Learning from Network Data Changes for Unsupervised Botnet Detection

Bruno Henrique Schwengber, **Andressa Vergütz**, **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 security management to improve efficiency, robustness and adaptability in 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 who employ an approach 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, TRUSTED 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 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. These techniques are restrictive when dealing with dynamic and adversarial environments (e.g., network traffic) [5]. Nevertheless, high-speed streams evolve and change their statistical distributions. These changes are called concept drifts [6]. When a concept drift occurs, the classification model becomes obsolete due to the data statistical distribution change. This negatively affects 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 traffic speed and volume [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 the new data traffic distributions [5]. Nevertheless, the use of supervised classifiers is slow, costly, and prone to error for the malicious traffic detection given the high speed and data volume [3], [11]. Thus, since concept drift improves 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 online botnet detection. Thus, by identifying concept drifts, clustering, and the similarity analysis, 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 the botnets. The major contributions of this article are as follows.

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

Manuscript received April XX, XXXX; revised August xx, 2021. 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.

B. H. Schwengber, A. Vergütz and N. G. Prates Jr. are with the Computer Science Department at Federal University of Paraná, Brazil. Emails: {bhschwengber,avergutz,ngpjunior}@inf.ufpr.br. M. Nogueira is with the Computer Science Department at both Federal University of Paraná and Federal University of Minas Gerais, Brazil. Email: michele@dcc.ufmg.br

a complete unsupervised environment.

We evaluate the TRUSTED system by 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 for analysis. In this work, we consider, as an instance, a set of features from the network traffic flow. Thus, we build a data stream from the extracted features to emulate an online environment for evaluation. Having as input the data stream, TRUSTED delimits the data window, identifies concept drifts, and detects botnets. The results demonstrate that the TRUSTED system reaches 87% - 95% accuracy, detecting botnet using concept drift identification to provide an optimized input for online learning. Comparing 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 the concept drift in online learning and details the TRUSTED system. Section V presents the performance evaluation methodology and discusses the results. Finally, Section VII concludes the article and discusses future works.

II. Challenges on Network Security Management

Network security management solutions follow data collection, analysis, and identification [14]. These phases are necessary to reach the solution goal. This section enlightens the security management main issues related to data acquisition, operational costs, attack detection, device heterogeneity, and concept drifts. Network security management solutions must 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 security management involves operational costs [4], [5]. The operational cost increases with the large-scale 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 information [15].

Network security management solutions rely on attacks and traffic signatures or anomaly detection [16], [14]. However, many of these solutions depend on a labeled dataset to train the models, resulting in three main problems. First, the 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, machine learning supervised solutions cannot accurately detect botnet attacks because new botnets launch several zero-day attacks every day. Hence, the modeling does not know about previous attacks [3]. The security management solutions must employ online unsupervised algorithms to address these limitations since it does not require labeled training data or previous knowledge about attack signatures to perform traffic classification. Also, they work with a windowing strategy to avoid large data storage.

Real-world data stream (e.g., network traffic) evolves and changes over time. These changes are known as concept drift [6]. When a concept drift occurs, the classification model becomes outdated because data distribution has changed, and then it can not correctly detect botnets [7]. Supervised and unsupervised approaches exist to identify concept drift. Although the supervised approaches present only a few false-positive drifts, they need labels to perform the identification. However, the 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 [7], [6], [17], and network attack detection [4], [5]. 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]. 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], presenting challenges such as storage and human intervention to label data, i.e.,making it 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], [18]. Other works employ online unsupervised approaches to detect botnets by timely analyzing network traffic. Such online approaches are faster than offline due to timely data analysis, improving the decision making process [9], [10].

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 of 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% 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 a K-Means clustering algorithm. The proposed technique analyzes the similarity between the instances in the same cluster to detect the bot. When a cluster exceeds a certain similarity threshold, i.e.,when it has similar instances, the group is labeled as a suspect. However, this technique does not consider concept drifts related to data distribution, which can negatively affect its performance.

Yahyazadeh e Abadi [10] presented an online unsupervised botnet detection system supported by negative reputation 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 activities 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 the occurrence of 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.

This work presents the TRUSTED system, an unsupervised online botnet detection system with the perception of concept drifts. TRUSTED detects network traffic from botnets in an online way to streamline decision making. Unlike other works, the TRUSTED system identifies concept drifts to keep the classification model updated continuously.

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 occurs 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 has two data blocks *W0* and *W1*, containing the old and recent classification results, respectively. After each classification, the ADWIN method compares both data blocks using Hoeffding Bound to measure if they are large and distinct enough. While there is no drift, the window keeps growing. If a drift occurs, the window shrinks, keeping only the recent data.

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 label, i.e.,a new pattern in the data distribution [19], [20], [21]. Although the novelty detection strategy has resulted in the successful identification of concept drifts, it suffers from the curse of dimensionality because it depends on distance measurement [17]. The multivariate distribution monitoring strategy monitors the data distribution of each statistical properties. In general, this strategy uses two data blocks to summarize old data and current data. Then, it compares both data blocks analyzing statistical data properties, such as mean and standard deviation [6], [22], [23], [24], [25]. Although multivariate distribution monitoring is prone to false alarms, it presents a fast answer to the system. Thus, it works better than novelty 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 (old and new data). This method identifies drift by labeling the old and new data with 0 e 1, respectively, and submitting the data to a train and test classification. If the classification gets an AUC score higher than 70%, it means that data is a drift. Otherwise, it means that there is no drift identification in data properties. In this article, we use 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**nsupervi**S**ed bo**T**n**E**t **D**etection. Its goal is to detect botnets in an online way 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 [26], this work presents the D3N in a complete unsupervised environment as it compares with unsupervised botnet detection approaches.

A. The TRUSTED Specification

The TRUSTED system follows the phases of pre-processing, the identification of concept drift, and botnet detection, illustrated in Fig. 1. The system captures network

Fig. 1. The TRUSTED System Architecture

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 the features. The second phase builds the sliding data window and identifies possible concept drift in the data behavior. Finally, the third phase detects malicious behavior by clustering the data stream and using similarity analysis.

1) Pre-Processing: The pre-processing phase is responsible for collecting network traffic, separating it by flows, and extracting the statistical features to create flow instances. First, the TRUSTED system splits the collected traffic into flows $F = \{p_1, p_2, p_3, ..., p_t\}$ following the tuple ⟨ Source IP, Destination IP, Source Port, Destination Port and communication protocol⟩. 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 $X^F = \{x_1, x_2, x_3, ..., x_i\}$, where x is an extracted feature and i is an index associated to each feature. Thus, the next phase creates the data window $W = \{X_1^F, X_2^F, X_3^F, ..., X_z^F\}$ that contains the flow instances.

2) Concept Drift Identification: This phase receives as input the set of flow instances from the pre-processing phase. It first constructs the sliding window and then identifies the concept drift. The sliding window comprises data of type X^F arriving from the pre-processing phase and organized following the first-in/first-out order. The concept drift identification checks the data window looking for concept drifts.

The sliding window has a fixed size separated into two blocks, old and new data blocks, presented in Fig. 2. Both blocks have a fixed size and receive as input data of type X^F. While the data window is not full of old or new data, it keeps adding new instances X^F to fill it. When the window gets full, the TRUSTED system runs the concept drift identification.

The concept drift identification uses a non-linear discriminative drift detector (D3N) proposed in a previous work [26] and adapted from the (D3) method [6]. The motivation for this new drift detector lies in the fact that network traffic statistical distributions change inversely related to the features (e.g.,more bytes and fewer packets per flow). Algorithm 1 details the abstract steps of the D3N method. The method is an unsupervised and multivariate distribution detector. Again, this method was required because the addressed scenario needs an unsupervised and multivariate distribution drift detector given its multidimensional stream.

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
```

Algorithm 1 receives as inputs data stream of type X^F, the window size (δ), the percentage of new instances (σ), and a threshold (β). 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 $\delta(1 + \sigma)$. 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. Otherwise, the system applies labels to the instances (label 0 for old data and label 1 for new data). Formally, in lines 8 and 9, the algorithm defines δ instances with label 0 and σ instances with label 1. For visual comprehension, Fig. 2 shows the data window division. The system uses these labeled instances to train and test a decision tree classifier in line 10. If the classification result has a performance higher than β, it notifies a drift occurrence.

Fig. 2 shows how the data window construction works for concept drift identification. The full data window has a specific size defined as $\delta(1 + \sigma)$ and contains the old δ and new $\delta\sigma$ instances of network data. As TRUSTED works with streaming data, network data comes constantly. Note that the network traffic has a pre-processing phase before the drift identification. The system identifies a concept drift when the decision tree classifier separates the instances with an area under the curve (AUC) performance higher than β. Line 11 of the algorithm 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, TRUSTED replaces a set of σ 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.

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 a maximum Euclidean distance γ. Moreover, agglomerative algorithms use a tree data structure to manage the clusters [27]. 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 each flow instance as 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 maintaining the maximum Euclidean distance γ. In lines 8 and 9, the algorithm removes the old clusters and adds the new clusters to the clusters group A. In line 10, the algorithm adds the set of clusters A to the hierarchical tree T.

From the clusters created by the Algorithm 2, the system performs the similarity analysis. It calculates the euclidean

Fig. 3. Hierarchical Tree (dendrogram)

Algorithm 2 HAC: Hierarchical Agglomerative Clustering

```
Input: stream, $\gamma$
Output: Hierarchical Tree T
 1: A ← ⊘ {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
```

distance between all flow instances of the same cluster following the Equation 1. 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. This threshold determines the critical distance between the flow instances to detect botnets. 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 datasets involve the Botnet 2014[1] (**dataset 1**) from University of New Brunswick [12], and CTU-13[2] (**dataset 2**) from Stratosphere Lab [13]. We have employed these datasets for both environment settings since their 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 [9], [10], [5] to deeply evaluate the performance. The next subsections detail the datasets characteristics, the data preparation, and 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 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].

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

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]. For our performance evaluation, we have considered the 4, 5, 10, and 11 scenarios, called as CTU-4, CTU-5, CTU-10, and CTU-11. Table II presents the malware used, duration, and the number of bots from each scenario considered.

The TRUSTED system receives as input both datasets. It groups the network traffic into F flow instances following the 5-tuple ⟨ Source IP, Destination IP, Source Port, Destination

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

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

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%

(a) Botnet 2014 (b) CTU-4 (c) CTU-5

(d) CTU-10 (e) CTU-11

Fig. 4. Datasets packet distribution and attacks

Port, and communication protocol ⟩. 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. Specifically, Fig. 4a shows the Botnet 2014 dataset packet distribution, and Figs. 4b, 4c, 4d, 4e 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 presented more dense attack traffic around the 70% and 90% percent of data, while the other datasets showed disperse attack traffic. Thus, TRUSTED aims to detect all data traffic changes and then the botnet attacks.

B. Environment Settings

We have conducted an **empirical setup** evaluation to determine the ideal setting parameters since they influence in the performance results and system operation. These parameters involve the window size, percent of new data, concept drift threshold, and clustering distance. Thus, we have changed the parameters values along the evaluations to select the best setup. Afterward, we have created two environments settings, *(i)* **offline** and *(ii)* **online**, employing the best setup obtained through the empirical setup evaluation (see the results in Subsection VI). Both environments receive as input the datasets. As TRUSED works in online environments, we have created data traffic streams using the datasets to detect the concept drifts and botnets. The data traffic streams contain the flow instances. Based on the streams, we have created the data sliding windows and analyzed them incrementally. The implementation used Python-based libraries: Scapy[3] v2.4.4, scikit-learn[4] v0.23, scikit-multiflow[5] v0.5.3, and Pandas[6] 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

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

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

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

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

the system. Moreover, we have evaluated an empirical setup to define the ideal 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[7] (RNP) from Brazil (partnership with MENTORED project). The RNP spread its backbone through all Brazil states called computational resource islands. Our environment considered two islands (or states), *Rio Grande do Sul* (RS) and *Minas Gerais* (MG). Each island contains a border router and a virtualization server with Intel(R) Xeon(R) 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. We have reproduced the CTU-13 scenarios 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, we have executed the botnet detection. Finally, in this environment, we have evaluated TRUSTED w/ ARF and TRUSTED w/ HAC, ignoring the FIXED w/ HAC because of its static properties and the relevant drop detection capacity noted in the offline environment results (see Subsection VI-B).

To evaluate TRUSTED, 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, we have based the results on these metrics.

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

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

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

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

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

VI. Results

This section presents the TRUSTED system performance evaluation results. The system seeks to identify concept drifts and detects botnets. Next, we present 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, we show the TRUSTED instances comparison results, i.e., state-of-the-art comparison.

A. Empirical Setup

We have evaluated the system parameters to determine the ideal 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, we have analyzed the data window size influence, considering sizes of 100, 250, 500, 750, and 1000, as shown Fig. 5a. 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 more data examples for TRUSTED to identify the concept drifts. Figs. 5b and 5c 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. We 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, we have defined for the best setup 20% for the new data percentage and 70 as drift threshold.

Fig. 5d presents the results considering the variation in the cluster maximum Euclidean distance defined in the Algorithm 2 (see Section IV). The data input of the algorithm varies between 0 and 1 (such values represent the set of flow features). We have tested distances between 0.10 and 0.30, 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, we have defined the best setup with optimized system values as 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 the TRUSTED w/ HAC (the system with unsu-

(a) (b) (c) (d)

Fig. 5. TRUSTED Empirical Setup

pervised 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]). We present the results following the accuracy, precision, recall, and F1-Score metrics.

Fig. 6a presents the accuracy results from the three systems instances over the Botnet 2014 dataset. The TRUSTED w/ ARF presented the best results achieving 99% of accuracy and high stability. However, this result is due to use 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 the FIXED w/ HAC, the 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 performance of TRUSTED and FIXED systems on the green and blue lines, showed in Fig. 6a, 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. 6b shows all the metrics results. The TRUSTED w/ ARF reached 99% of performance in all metrics, being the best evaluation results. The results of the unsupervised methods show the 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. 6b. Although the 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.

To ensure the evaluation soundness besides the Botnet 2014 dataset, we have evaluated through 4 scenarios of CTU-13 dataset. Fig. 7 presents the accuracy metric over the scenarios evaluated comparing the three TRUSTED instances. In all executions presented in Figs. 7a, 7b, 7c, 7d, the TRUSTED w/ HAC proposed in this article overcomes the traditional FIXED w/ HAC strategy. Although the 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. 8 supports Fig. 7 results by presenting for all scenarios the precision, recall, and F1-Score metrics that match the difference presented by the accuracy over the datasets.

(a) Acuracy Comparison (b) General Comparison

Fig. 6. Systems Comparison with Botnet 2014 Dataset

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 retrain 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 subsection VI-C. In online the online environment, we did not supply offline samples and labels to the supervised approach.

C. Online environment

Fig. 9 presents the results of accuracy in the CTU-13 4 scenarios evaluated. As a general observation, the 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.

The TRUSTED w/ ARF did not receive an offline sample to pre-train the classifier, just got supplied with labels to retrain 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

(a) CTU4 (b) CTU5 (c) CTU10 (d) CTU11

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

(a) CTU4 (b) CTU5 (c) CTU10 (d) CTU11

Fig. 8. General Comparison CTU-13 Dataset Offline Evaluation

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 subsection VI-B). Therefore, in a non-ideal scenario (e.g., real world), the supervised approaches are unfeasible.

Although the TRUSTED w/ HAC is unsupervised, it is stable during botnet detection compared to the supervised TRUSTED w/ ARF. Except in Fig. 9c, where the TRUSTED w/ HAC accuracy constantly drops during the botnet detection, it has happened because the system did not identify any concept drifts in the data distribution. Hence, the TRUSTED w/ HAC did not adapt to detect botnet. Besides, Fig. 10 presents the TRUSTED w/ HAC mean results as worst only on Fig. 10c the CTU-10 base. However, in all other online evaluations, the unsupervised approach proposed in this article performed better than supervised.

In Fig. 9d, there is an abrupt decrease in the TRUSTED w/ HAC results near 40% of the data. This accuracy loss 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.

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 *Proceedings of the 2019 3rd International Conference on Compute and Data Analysis*, ser. ICCDA 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 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," *Journal of Inf. Security and Applications*, vol. 38, pp. 8–27, 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," *Journal of Ambient Intelligence and Humanized Computing*, 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," *Telecommunication Systems*, 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 *Proceedings of the 6th ACM Workshop on Moving Target Defense*, 2019, pp. 79–88.

(a) CTU4 (b) CTU5 (c) CTU10 (d) CTU11

Fig. 9. Accuracy Comparison CTU-13 Dataset Online Evaluation

(a) CTU4 (b) CTU5 (c) CTU10 (d) CTU11

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

[6] Ö. Gözüaçık, A. Büyükçakır, H. Bonab, and F. Can, "Unsupervised concept drift detection with a discriminative classifier," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 2365–2368.

[7] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, 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," *Journal of Advances in Computer Netws*, vol. 3, no. 1, 2015.

[9] X. Yu, X. Dong, G. Yu, Y. Qin, and D. Yue, "Data-adaptive clustering analysis for online botnet detection," in *2010 Third International Joint Conference on Computational Science and Optimization*, vol. 1. IEEE, 2010, pp. 456–460.

[10] M. Yahyazadeh and M. Abadi, "Botgrab: A negative reputation system for botnet detection," *Computers & Electrical Engineering*, vol. 41, pp. 68–85, 2015.

[11] A. Abou Daya, M. A. Salahuddin, N. Limam, and R. Boutaba, "Botchase: Graph-based bot detection using machine learning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 15–29, 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 *2014 IEEE Conference on Communications and Network Security*. IEEE, 2014, pp. 247–255.

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

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

[15] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 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 Transactions on Network and Service Management*, vol. 17, no. 1, pp. 45–59, 2020.

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

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

[19] E. R. Faria, J. a. Gama, and A. C. P. L. F. Carvalho, "Novelty detection algorithm for data streams multi-class problems," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 795–800. [Online]. Available: https://doi.org/10.1145/2480362.2480515

[20] 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. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 122–133.

[21] T. S. Sethi, M. Kantardzic, and H. Hu, "A grid density based framework for classifying streaming data in the presence of concept drift," *Journal of Intelligent Information Systems*, vol. 46, no. 1, pp. 179–211, 2016.

[22] J. Lee and F. Magoules, "Detection of concept drift for learning from stream data," in *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*. IEEE, 2012, pp. 241–245.

[23] G. Ditzler and R. Polikar, "Hellinger distance based drift detection for nonstationary environments," in *2011 IEEE symposium on computational intelligence in dynamic and uncertain environments (CIDUE)*. IEEE, 2011, pp. 41–48.

[24] L. I. Kuncheva and W. J. Faithfull, "Pca feature extraction for change detection in multidimensional unlabeled data," *IEEE transactions on neural networks and learning systems*, vol. 25, no. 1, pp. 69–80, 2013.

[25] 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 *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 935–944.

[26] B. H. Schwengber, A. Vergütz, N. Prates, and M. N. Lima, "A method aware of concept drift foronline botnet detection (to appear)," in *IEEE*

Global Communications Conference (GLOBECOM). IEEE, 2020, pp. 1–6.

[27] N. Grira, M. Crucianu, and N. Boujemaa, "Unsupervised and semi-supervised clustering: a brief survey," *A review of machine learning techniques for processing multimedia content*, vol. 1, pp. 9–16, 2004.

Bruno H. Schwengber holds B.Sc. in Computer Science from Federal University of Technology – Paraná – Brazil. He is currently a Computer Science M.Sc. student at 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 is a Ph.D. Candidate in Computer Science at 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 of the Institute of Electrical and Electronics Engineers (IEEE) and of the Brazilian Computer Society (SBC).

Nelson G. Prates Junior holds B.Sc. degree in Information Systems from Federal University of Santa Maria and M.Sc. degree in Computer Science from Federal University of Parana (UFPR), 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 is an associate professor in the computer science department of Federal University of Minas Gerais. She holds a Doctorate degree in Computer Science from the UPMC-Sorbonne University, Laboratoire d'Informatique de Paris VI (LIP6). She was in a Sabbatical Leave at Carnegie Mellon University in 2016-2017. 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.