Lifelong Autonomous Botnet Detection

Alex Medeiros de Araújo*, **Anderson Bergamini de Neira***, **Michele Nogueira***†
*Department of Informatics - Federal University of Paraná, Brazil
† Department of Computer Science - Federal University of Minas Gerais, Brazil
Emails: {amaraujo, abneira}@inf.ufpr.br, michele@dcc.ufmg.br

***Abstract*—Botnet-driven attacks have attracted attention due to their diversity, high potential to cause damage and massive data generation. Existing botnet detection solutions are usually specific to a type of attack behavior. This particularity makes attack detection challenging because it involves a high operational overhead for manually calibrating and managing a large set of solutions for different attacks and variations. Hence, this work presents LBDS, a botnet detection system that acts autonomously in dynamic environments. It relies on concept drift and AutoML, two main techniques that consider dynamic behavior on data distribution. The LBDS evaluation has followed a diverse set of attacks and protocols. Results demonstrate that the system detects botnets utilizing different detection techniques, indicating its ability to consider various aspects of data and attacks.**

***Index Terms*—AutoML, Network data analysis, Big data for cybersecurity, Concept drift.**

I. Introduction

A botnet consists of malware-infected devices remotely controlled by a botmaster. Once infected, botmasters employ these devices for different purposes, motivated mainly by financial gain [1]. Recently, the Malware-as-a-Service (MaaS) market has made it possible to monetize botnets in different ways, such as carrying out Distributed Denial of Service (DDoS) attacks, mining cryptocurrencies, extortion, and carrying out SPAM campaigns [2]. Botnet rental services have made cyber attacks more accessible, mainly for non-technical users, which is one of the main factors for increasing the frequency of attacks conducted by botnets [3].

Detecting botnets is challenging. In general, botnet detection proposals are specific to sets of attacks and require a great deal of effort from experts on analyzing high data volume and considering characteristics as network topology and the implementation of the attack [4]. Hence, there is a significant number of works on botnet detection applying machine learning techniques [5]–[7]. Although most works offer benefits, the use of these methods still requires significant manual effort and time calibrating parameters and adapting the methods to the network behavior [8]. Automating these steps has been pointed as a promising way to overcome these challenges, making the validation and deployment of machine learning-based methods reproducible and reliable [8].

This work was supported by National Council for Scientific and Technological Development (CNPq/Brazil), grants #309129/2017-6 and #432204/2018-0, by São Paulo Research Foundation (FAPESP), grant #2018/23098-0, by the Coordination for the Improvement of Higher Education Personnel (CAPES/Brazil), grants #88887.501287/2020-00 and #88887.509309/2020-00, and by the National Teaching and Research Network (RNP) by the GT-Periscope project.

Existing botnet detection techniques based on machine learning assume a static behavior from attacks. They assume that data distribution remains the same throughout the application/service lifecycle. This is one of the biggest problems faced by these solutions because of the wide variety of attack techniques [9]. An unexpected change in the behavior of the network results in a large discrepancy between the distribution of observed data and the data distribution expected by the solution. Thus, this work addresses the problem of how to effectively detect botnets with minimal human intervention in an environment characterized by frequent changes in network behavior and attack vectors.

This work presents the **L**ifelong **B**otnet **D**etection **S**ystem (LBDS), a modular system that integrates and manages different machine learning techniques with the goal of (i) designing a solution for detecting known and unknown botnets with minimal human intervention, (ii) automatically adjusting the solution parameters, eliminating the loss of effectiveness throughout its life cycle,and (iii) mitigating the need to acquire labeled datasets. LBDS selects the most appropriate set of hyperparameters for a given context. Furthermore, it considers changes in the network behavior throughout its execution to adapt the solution.

The performance evaluation aims to explore the LBDS feasibility in operating in an environment with different labels and where the network behavior evolves. For this purpose, this work evaluated the system with an attack scenario present in the CTU-13 dataset. The test scenario presents a variety of different botnet behavior and protocols. Results demonstrate that LBDS effectively detects botnets in the presence of concept drift, i.e., under the variation in data distribution. In the supervised mode, where there are labels from both normal and bot devices, the system maintains recall and prevision metrics above 90% for most of its execution. In the semi-supervised mode, with labels from normal devices only, the system maintains recall above 90%.

This paper proceeds as follows. Section II discusses related works. Section III details the proposed system. Section IV describes the performance evaluation and discusses the results. Section V presents the final remarks.

II. Related works

The literature presents several works that use machine learning techniques to detect botnets [5]. In [6], Wang and Yang offered a solution for detecting botnets in peer-to-peer networks using graphs and neural networks. They extract local graphs where the attributes of each edge help to characterize a device. These attributes are inputs for a neural network. In [7], Desai, Shi, and Suo presented a combination of supervised and unsupervised machine learning for botnet detection. Hence, supervised learning uses the output of unsupervised learning to classify devices as attackers or not. In [10], Neira, Medeiros, and Nogueira found that the same machine learning techniques are not always adequate to deal with the different types of existing attacks. Furthermore, changing machine learning techniques or reconfiguring the current technique can prevent performance degradation in the face of different attacks.

In [11], Wang, Tian, and Jia presented an approach for detecting botnets that consider changes in data distribution. The authors identify changes in concept by checking if the data distribution changed since the last tuning of the algorithm. If there is a difference, the most relevant traffic characteristics are evaluated to identify the bots in the new concept. The classification algorithm is adjusted to give greater importance to these features. In [12], Schwengber *et al.* identified changes in concept utilizing the Discriminative Drift Detector algorithm. In this algorithm, an unsupervised classifier differentiates the training from the current observed data. The proposed algorithm notifies the occurrence of a concept drift when the classifier differentiates between the two. After identifying a change in concept, data present in the new concept retrain the classifier used to identify bots. In [10], Neira, Medeiros, and Nogueira proposed an autonomous system to detect different types of botnets during their execution. The solution adapts to different scenarios and learns how to detect different types of botnets throughout its execution. Thus, utilizing AutoML, the solution selects the learning techniques.

III. The LBDS system

This section introduces LBDS, a system for Lifelong Botnet Detection. The LBDS system builds a machine learning pipeline based on the characteristics of the network, ensuring continuous detection while considering the particularities of each attack. The machine learning pipeline consists of a botnet detection technique, the most appropriate set of hyperparameters, and preprocessing techniques. The LBDS system comprises four main modules, as shown in Fig. 1. Module 1 is responsible for collecting the network traffic utilized for analysis by the other modules and extracting representations of device behavior from the raw network data. Module 2 follows a mechanism for detecting concept drift. This module is responsible for indicating when the system should replace a detection pipeline. Module 3 builds a machine learning pipeline for classifying devices best suited for a given context. Module 4 identifies bots by executing the classification pipeline built by Module 3. The following subsections detail each of the modules.

A. Data preprocessing module

The LBDS system must analyze the data disseminated among all network devices to operate correctly. The data capture module is responsible for monitoring, collecting, and making network data available to the other modules of the system. The capture module consumes traffic in two ways: real-time processing and historical datasets. In real-time processing mode, the capture module consumes live network data; thus, it is necessary to place the LBDS system at a central point where it is possible to observe the communication among all devices. In the historical dataset mode of operation, the capture module takes historical data in PCAP or CSV (Comma-separated values) format as input.

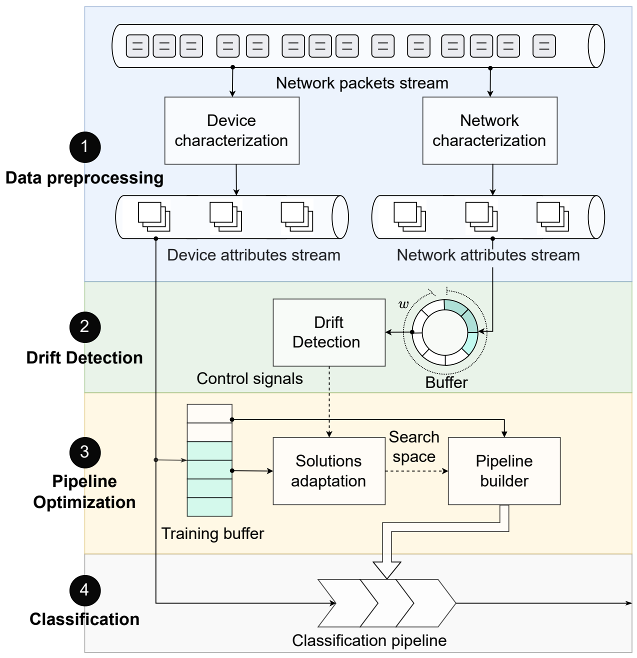

Fig. 1. The LBDS system architecture

The LBDS system segments the network traffic by time windows in the preprocessing module. This segmentation makes it possible to identify bots after capturing a portion of the total traffic, allowing the detection of botnets and the adjustment of solutions online. The time windows have two parameters, window width and the time jump size. The window width, represented as w, defines the length of the analyzed segments. The time jump size, expressed as s, defines the frequency at which the system forms the new segments.

The right choice of segmentation parameters is essential for the effectiveness and efficiency of the system implementation. A low w value results in a lower computational cost because it reduces the amount of data in each segment. However, this reduction can be accompanied by a reduction in effectiveness, as the data present in the segment may not be sufficient to capture the behavior of a device significantly. Increasing the s parameter reduces the computational cost. However, this change results in efficiency complications because too large a hop can ignore behaviors with a duration shorter than s.

Module 1 of Fig. 1 shows that the preprocessing module extracts two continuous streams of data: the device attributes stream and the network attributes stream. The device attributes stream consists of statistical information of all packets transmitted between two devices, as proposed by [13]. Table I shows the attributes utilized by the system. This work highlights the number of sent packets, the ratio of the size of the received packets, the maximum interval time in the received packet, and the ratio of the time interval of the received packets. Finally, the system does not use the payload data to preserve user privacy. LBDS utilizes this stream as input in Module 3 to construct the device classification pipeline. The network attributes stream is composed of information that characterizes network traffic. LBDS calculates the total number of sent packets, the sum of packet bytes, and the minimum, maximum, and average time between packet transmission for each time window. This second stream is consumed directly by the drift detection algorithm in Module 2.

TABLE I
EMPLOYED FEATURES

Features	Description
Tsr/Tss	Sum of received/sent packets size (bytes)
Rpc/Spc	Amount of packets received/sent
Savg	Average sent packet size
Smin	Minimum sent packet size (bytes)
Smax	Maximum sent packet size (bytes)
Svar	Variance of sent packets (bytes)
Ravg	Average received packet size
Rmin	Minimum received packet size
Rmax	Maximum received packet size (bytes)
Rvar	Variance of received packets (bytes)
SITmin	Minimum interval time in sent packages
SITmax	Maximum interval time in sent packages
SITavg	Average interval time of sent packages
RITmin	Minimum interval time in received packages
RITmax	Maximum interval time in received packages
RITavg	Average interval time of received packages

B. Drift Detection Module

Deploying a machine learning algorithm in a given environment often follows the assumption that this environment is static. Thus, data distribution that algorithms will use as input remains the same for an indefinite period. While this assumption may be valid in many applications, it may be unrealistic in some scenarios. In these scenarios, when there is a change in data distribution, the algorithm's input becomes inconsistent with the examples utilized during training, and it is necessary to update or replace the algorithm. This change in data distribution is known as concept drift [14].

Two classes of methods detect concept drifts [14]. The first class includes methods that monitor the error rates of a machine learning algorithm. If the values of performance metrics deviate from the values observed during training, data observed by the algorithm is not consistent with the training examples. The second class comprises methods that use distance measures to estimate the similarity between past and current data distributions.

The drift detection module is based on a similarity-based algorithm to detect changes in network behavior. LBDS uses a modified version of the One-Class Drift Detector (OCDD) algorithm [15]. An unsupervised concept drift detection algorithm that can be used in environments where labels for changes in data streams are not available, removing the need for label acquisition strategies in real-time. The OCDD technique uses a novelty detection algorithm to identify when a portion of the most recent network data differs from the oldest. In this case, when there is a difference, it is indicated that a concept drift has occurred.

Fig. 1, step 2, illustrates a buffer that stores the last w samples of network attributes obtained from the attribute stream, represented by S_r. During its initialization phase, the module stores samples in the buffer according to their order of arrival until it totals w elements. With the buffer filled, the stored data is standardized to values with a mean equal to 0 and a standard deviation equal to 1. The system utilizes standardized data as input for training the detector C; this detector is an instantiation of the LocalOutlierDetection, an implementation of a novelty detection algorithm present in the Scikit-learn[1] library. The drift detection module starts the detection phase after filling the buffer and training the first version of the C detector. During the detection phase, the module executes the novelty detection algorithm on the data present in the buffer. A drift is detected if the buffer contains a proportion of data classified as novelty α, greater than a threshold p. If a drift is detected, the drift detection module notifies the pipeline optimization module that the existing detection pipeline must replace; Section III-C details this process. After this signaling, the system deletes the oldest data portion from the buffer, and detector C is retrained only with the most recent data portion; this new data represents the new state of the network.

C. Pipeline Optimization Module

Fig. 1 shows the design of the optimization module: a data buffer, the Solution Adaptation Component (SAC), and the Pipeline Builder Component (PBC) (see step 3 in the figure). This module uses the data buffer to store device information made available in the device attribute stream. The buffer has a maximum size specified by the system administrator. After being notified of the occurrence of one drift, or the initial execution of the system, the pipeline optimization module starts filling the buffer with data from network devices.

After filling the buffer, the SAC component begins its execution. This component determines the set of machine learning classifiers and preprocessing techniques that the system can utilize from data in the buffer (Fig. 2). If external systems or the system administrator have enriched the buffer data with normal devices and bots labels, SAC determines that it is possible to build a solution for detecting bots with supervised machine learning techniques. In this case, the system will initiate the AutoSklearn[2] to build the solution. If data in the buffer contains unbalanced classes, SAC alleviates this problem by applying ClusterCentroids, a subsampling algorithm implemented by the Imbalanced-learn [16] library. If the training data has only normal device labels, SAC determines that the solution must use a semi-supervised learning technique. In this

[1]https://scikit-learn.org
[2]https://automl.github.io/auto-sklearn/master/

case, the LBDS system uses the positive-unlabeled learning algorithm proposed by [17].

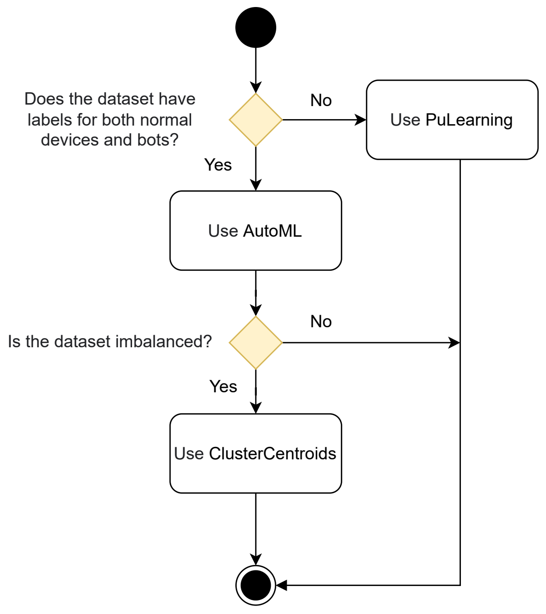
Does the dataset have labels for both normal devices and bots?
No
Use PuLearning
Yes
Use AutoML
Is the dataset imbalanced?
No
Yes
Use ClusterCentroids

Fig. 2. Search space definition

The PBC component builds a classification pipeline. This mechanism encapsulates machine learning algorithms in an executable manner for bot detection. The pipeline is composed of a set of operations applied sequentially, where each operation utilizes the result of the previous operation. The last operation in the pipeline will output the classification of a device into a bot or normal device. The intermediate steps aim to guarantee the execution of the last step without errors. An example of this type of operation is filling mandatory values expected by the classification step.

D. Classification Module

The classification module is responsible for invoking the classification pipeline. This module directs the data obtained from the device attributes stream to the input of the classification pipeline. This module is also responsible for integrating the output of LBDS with external systems.

IV. Results

This section presents the evaluation of LBDS for detecting botnets. This evaluation verifies whether the system detects behavior drifts in network traffic, mainly if drifts occur close to the beginning of the attack. By identifying drifts, the system avoids performance degradation by building a new classification pipeline. Therefore, the botnet detection pipeline is the most appropriate for current network behavior distribution, permitting the botnet detection system to adapt automatically to different attacks conducted by botnets.

This work has tested the system with different levels of label availability, deploying the system in a semi-supervised and supervised mode. Therefore, this work shows the results of two conducted experiments utilizing Scenario 10 from the CTU-13 dataset[3]. This dataset has 13 botnet traffic captures recorded at the Czech Technical University. Scenario 10 presents different attacks and detailed labels of network events. Attacks occurred on August 18, 2011, during two periods. In the first period, the dataset reports SYN Flood and ACK DDoS attacks, and in the second period, the dataset reports ICMP Flood attacks.

While the CTU-13 is a popular dataset for evaluating botnet detection techniques, the most common form of evaluation is using a batch learning approach, where a static model is built offline on the whole dataset. This makes comparisons with previous works difficult since we consider a dynamic evaluation with performance metrics changing over time.

Both experiments follow the Test-Then-Train [18] approach. Whenever the LBDS system receives a new sample, it is first utilized for the classification pipeline test and later inserted into the training buffer. This work has segmented network and device data in all experiments by a one-second window (w) and a one-second hop (s). Previous experiments have shown that for drift detection the best performance is obtained with a buffer of four minutes and a value of p equal to 0.4 [15]. As the segmentation window is one second in size, the drift detection module's buffer comprises 240 samples of network attributes. The maximum size of the training buffer present in the pipeline optimization module was also set to four minutes.

This work uses precision (Eq. 1) and recall (Eq. 2) to evaluate the system performance. These metrics are based on the number of true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN).

$$Precision = \frac{TP}{TP + FP} \quad (1) \qquad Recall = \frac{TP}{TP + FN} \quad (2)$$

A. Experiment 1

Experiment 1 consists of testing the system in semi-supervised mode. After each drift occurrence, the data present in the training buffer is enriched with normal device labels provided by the system admin. Among all normal devices active during the pipeline construction phase, 50% were assigned labels by the admin; the other 50% remained unlabeled. Fig. 3 shows the results obtained in this experiment. The gray frames (Frames A, B, C, D, and E) indicate when the system detected drifts in the network traffic data. Furthermore, the dotted red frames (Frames 1, 2, and 3) indicate when an attack occurs (the beginning and end). Frame A indicates a drift at the beginning of the capture because the system is initializing. At the beginning of the system execution, normal devices precision and recall values vary between 80% and 100% after the first training. The pipeline system correctly classifies almost all normal devices as normal devices. Because the bots are not yet active, it is impossible to calculate the bot class's precision and recall in the initial moments. Close to second 1000, the system detects a drift (Frame B). At this point, the system retrains the classification pipeline utilizing the data stored in the training buffer. After retraining, normal

[3]https://www.stratosphereips.org/datasets-ctu13

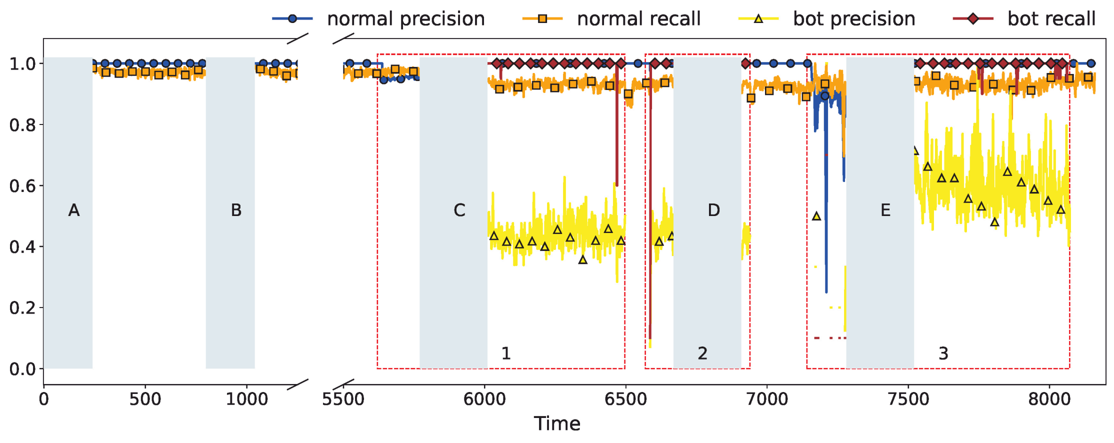

Fig. 3. Precision and recall over time for Experiment 1

device metrics results are still close to 100%. The dataset documentation does not indicate an attack at this time. Thus, this drift in Frame B is a false positive.

Fig. 3 shows that, close to second 6000, an attack is initiated (Frame 1). Notably, the pipeline built after the second drift is ineffective in detecting bots at the beginning of the attack. The bot recall metric demonstrates this for the bot class, which is zero. A few seconds after the start of the attack, the system identifies a change in network behavior and builds a new classification pipeline. After retraining, the system continues to classify non-bots with high metrics. However, the most important is that the system correctly starts to classify the bots. Precision and recall for the bot class, respectively, yellow and brown lines, are visible in the figure. After the drift (Frame C), the recall for the bot class is practically constant at 100%; thus, the system identifies all bots. However, the bot precision ranges from 40% to 80%.

Frame 2 of Fig. 3 shows the second attack in the dataset. The drift detection algorithm has correctly identified the attack, but it was impossible to significantly measure performance metrics after training the new pipeline due to its short duration. Frame 3 has the beginning of the third attack. Before the drift detection, there was a dip in the bot recall metric during the start of the attack. However, the system correctly classifies normal devices after the drift detection and construction of a new classification pipeline. As a result, the precision and recall of normal devices stay close to 100%, and the precision of bots between 35% and 90%.

B. *Experiment 2*

The second experiment consists of testing the LBDS system utilizing supervised machine learning. As supervised techniques require labels from both classes for training, tests have randomly selected 50% of bots and normal device labels. This choice aims to simulate real scenarios where obtaining labeled data is expensive.

Fig. 4 shows the results obtained in Experiment 2. The drift detection module uses the same network attribute data from

TABLE II
THE BEST MACHINE LEARNING PIPELINE PER SCENARIO IDENTIFIED BY LBDS

Frame C	**Data Preprocessor**	**Imputation**	Most frequent
		Rescaling	Robust scaler
	Features Preprocessor		SelectFwe
	Classifier		RandomForestClassifier
Frame D	**Data Preprocessor**	**Imputation**	Mean
		Rescaling	Min max
	Features Preprocessor		PolynomialFeatures
	Classifier		HistGradientBoostingClassifier
Frame E	**Data Preprocessor**	**Imputation**	Most frequent
		Rescaling	Robust scaler
	Features Preprocessor		ExtraTreesPreprocessorClassification
	Classifier		AdaBoostClassifier

Experiment 1; thus, the system detected the same drifts as the previous experiment (Frames A, B, C, D, and E). Because this experiment restricts the system to supervised techniques only, creating a classification pipeline at the beginning of Fig. 4 between Frame A and frame C was impossible as there was no bot activity.

As in the previous experiment, Frame 1 of Fig. 4 shows the beginning of the first attack. The LBDS built a new classification pipeline when the system detected a drift (Frame C). This new pipeline correctly classifies almost all the devices in the network, with all the metrics staying above 90%. The performance metrics stay above 90% after the second drift (Frame 2) when the system builds a new pipeline (Frame D). After the third attack (Frame 3), the system presents instability in performance; this is demonstrated right after the attack when the recall for the bot class approaches 10%. However, after the system builds a new pipeline (Frame E), the metrics reach close to 100% again.

Table II shows the resulting classification pipeline for every training frame. The Pipeline Builder Component selected different detection techniques for different scenarios; this indicates that the LBDS considered different aspects of an attack when choosing the best way to carry out detection.

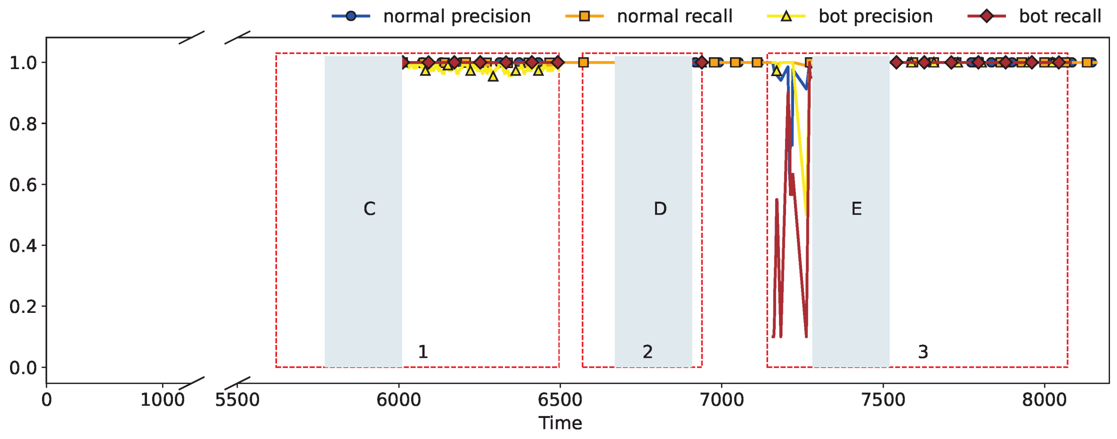

Fig. 4. Precision and recall over time for Experiment 2

C. Discussion

Experiment 1 indicates that the LBDS system utilizing semi-supervised learning could correctly classify almost all normal devices. Usually, the system cannot classify the bots during the beginning of the attacks, but there is a rise in performance metrics after retraining. This work found that the supervised mode was superior to the semi-supervised mode. The system selects the best-supervised learning technique considering the behavior of the devices to achieve this result. Finally, it is essential to note that precision and recall close to 100% is usually a signal of overfitting. However, two different strategies were utilized during the evaluations to combat overfitting. A Test-Then-Train approach was employed; and no device that appears in the training data was used for testing; therefore, the tests did not use data from devices that appear in training.

V. Conclusion

This work presented LBDS, a system for autonomous and continuous detection of botnets. The LBDS system is composed of four independent modules that interact to form an integrated system able of adapting to changes in network behavior. These modules enable the LBDS system to automatically build a classification pipeline for different types of botnets at runtime utilizing different machine learning techniques. The system evaluation followed two experiments and the results demonstrated the effectiveness of the LBDS system in detecting botnets. In addition, it was possible to observe that the system adapts by selecting and retraining detection techniques for different attacks; this indicates that LBDS was able to consider different aspects of an attack when choosing the best way to carry out detection. Future work aims to investigate different strategies to adapt an existing classification pipeline, including the possibility of retraining an existing technique or recycling a discarded pipeline, depending on the profile of network data. Furthermore, future research intends to decrease the dependence on labeled data while maintaining high-performance metrics.

References

[1] S. Traer and P. Bednar, "Motives behind DDoS attacks," in *Digital Transformation and Human Behavior*. Springer, 2021, pp. 135–147.

[2] R. Anderson, C. Barton, R. Boehme, R. Clayton, C. Ganan, T. Grasso, M. Levi, T. Moore, and M. Vasek, "Measuring the changing cost of cybercrime," in *WEIS*. Cambridge, 2019.

[3] G. Bottazzi and G. Me, "The botnet revenue model," in *SINCONF*. ACM, 2014, p. 459–465.

[4] I. Arnaldo and K. Veeramachaneni, "The holy grail of "Systems for Machine Learning"," *SIGKDD*, vol. 21, no. 2, pp. 39–47, Nov. 2019.

[5] S. N. T. Vu, M. Stege, P. I. El-Habr, J. Bang, and N. Dragoni, "A survey on botnets: Incentives, evolution, detection and current trends," *Future Internet*, vol. 13, no. 8, p. 198, 2021.

[6] Y. Yang and L. Wang, "LGANet: Local graph attention network for peer-to-peer botnet detection," in *CTISC*, 2021, pp. 31–36.

[7] M. G. Desai, Y. Shi, and K. Suo, "A hybrid approach for IoT botnet attack detection," in *IEMCON*, 2021, pp. 0590–0592.

[8] M. Feurer and F. Hutter, *Hyperparameter Optimization*. Springer, 2019, pp. 3–33.

[9] J. Á. Cid-Fuentes, C. Szabo, and K. Falkner, "An adaptive framework for the detection of novel botnets," *C&S*, vol. 79, pp. 148–161, Nov. 2018.

[10] A. B. de Neira, A. M. Araujo, and M. Nogueira, "Early botnet detection for the Internet and the Internet of things by autonomous machine learning," in *MSN*, Japan, 2020, pp. 516–523.

[11] Z. Wang, M. Tian, and C. Jia, "An active and dynamic botnet detection approach to track hidden concept drift," in *ICICS*. Springer, 2017, pp. 646–660.

[12] B. H. Schwengber, A. Vergütz, N. G. Prates, and M. Nogueira, "A method aware of concept drift for online botnet detection," in *GLOBECOM*. IEEE, 2020, pp. 1–6.

[13] D. Zhao, I. Traore, B. Sayed, W. Lu, S. Saad, A. Ghorbani, and D. Garant, "Botnet detection based on traffic behavior analysis and flow intervals," *C&S*, vol. 39, pp. 2–16, 2013.

[14] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 12, pp. 2346–2363, 2018.

[15] Ö. Gözüaçık and F. Can, "Concept learning using one-class classifiers for implicit drift detection in evolving data streams," *Artificial Intelligence Review*, vol. 54, no. 5, pp. 3725–3747, 2021.

[16] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *JMLR*, vol. 18, no. 17, pp. 1–5, 2017.

[17] C. Elkan and K. Noto, "Learning classifiers from only positive and unlabeled data," in *SIGKDD*, 2008, pp. 213–220.

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