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

$$F1\ Score = 2 * \frac{Recall * Precision}{Recall + Precision}. \tag{5}$$

VI. RESULTS

This section presents the TRUSTED system performance evaluation results. The system seeks to identify concept drifts and detects botnets. Next, it presents the empirical setup evaluation results considering different values for the setting parameters (e.g., window size and clustering distances), the offline environment results, and the online environment results. For both environments, it is shown the TRUSTED instances comparison results, i.e., state-of-the-art comparison.

A. Empirical Setup

As mentioned, an empirical evaluation of the system parameters has been pursued to determine the best setup based on classifiers metrics results. These parameters involve the window size, percent of new data, concept drift threshold, and clustering distance. Fig. 5 presents accuracy, precision, recall, and F1-Score results. Initially, tests intended to investigate the data window size influence, considering sizes of 100, 250, 500, 750, and 1000, as shown Fig. 5(a). Based on the results, larger the window size, better the system performance concerning the accuracy, recall, and F1-Score metrics. But, the precision suffered a slight decrease due to false positives. The biggest window size (1000) received more flow instances, offering

[4]Scapy, https://scapy.net/, Last accessed on Oct. 2020.

[5]Scikit-learn, https://scikit-learn.org/stable/, Last accessed on Nov. 2020.

[6]Scikit-multiflow, https://scikit-multiflow.github.io/, Last accessed on Nov. 2020.

[7]Pandas, https://pandas.pydata.org/, Last accessed on Nov. 2020.

[8]RNP: https://www.rnp.br/, Last accessed on Nov. 2020.

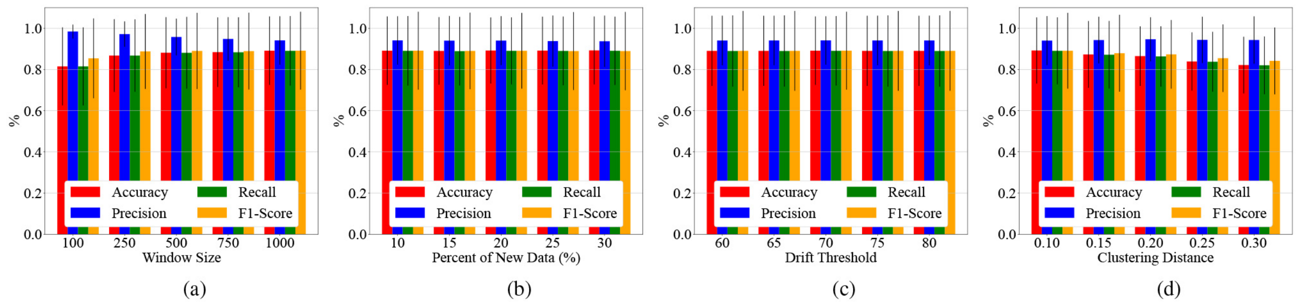


Fig. 5. The TRUSTED system empirical setup.

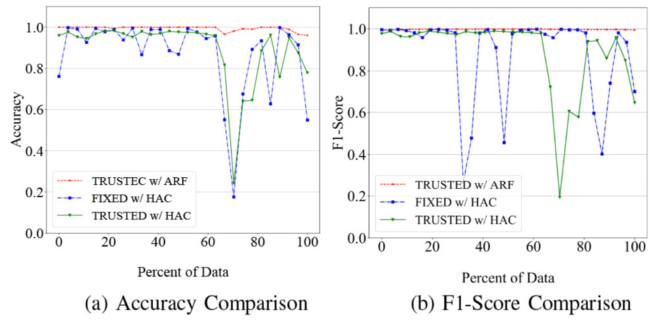
more data examples for TRUSTED to identify the concept drifts. Figs. 5(b) and 5(c) show similar results for the new data percentage and the concept drift threshold. Once the percentage of new data ranges from 10% to 30%, the system started analyzing concept drifts with a considerable amount of data, i.e., 10% of new data contain sufficient flow instances to identify drifts. The concept drift threshold is based on the AUC metric results obtained through the discriminative classifier. Experiments have analyzed the threshold ranging from 60 to 80. When the classifier result achieved an AUC higher than 60, the system was able to identify a concept drift with the same performance when using 80 as the threshold. Therefore, such tested values did not show significant changes in system performance. Hence, experiments have shown as best setup 20% for the new data percentage and 70 as drift threshold.

Fig. 5(d) presents the results considering the variation in the cluster maximum Euclidean distance defined in Algorithm 2 (see Section IV). The data input of the algorithm varies between 0 and 1 (such values represent the set of flow features). Distances between 0.10 and 0.30 have been tested, increasing 0.05 per round. According to the results, the greater the maximum cluster distance, the lower is the system performance. The increase of distance inside the clusters generates a high generalism, which affected the system performance when testing values higher than 0.1. Moreover, distance with values lower than 0.1 should cause overfitting in the clustering algorithm. Thus, given the results and discussions over the TRUSTED system empirical setup, the best setup with optimized system values has window size equal to 1000, percentage of new data of 20%, concept drift threshold of 70%, and the cluster maximum distance of 0.10.

B. Offline Environment

Fig. 6 presents the comparison results, in an offline environment, between TRUSTED w/ HAC (the system with unsupervised learning algorithm and concept drift identification), TRUSTED w/ ARF (the system with Adaptive Random Forest algorithm presented in [5]), and FIXED w/ HAC (the system with unsupervised algorithm, fixed windows, and without concept drift identification [9], [10]). Results follow the accuracy, precision, recall, and F1-Score metrics.

Fig. 6(a) presents the accuracy results from the three experiments over the Botnet 2014 dataset. TRUSTED w/ ARF has presented the best results achieving 99% of accuracy and high stability. However, this result is due to the use of



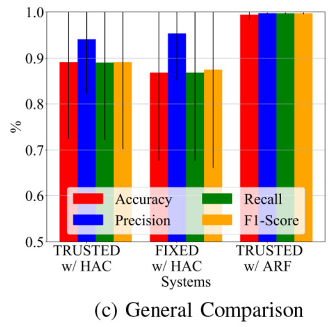
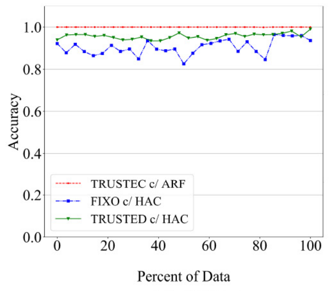
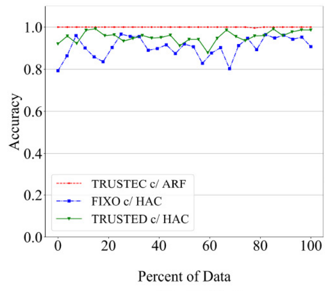


Fig. 6. Systems Comparison with Botnet 2014 Dataset.

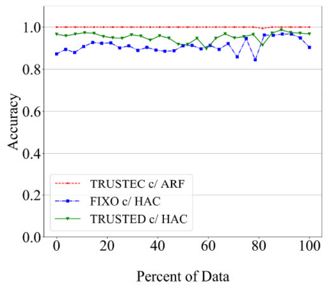
supervised learning models, i.e., re-training the classification model with labeled instances whenever it identifies a concept drift [5]. The use of online supervised learning is expensive and slow in the network security environment [3]. Compared to FIXED w/ HAC, TRUSTED w/ HAC has a more stable classification and adequate generality (without overfitting) in the botnet detection, as shown in Fig. 6. Furthermore, analyzing the accuracy and F1-Score performance of TRUSTED and FIXED systems on the green and blue lines, showed in Fig. 6(a) and 6(b), confirm that FIXED presented a high traffic generalization, and thus, incorrectly classifies traffic when occurring an attack. During the occurrence of attack overload, the TRUSTED system performance loss is less than FIXED. Moreover, TRUSTED keeps the accuracy more stable than FIXED during the evaluation in the Botnet 2014 dataset. Fig. 6(c) shows all the metrics results. TRUSTED w/ ARF reached 99% of performance in all metrics, being the best evaluation results. The results of the unsupervised methods show TRUSTED w/ HAC with high performance and stability over the base showing the smallest standard deviation during the evaluation indicated by the error line in Fig. 6(c).



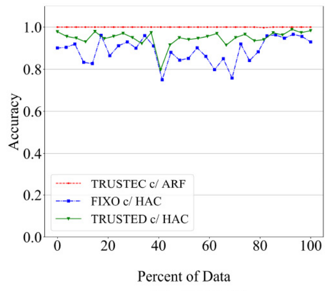
(a) CTU-4



(b) CTU-5

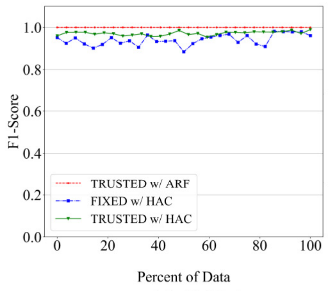


(c) CTU-10

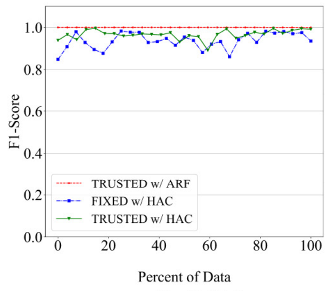


(d) CTU-11

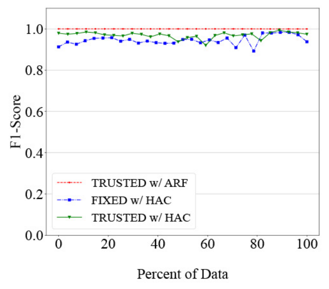
Fig. 7. Accuracy Comparison CTU-13 Dataset Offline Evaluation.



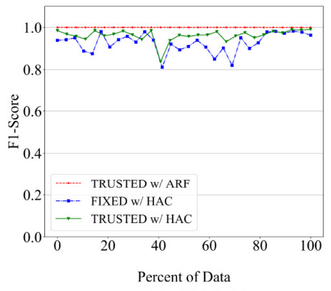
(a) CTU-4



(b) CTU-5

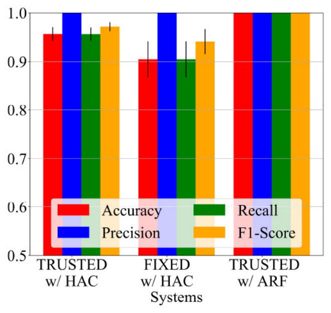


(c) CTU-10

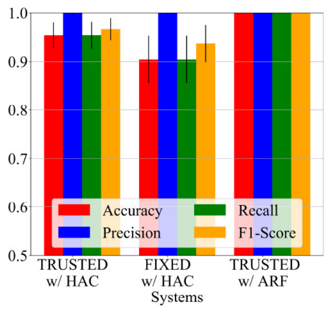


(d) CTU-11

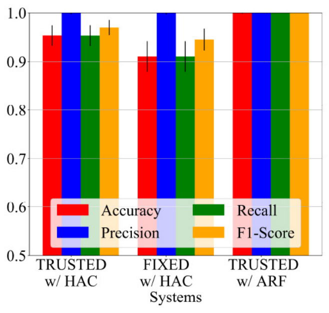
Fig. 8. F1-Score Comparison CTU-13 Dataset Offline Evaluation.



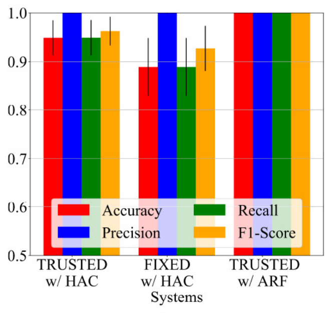
(a) CTU-4



(b) CTU-5



(c) CTU-10



(d) CTU-11

Fig. 9. General Comparison CTU-13 Dataset Offline Evaluation.

Although TRUSTED w/ HAC overcomes the FIXED w/ HAC in accuracy, recall and F1-Score, it suffers a slight decrease in precision due to more false positives.

In order to ensure the evaluation soundness besides the Botnet 2014 dataset, experiments followed four scenarios for CTU-13 dataset. Figs. 7 and 8 presents the accuracy and F1-score metrics over the scenarios evaluated comparing the three approaches. In all executions presented in Figs. 7(a), 7(b), 7(c), 7(d), 8(a), 8(b), 8(c), 8(d), TRUSTED w/ HAC proposed in this article overcomes the traditional FIXED w/ HAC strategy. Although TRUSTED w/ ARF [5] presents better results, it re-trains the classifier after each classification, and this supervised strategy is infeasible in network security management according to [3]. Also, Fig. 9 supports Figs. 7 and 8 results by presenting for all scenarios the precision and recall metrics that match the difference presented by the accuracy over the datasets. These other metrics are necessary since the datasets are unbalanced.

The results reached by TRUSTED w/ ARF in the offline environment consider pre-training with offline samples and labels. Besides, it is supplied with the correct labels to re-train the classifier whenever it classifies a new instance. We have considered this as the ideal environment for supervised approaches. However, in online scenarios without label delivery, it should not perform well, see Section VI-C. In online environment, there is no supply of offline samples and labels to the supervised approach.

C. Online Environment

Figs. 10 and 11 presents the results of accuracy and F1-score) in the four scenarios related to CTU-13 online evaluated. As a general observation, TRUSTED w/ HAC has a decrease in its results in all scenarios. This occurrence is due to a lack of instances at the beginning of the evaluation, where the data window has just a few instances. Also, the continuous decrease presents a controversial point against the empirical setup since it shows that the more instances to analyze better the result.

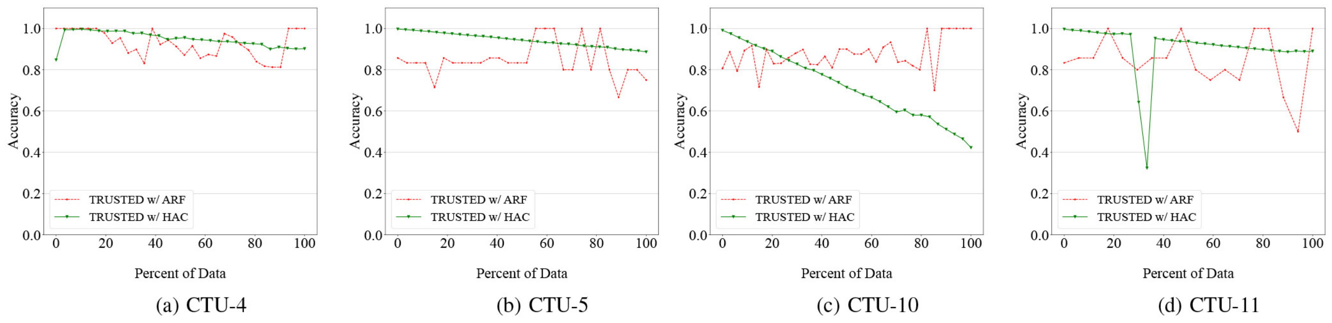


Fig. 10. Accuracy Comparison CTU-13 Dataset Online Evaluation.

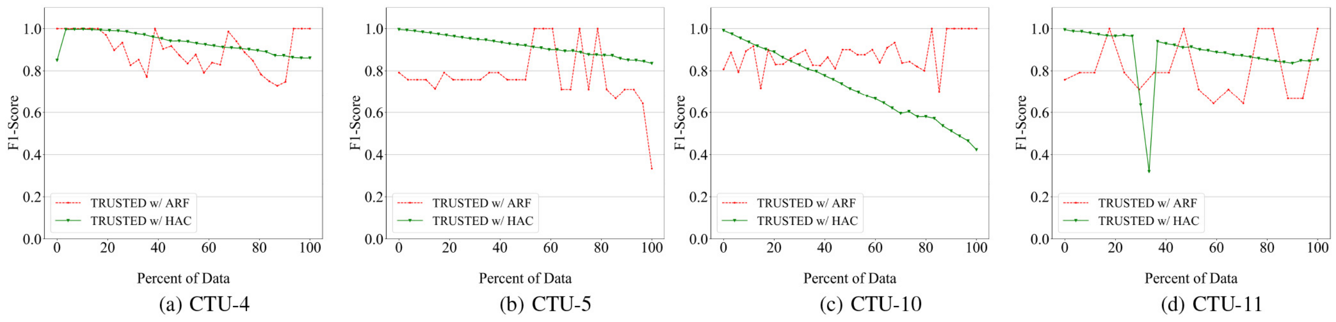


Fig. 11. F1-Score Comparison CTU-13 Dataset Online Evaluation.

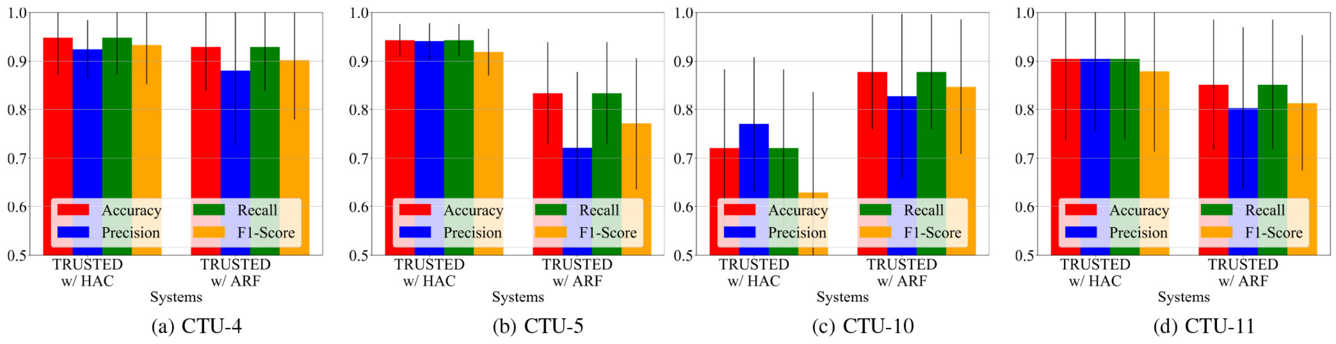


Fig. 12. General Comparison CTU-13 Dataset Online Evaluation.

TRUSTED w/ ARF did not receive an offline sample to pre-train the classifier, just got supplied with labels to re-train during the evaluation. This fact causes the approach fluctuation, presenting the performance going down and up by re-training the model to classify the instances correctly. Also, it has found more drifts since it monitors the performance results. In every abrupt change in the classification performance, this approach was able to adapt the data window and receive the labels for re-training. These results emphasize that supervised approaches must have prior training to reach a high-performance classification (e.g., more than 99% of accuracy as seen in Section VI-B). Therefore, in a non-ideal scenario (e.g., real world), the supervised approaches are unfeasible.

Although TRUSTED w/ HAC is unsupervised, it is stable during botnet detection compared to the supervised TRUSTED w/ ARF. Except in Figs. 10(c) and 11(c), where the TRUSTED w/ HAC accuracy and F1-Score constantly drops during the botnet detection, it has happened because the system did not identify any concept drifts in data distribution. Hence, TRUSTED w/ HAC did not adapt to detect botnet. Besides, Fig. 12 presents the results for TRUSTED w/ HAC being the worst results on CTU-10, Fig. 12(c). However, in all other online evaluations, the unsupervised approach proposed in this article performed better than supervised.

In Figs. 10(d) and 11(d), there is an abrupt decrease in the TRUSTED w/ HAC results near of 40% of data. This decrease refers to an attack in the dataset. The system could not identify the difference between the benign and the botnet traffic at that moment. This misunderstanding occurs when benign traffic has high similarity between devices or when the botnet traffic has a significant difference in each bot.

D. Discussion and Limitations

The performance evaluation demonstrates that the TRUSTED system is able to detect malicious network activity with high accuracy and precision. This section discusses further aspects of the system and its limitations, such as privacy concerns and operation costs, along with the challenges of the system on network security.

For privacy concerns, the TRUSTED system does not perform payload inspection (user-level data) since it considers flow-level features. Thus, it overcomes the data acquisition challenge (see Section II). Moreover, the system is encryption independent because it considers usual network traffic features (e.g., amount of packets). However, the system does not perform secure transmission, being an open topic for further investigations. Encryption techniques to prevent third-parties from accessing sensitive data can improve user privacy.

For operational costs concerns, the TRUSTED system keeps an instance-based sliding window. This sliding window has a maximum amount of data following a first-in-first-out rule. Thereby, it prevents a high operational cost and also prevents under and overfitting by changing the inputs for the system botnet detection. The evaluation presented an empirical setup estimating the sliding window fixed maximum size, but these results only refer to the used dataset. In the real world, a thorough evaluation is required to adapt to the size, heterogeneity, and network purposes.

For supervised learning limitations, the TRUSTED system carries out unsupervised online learning using hierarchical clustering with similarity analysis over a sliding window. Although this strategy presents general good results in the evaluated scenarios, it may suffer concerning long-term activities history and prone to wrong classifications (see Section VI-C, scenarios CTU 10 and 11). As the system works with separate phases (e.g., concept drift detection and botnet detection), the botnet detection phase can be improved by other strategy or solution in the future, such as adaptive Random Forest.

The need for concept drift detection is evident in the online learning literature since without concept drift awareness, the classification models became obsolete [9], [10], [20]. Thus, the TRUSTED system presented an unsupervised approach to deal with the possible drifts in the data stream. This approach is sensitive to its parameters setting (e.g., percent of new data and threshold of drift detection); hence, this article presented an empirical setup evaluation over these parameters. Note that the percentage of new instances and the concept drift threshold do not show any performance fluctuation in our evaluation. However, in other network setup and context, it is necessary a thorough measurement to adapt the approach to the size, heterogeneity, and network purposes.

VII. Conclusion and Future Works

This article introduced the TRUSTED system, which detects botnets in an unsupervised and online way considering the concept drifts in network traffic. The system builds a sliding data window identifying possible concept drifts to provide a data set optimized for the botnet detection phase that clusters the flow instances and analyzes the similarity. The system assessment followed a trace-driven approach, employing a database containing various types of attacks, normal traffic, and an experimental scenario. The results demonstrate that the system reached 87% - 95% accuracy and a low false-positive rate. Also, concept drift identification has increased the TRUSTED system performance compared to works using the fixed window strategy.

References

[1] P. Wainwright and H. Kettani, "An analysis of botnet models," in *Proc. 3rd Int. Conf. Comput. Data Anal.*, 2019, pp. 116–121. [Online]. Available: https://doi.org/10.1145/3314545.3314562

[2] M. Ammar, G. Russello, and B. Crispo, "Internet of Things: A survey on the security of IoT frameworks," *J. Inf. Security Appl.*, vol. 38, pp. 8–27, Feb. 2018.

[3] A. Al Shorman, H. Faris, and I. Aljarah, "Unsupervised intelligent system based on one class support vector machine and Grey Wolf optimization for IoT botnet detection," *J. Ambient Intell. Humanized Comput.*, vol. 11, no. 7, pp. 2809–2825, 2020.

[4] V. Carela-Español, P. Barlet-Ros, A. Bifet, and K. Fukuda, "A streaming flow-based technique for traffic classification applied to 12+ 1 years of Internet traffic," *Telecommun. Syst.*, vol. 63, no. 2, pp. 191–204, 2016.

[5] P. Casas, P. Mulinka, and J. Vanerio, "Should i (re) learn or should i go (on)? Stream machine learning for adaptive defense against network attacks," in *Proc. 6th ACM Workshop Moving Target Defense*, 2019, pp. 79–88.

[6] Ö. Gözüaçık, A. Büyükçakır, H. Bonab, and F. Can, "Unsupervised concept drift detection with a discriminative classifier," in *Proc. 28th ACM Int. Conf. Inf. Knowl. Manag.*, 2019, pp. 2365–2368.

[7] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surveys*, vol. 46, no. 4, pp. 1–37, 2014.

[8] P. Barthakur, M. Dahal, and M. K. Ghose, "CluSiBotHealer: Botnet detection through similarity analysis of clusters," *J. Adv. Comput. Netws*, vol. 3, no. 1, pp. 49–55, 2015.

[9] X. Yu, X. Dong, G. Yu, Y. Qin, and D. Yue, "Data-adaptive clustering analysis for online botnet detection," in *Proc. 3rd Int. Joint Conf. Comput. Sci. Optim.*, vol. 1, 2010, pp. 456–460.

[10] M. Yahyazadeh and M. Abadi, "BotGrab: A negative reputation system for botnet detection," *Comput. Elect. Eng.*, vol. 41, pp. 68–85, Jan. 2015.

[11] A. A. Daya, M. A. Salahuddin, N. Limam, and R. Boutaba, "BotChase: Graph-based bot detection using machine learning," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 1, pp. 15–29, Mar. 2020.

[12] E. B. Beigi, H. H. Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches," in *Proc. IEEE Conf. Commun. Netw. Security*, 2014, pp. 247–255.

[13] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Comput. Security*, vol. 45, pp. 100–123, Sep. 2014.

[14] A. Bijalwan, V. K. Solanki, and E. S. Pilli, "Botnet forensic: Issues, challenges and good practices," *Netw. Protocols Algorithms*, vol. 10, no. 2, pp. 28–51, 2018.

[15] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proc. SIAM Int. Conf. Data Min.*, 2007, pp. 443–448.

[16] I. Hafeez, M. Antikainen, A. Y. Ding, and S. Tarkoma, "IoT-KEEPER: Detecting malicious IoT network activity using online traffic analysis at the edge," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 1, pp. 45–59, Mar. 2020.

[17] T. S. Sethi and M. Kantardzic, "On the reliable detection of concept drift from streaming unlabeled data," *Expert Syst. Appl.*, vol. 82, pp. 77–99, Oct. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417417302439

[18] G. H. Ribeiro, E. R. de Faria Paiva, and R. S. Miani, "A comparison of stream mining algorithms on botnet detection," in *Proc. 15th Int. Conf. Avail. Rel. Security*, 2020, pp. 1–10.

[19] W. Wu, J. Alvarez, C. Liu, and H.-M. Sun, "Bot detection using unsupervised machine learning," *Microsyst. Technol.*, vol. 24, no. 1, pp. 209–217, 2018.

[20] V. Siloa and B. Soniva, "Data stream clustering for botnet detection," in *Proc. Int. Conf. Soft-Comput. Netw. Security (ICSNS)*, 2018, pp. 1–5.

[21] S. Khanchi, A. Vahdat, M. I. Heywood, and A. N. Zincir-Heywood, "On botnet detection with genetic programming under streaming data label budgets and class imbalance," *Swarm Evol. Comput.*, vol. 39, pp. 123–140, Apr. 2018.

[22] Z. Liu, X. Yun, Y. Zhang, and Y. Wang, "CCGA: Clustering and capturing group activities for DGA-based botnets detection," in *Proc. 18th IEEE Int. Conf. Trust Security Privacy Comput. Commun. 13th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, 2019, pp. 136–143.

[23] L. Mai and M. Park, "A comparison of clustering algorithms for botnet detection based on network flow," in *Proc. 8th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, 2016, pp. 667–669.

[24] D. C. Le, A. N. Zincir-Heywood, and M. I. Heywood, "Data analytics on network traffic flows for botnet behaviour detection," in *Proc. IEEE Symp. Series Comput. Intell. (SSCI)*, 2016, pp. 1–7.

[25] J. van Roosmalen, H. Vranken, and M. van Eekelen, "Applying deep learning on packet flows for botnet detection," in *Proc. 33rd Annu. ACM Symp. Appl. Comput.*, 2018, pp. 1629–1636. [Online]. Available: https://doi.org/10.1145/3167132.3167306

[26] A. Sapello, C. Serban, R. Chadha, and R. Izmailov, "Application of learning using privileged information (LUPI): Botnet detection," in *Proc. 26th Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2017, pp. 1–8.

[27] F. Haddadi and A. N. Zincir-Heywood, "Benchmarking the effect of flow exporters and protocol filters on botnet traffic classification," *IEEE Syst. J.*, vol. 10, no. 4, pp. 1390–1401, Dec. 2016.

[28] E. R. Faria, J. A. Gama, and A. C. P. L. F. Carvalho, "Novelty detection algorithm for data streams multi-class problems," in *Proc. 28th Annu. ACM Symp. Appl. Comput.*, 2013, pp. 795–800. [Online]. Available: https://doi.org/10.1145/2480362.2480515

[29] J. W. Ryu, M. M. Kantardzic, M.-W. Kim, and A. Ra Khil, "An efficient method of building an ensemble of classifiers in streaming data," in *Big Data Analytics*, S. Srinivasa and V. Bhatnagar, Eds. Heidelberg, Germany: Springer, 2012, pp. 122–133.

[30] T. S. Sethi, M. Kantardzic, and H. Hu, "A grid density based framework for classifying streaming data in the presence of concept drift," *J. Intell. Inf. Syst.*, vol. 46, no. 1, pp. 179–211, 2016.

[31] J. Lee and F. Magoules, "Detection of concept drift for learning from stream data," in *Proc. IEEE 14th Int. Conf. High Perform. Comput. Commun. IEEE 9th Int. Conf. Embedded Softw. Syst.*, 2012, pp. 241–245.

[32] G. Ditzler and R. Polikar, "Hellinger distance based drift detection for nonstationary environments," in *Proc. IEEE Symp. Comput. Intell. Dyn. Uncertain Environ. (CIDUE)*, 2011, pp. 41–48.

[33] L. I. Kuncheva and W. J. Faithfull, "PCA feature extraction for change detection in multidimensional unlabeled data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 1, pp. 69–80, Jan. 2014.

[34] A. A. Qahtan, B. Alharbi, S. Wang, and X. Zhang, "A PCA-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2015, pp. 935–944.

[35] B. H. Schwengber, A. Vergütz, N. Prates, and M. N. Lima, "A method aware of concept drift foronline botnet detection (to appear)," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2020, pp. 1–6.

[36] J. Ye, Z. Zhao, and H. Liu, "Adaptive distance metric learning for clustering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2007, pp. 1–7.

[37] N. Grira, M. Crucianu, and N. Boujemaa,"Unsupervised and semi-supervised clustering: A brief survey," in *A Review of Machine Learning Techniques for Processing Multimedia Content*. Le Chesnay Cedex, France: INRIA Rocquencourt, 2004, pp. 9–16.

Bruno Henrique Schwengber received the B.Sc. degree in computer science from the Federal University of Technology—Paraná, Brazil. He is currently pursuing the M.Sc. degree in computer science with the Federal University of Paraná working on network security. His research background comprises SDN simulations and the evaluation of the impact of concept drifts in online botnet detection.

Andressa Vergütz (Student Member, IEEE) is currently pursuing the Ph.D. degree in computer science with the Federal University of Paraná, Brazil. Her research interests include smart healthcare, network performance management, wireless body networks, IoT, IoMT, network security, and data analysis. She is a member of Wireless and Advanced Networks Research Team and Center for Computational Security Science. She is a Student Member and of the Brazilian Computer Society (SBC).

Nelson G. Prates, Jr. received the B.Sc. degree in information systems from the Federal University of Santa Maria and the M.Sc. degree in computer science from the Federal University of Parana, Brazil. His research background comprises Internet-of-Things (IoT) security and software-defined networks, focused on providing more security, and privacy to users. His research background also involves the development of experimental testbeds to IoT networks and SDN.

Michele Nogueira received the Doctoral degree in computer science from the Laboratoire d'Informatique de Paris VI, UPMC-Sorbonne University. She is an Associate Professor with the Computer Science Department, Federal University of Minas Gerais. She was in a Sabbatical Leave with Carnegie Mellon University from 2016 to 2017. Her research interests include wireless networks, network security, and resilience. She was an Associate Technical Editor for the *IEEE Communications Magazine* and the *Journal of Network and Systems Management*.