BotChase: Graph-Based Bot Detection Using Machine Learning

Abbas Abou Daya, Mohammad A. Salahuddin, Noura Limam, and Raouf Boutaba, *Fellow, IEEE*

***Abstract*—Bot detection using machine learning (ML), with network flow-level features, has been extensively studied in the literature. However, existing flow-based approaches typically incur a high computational overhead and do not completely capture the network communication patterns, which can expose additional aspects of malicious hosts. Recently, bot detection systems that leverage communication graph analysis using ML have gained attention to overcome these limitations. A graph-based approach is rather intuitive, as graphs are true representation of network communications. In this paper, we propose BotChase, a two-phased graph-based bot detection system that leverages both unsupervised and supervised ML. The first phase prunes presumable benign hosts, while the second phase achieves bot detection with high precision. Our prototype implementation of BotChase detects multiple types of bots and exhibits robustness to zero-day attacks. It also accommodates different network topologies and is suitable for large-scale data. Compared to the state-of-the-art, BotChase outperforms an end-to-end system that employs flow-based features and performs particularly well in an online setting.**

***Index Terms*—Security management, botnet detection, machine learning.**

I. Introduction

UNDOUBTEDLY, organizations are constantly under security threats, which not only costs billions of dollars in damage and recovery, it often also detrimentally affects their reputation. A botnet-assisted attack is a widely known threat to these organizations. According to the U.S. Federal Bureau of Investigation, "Botnets caused over $9 billion in losses to U.S. victims and over $110 billion globally. Approximately 500 million computers are infected each year, translating into 18 victims per second." The most infamous attack, called Rustock, infected 1 million machines, sending up to 30 billion spam emails a day [1]. Hence, it is imperative to defend against these botnet-assisted attacks.

A botnet is a collection of bots, agents in compromised hosts, controlled by botmasters via command and control (C2) channels. A malevolent adversary controls the bots through botmaster, which could be distributed across several agents that reside within or outside the network. Hence, bots can be used for tasks ranging from distributed denial-of-service (DDoS), to massive-scale spamming, to fraud and identify theft. While bots thrive for different sinister purposes, they exhibit a similar behavioral pattern when studied up-close. The intrusion kill-chain [2] dictates the general phases a malicious agent goes through in-order to reach and infest its target.

Anomaly-based methods are widely used in bot detection [3], [4]. They first establish a baseline of normal behavior for the protected system and model a decision engine. The decision engine determines and alerts any divergence or statistical deviations from the norm as a threat. *Machine learning* (ML) [3] is an ideal technique to automatically capture the normal behavior of a system. The use of ML has boosted the scalability and accuracy of anomaly-based IDSs [4]. The most widely employed learning paradigms in ML include *supervised* and *unsupervised*. Supervised learning uses labeled training datasets to create models. It is employed to learn and identify patterns in the known training data. However, labeling is non-trivial and typically require domain experts to manually label the datasets [3]. This can be cumbersome and prone to error, even for small datasets. On the other hand, unsupervised learning uses unlabeled training datasets to create models that can discriminate between patterns in the data.

An important step prior to learning, or training a ML model, is feature extraction. These features act as discriminators for learning and inference, reduce data dimensionality, and increase the accuracy of ML models. The most commonly employed features in bot detection are flow-based (e.g., source and destination IPs, protocol, number of packets sent and/or received, etc). However, these features do not capture the topological structure of the communication graph, which can expose additional aspects of malicious hosts. In addition, flow-level models can incur a high computational overhead, and can also be evaded by tweaking behavioral characteristics e.g., by changing packet structure [5].

Graph-based features, derived from flow-level information, which reflect the true structure of communications, interactions, and behavior of hosts, are an alternate that overcome these limitations. We show that incorporating graph-based features into ML yields robustness against complex communication patterns and unknown attacks. Moreover, it allows for cross-network ML model training and inference. The major contributions of this paper are as follows.

Manuscript received June 10, 2019; revised November 8, 2019 and January 18, 2020; accepted January 22, 2020. Date of publication February 7, 2020; date of current version March 11, 2020. This work was supported in part by the Royal Bank of Canada and in part by the Natural Sciences and Engineering Research Council of Canada Collaborative Research and Development under Grant 530335. The associate editor coordinating the review of this article and approving it for publication was S. Scott-Hayward. *(Corresponding author: Noura Limam.)*

The authors are with the David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L6P7, Canada (e-mail: aaboudaya@uwaterloo.ca; mohammad.salahuddin@uwaterloo.ca; noura.limam@uwaterloo.ca; rboutaba@uwaterloo.ca).

Digital Object Identifier 10.1109/TNSM.2020.2972405

- We propose BotChase, an anomaly-based bot detection system that is protocol agnostic, robust to zero-day attacks, and suitable for large datasets.
- We employ a two-phased ML approach that leverages both supervised and unsupervised learning. The first phase filters presumable benign hosts. This is followed by the second phase on the pruned hosts, to achieve bot detection with high precision.
- We propose feature normalization (F-Norm) on top of graph-based features in BotChase and evaluate various ML techniques. Our graph-based features, inspired from the literature and derived from network flows, undergo F-Norm to overcome severe topological effects. These effects can skew bot behavior in different networks, exacerbating ML prediction. Furthermore, these features allow to combine data from different networks and promote spatial stability [6] in the ML models.
- We compare the performance of our graph-based features with flow-based features from BotMiner [7] and BClus [8] in a prototype implementation of BotChase. Furthermore, we compare BotChase with BotGM [9] and the end-to-end system proposed for BClus.
- We evaluate the BotChase prototype system in an online setting that recurrently trains and tests the ML models with new data. We also leverage the Hoeffding Adaptive Tree (HAT) [10] classifier for incremental learning. This is crucial to account for changes in network traffic and host behavior.

This paper is an extension of our preliminary work in [11]. The rest of the paper is organized as follows. In Section II, we present a background on bot detection, highlight limitations of the state-of-the-art and motivate the problem. Our system design is delineated in Section III. We evaluate the BotChase prototype in Section IV. In Section V, we conclude with a summary of our contributions and instigate future directions.

II. Background and Related Works

Botnet detection has been an active area of research that has generated a substantial body of work. Common botnet detection approaches are passive. They assume successful intrusions and focus on identifying infected hosts (bots) or detecting C2 communications, by analyzing system logs and network data, using signature- or anomaly-based techniques. Signature-based techniques have commonly been used to detect pre-computed hashes of existing malware in hosts and/or network traffic. They are also used to isolate IRC-based bots by detecting bot-like IRC nicknames [12], and to identify C2-related DNS requests by detecting C2-like domain names [13].

Metadata such as regular expressions based on packet content and target IP occurrence tuples [14] is an example of what could be employed in a signature and pattern detection algorithm. More generally, signature-based techniques have been employed to identify C2 by comparison with known C2 communication patterns extracted from observed C2 traffic, and infected hosts by comparison with static profiles and behaviours of known bots [15]. However, they can be easily subverted by unknown or modified threats, such as zero-day attacks and polymorphism [15], [16]. This undermines their suitability to detect sophisticated modern botnets.

On the other hand, anomaly-based techniques use heuristics to associate certain behaviour and/or statistical features extracted from system or network logs, with bots and/or C2 traffic. C2 occurs at the early stages of a botnet's lifecycle, thus its detection is deemed essential to prevent malicious activities. Most existing anomaly-based C2 detection techniques are based on the statistical features of packets and flows [7], [12], [17]–[27]. Works like [17], [18] are focused on specific communication protocols, such as IRC, providing narrow-scoped solutions. Whereas, BotMiner [7] is a protocol-independent solution, which assumes that bots within the same botnet are characterized by similar malicious activities and communication patterns. This assumption is an over simplification, since botnets often randomize topologies [15] and communication patterns as we observe in newer malwares, such as Mirai [28]. Other works, such as [23], [27], leverage ML and traffic-based statistical features, for detecting C2 with low error rates. However, such techniques require that all flows are compared against each other to detect C2 traffic, which incurs a high computational overhead. In addition, they are unreliable, as they can be evaded with encryption and by tweaking flow characteristics [5]. Therefore, it is evident that a non-protocol-specific, more efficient, and less evadable detection method is desired.

Anomaly-based bot detection solutions that do not focus on detecting C2 per se, but rather identify bots by observing and analyzing their activities and behaviour, address some of the aforementioned issues. Graph-based approaches, where host network activities are represented by communication graphs, extracted from network flows and host-to-host communication patterns, have been proposed in this regard [5], [9], [29]–[40]. Le *et al.* [37] present a strong case for leveraging Self-Organizing Maps (SOMs) in the context of bot detection with recall rates beyond 90%. However, SOM remains an unsupervised learning algorithm that ultimately requires manual expertise to distinguish unknown network traffic.

BotGM [9] builds host-to-host communication graphs from observed network flows, to capture communication patterns between hosts. A statistical technique, the inter-quartile method, is then used for outlier detection. Their results exhibit moderate accuracy with low false positives (FPs) based on different windowing parameters. However, BotGM generates multiple graphs for every single host. That is, for every pair of unique IPs, a graph is constructed, such that every node in the graph represents a unique 2-tuple of source and destination ports, with edges signifying the time sequence of communication. However, this entails a high overhead.

Khanchi *et al.* [41] propose botnet detection on non-stationary stream of data using incremental non-overlapping window. They propose to use a team of genetic programs to predict on records in a stream of data. These records are archived to build a data subset to further train the classifier. The true labels of the records are requested from human operators as long as the label budget is met. However, if records from minor classes are targeted for true label queries, minor

classes are promoted aggressively, reducing the performance of major classes.

Chowdhury *et al.* [39] use ML for clustering the nodes in a graph, with a focus on dimensionality and topological characterization. Their assumption is that most benign hosts will be grouped in the same cluster due to similar connection patterns, hence can be eliminated from further analysis. Such a crucial reduction in nodes effectively minimize detection overhead. However, their graph-based features are plagued by severe topological effects (see Section IV). They use statistical means and user-centric expert opinion to tag the remaining clusters as malicious or benign. However, leveraging expert opinion can be cumbersome, error prone and infeasible for large datasets. Recently, rule-based host clustering and classification [40] have been proposed, where pre-defined thresholds are used to discriminate between benign and suspicious hosts. Unfortunately, relying on static thresholds make the technique prone to evasion and less robust to ML-based outlier detection.

Big Data has received a lot of attention lately, which is also often paired with streaming. Employing ML in a streaming context [42] undoubtedly yields better result than batching. Keen statistical techniques, such as concept drifts and ADWIN windowing, help surmount the challenges facing classification in data streaming. However, retraining the ML model only when a concept drift occurs may require specific threshold tuning, which does not generalize.

Graph-based approaches using ML for bot detection are intuitive and show promising results. In this paper, we propose BotChase, an anomaly-, graph-based bot detection system, which is protocol agnostic, i.e., it detects bots regardless of the protocol. BotChase employs graph-based features in a two-phased ML approach, which is robust to zero-day attacks, spatially stable, and suitable for large datasets.

III. System Design

The BotChase system consists of three major components, as depicted in Fig. 1. These components pertain to data preparation and feature extraction, model training, and inference. In the following, we discuss these components.

A. Dataset Bootstrap

1) Flow Ingestion: The input to the system are bidirectional network flows. These flows carry the basic IP layer information required to forward packets from one network node to another. By applying reduction on these flows, we create a set T that contains 4-tuple flows $t_i = (sip_i,\ srcpkts_i,\ dip_i,\ dstpkts_i)$. sip_i is the source IP address that uniquely identifies a source host, $srcpkts_i$ quantifies the number of data packets from sip_i to dip_i, the destination host IP address. The number of destination packets, $dstpkts_i$, is the number of data packets from dip_i to sip_i.

Set A is a set of tuples that have exclusive source and destination hosts. That is, if multiple tuples have the same source and destination hosts, they are reduced to form an aggregated exclusive tuple $a_x \in A$, such that $a_x = (sip_x,\ srcpkts_x,\ dip_x,\ dstpkts_x)$. Therefore, if two tuples $t_i, t_j \in T$ have the same source and destination hosts, i.e., $sip_x = sip_i = sip_j$ and $dip_x = dip_i = dip_j$, then the number of source packets and the number of destination packets are aggregated in a_x, such that

$$srcpkts_x = \sum_{t_k \in T \ \mid \ sip_x = sip_k, dip_x = dip_k} srcpkts_k \tag{1}$$

$$dstpkts_x = \sum_{t_k \in T \ \mid \ sip_x = sip_k, dip_x = dip_k} dstpkts_k. \tag{2}$$

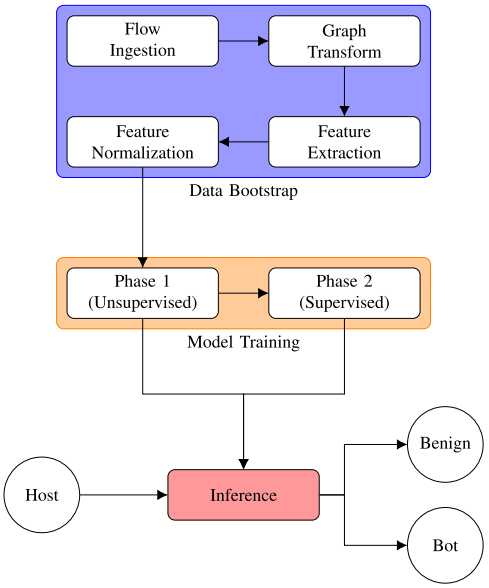

Fig. 1. Components of the BotChase bot detection system.

2) Graph Transform: The system creates a graph $G(V, E)$, where V is a set of nodes and E is a set of directed edges $e_{i,j}$ from node v_i to node v_j with weight $|e_{i,j}|$. The set of nodes V is a union of hosts from set A, such that

$$V = \bigcup_{\forall a_x \in A} (\{sip_x\} \cup \{dip_x\}). \tag{3}$$

For every $a_x \in A$, there exists directed edges $e_{i,j}$ and $e_{j,i}$ from v_i to v_j and v_j to v_i, respectively, such that $sip_x = v_i$ and $dip_x = v_j$. Therefore,

$$E = \bigcup_{\forall a_x \in A} (\{(sip_x,\ dip_x)\} \cup \{(dip_x,\ sip_x)\}). \tag{4}$$

The weights $|e_{i,j}|$ and $|e_{j,i}|$ of edges $e_{i,j}$ and $e_{j,i}$ are $srcpkts_x$ and $dstpkts_x$, respectively. Moreover, if there exists a reverse tuple $a_y \in A \mid dip_y = v_i,\ sip_y = v_j$, then $|e_{i,j}| = srcpkts_x + dstpkts_y$ and $|e_{j,i}| = dstpkts_x + srcpkts_y$.

3) Feature Extraction: BotChase creates the required *graph-based* feature set for the ML models. Features are intrinsic to the success of ML models that should represent and discriminate host behavior, especially bot behavior. We leverage the following commonly used graph-based features.

- ***In-Degree (ID)*** and ***Out-Degree (OD)***—The in-degree, $f_{i,0}$, and out-degree, $f_{i,1}$, of a node $v_i \in V$ are the number of its ingress and egress edges, respectively.

$$\mathcal{F}(e_{i,j}) = \begin{cases} 1, & \text{if } e_{i,j} \in E \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

$$f_{i,0} = \sum_{v_j \in V,\ v_i \neq v_j} \mathcal{F}\big(e_{j,i}\big) \qquad \forall v_i \in V \quad (6)$$

$$f_{i,1} = \sum_{v_j \in V,\ v_i \neq v_j} \mathcal{F}\big(e_{i,j}\big) \qquad \forall v_i \in V \quad (7)$$

These features play an important role in the network behavior of a host. Though, a higher ID for a host makes it a point of interest, it is often the case that nodes with a high ID offer a commonly demanded service. Therefore, observing ID alone may not signify malicious activity. For example, a gateway is a central point of communication in a network, but it is not necessarily a malicious endpoint. Intuitively, bots attempting to infect the network will tend to have a higher ID than benign hosts. Similarly, OD is also an intrinsic feature. Typically, in the reconnaissance stage of the intrusion kill-chain, bots attempt to survey the network. This mass surveillance can be captured via the OD.

- ***In-Degree Weight (IDW)*** and ***Out-Degree Weight (ODW)***—These features augment the ID and OD of the nodes in the graph. The in-degree weight, $f_{i,2}$, and the out-degree weight, $f_{i,3}$, of a node $v_i \in V$ is the sum of all the weights of its incoming and outgoing edges, respectively.

$$f_{i,2} = \sum_{v_j \in V,\ v_i \neq v_j,\ e_{j,i} \in E} \big|e_{j,i}\big| \qquad \forall v_i \in V \quad (8)$$

$$f_{i,3} = \sum_{v_j \in V,\ v_i \neq v_j,\ e_{i,j} \in E} \big|e_{i,j}\big| \qquad \forall v_i \in V \quad (9)$$

For a fine-grained differentiation, it is important to expose features that will eventually bring bots closer to each other, and discriminate bots from hosts. IDW and ODW features add another layer of information, further alienating the malicious hosts from the benign. Similar to ID, mass-data leeching bots will tend to expose a high IDW in the action phase of the intrusion kill-chain. Similarly, the ODW is the aggregate data packets a node has sent, which can potentially expose bots that mass-send payloads to hosts in a network.

- ***Betweenness Centrality (BC)***—The betweenness centrality of a node $v_i \in V$, inspired from social network analysis, is a measure of the number of shortest paths that pass through it, such that

$$f_{i,4} = \sum_{v_j, v_k \in V,\ v_i \neq v_j \neq v_k} \frac{\sigma_{v_j v_k}(v_i)}{\sigma_{v_j v_k}} \quad \forall v_i \in V. \quad (10)$$

where $\sigma_{v_j v_k}$ is the total number of shortest paths between node pairs $v_j,\ v_k \in V$, and $\sigma_{v_j v_k}(v_i)$ is the number of shortest paths that pass through v_i. This feature has a high computational overhead with $O(|V|.|E| + |V|^2.\log|V|)$ time complexity [43]. However, it can alienate bots early on as they attempt their first connections. This is when the bots exhibit low IDW and ODW. Thus, it would be more favorable for the shortest paths in the network to pass through the host. Likewise, when the IDW and ODW increase, the BC of a node decreases immensely, as it is less favored for being included in shortest paths.

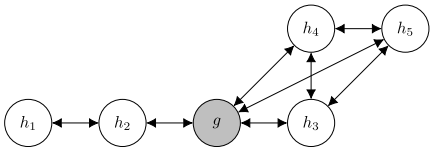

Fig. 2. Example topology of benign hosts with a gateway.

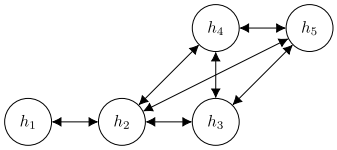

Fig. 3. Example topology of benign hosts without a gateway.

- ***Local Clustering Coefficient (LCC)***—Unlike BC, local clustering coefficient has a lower computational overhead, and it quantifies the neighborhood connectivity of a node $v_i \in V$, such that

$$f_{i,5} = \frac{\sum_{v_j, v_k \in N_i,\ v_i \neq v_j \neq v_k} \mathcal{F}\big(e_{j,k}\big)}{|N_i|(|N_i| - 1)} \quad \forall v_i \in V \quad (11)$$

where N_i is neighborhood set for v_i, $\forall v_j \in N_i \mid e_{i,j} \in E \vee e_{j,i} \in E$. LCC feature can play an important role in discriminating malicious host's behavior. Successfully infected hosts tend to exhibit a higher LCC, as bots often deploy collaborative P2P techniques, making its adjacent host pairs strongly connected.

- ***Alpha Centrality (AC)***—Alpha centrality, also inspired from social network analysis, is a feature that generalizes the centrality of a node $v_i \in V$. AC extends the Eigenvector centrality (EC), with the addition that nodes are also influenced by external sources. These external sources can be user-defined, according to their graphical analysis technique. In EC, each v_i is assigned an influence score x_i, that is iteratively exchanged with adjacent nodes. In essence, EC is the relative weight of a node in the graph, such that connections to high-scoring nodes contribute more to the score of v_i. Hence, AC is given as

$$f_{i,6} = \alpha A_i^T x_i + e_i \qquad \forall v_i \in V. \quad (12)$$

where A_i is the adjacency matrix, e_i is the external influence of node v_i, and α is influence factor that controls focus between external sources to internal influence. AC is important for the intermediate and terminal phases of the intrusion kill-chain. Early on, it may be negligible. However, as time progresses and bots perform more actions in the network, their AC will gradually increase, making it discriminative.

4) Feature Normalization (F-Norm): Topological alterations can severely affect the host's behavior and pattern of communication in the graph. For example, in Fig. 2, g acts as a gateway for host $h2$ to communicate with the rest of the network (i.e., hosts $h3$, $h4$ and $h5$). In this configuration, h_2 carries an ID of 2. In contrast, Fig. 3 shows the topology without a gateway, where h_2 communicates with other hosts in the network individually. This boosts the ID of host h_2 to 4. To alleviate this adverse effect of different network topologies,

we normalize the above *base* features to include neighborhood relativity. To control the overhead of computing these *normalized* features, the neighborhood set N_i for $v_i \in V$ is restricted to a certain depth *D*. The mean of *j* features for v_i across its neighbors $v_k \in N_i$ are computed. Each feature for v_i is then normalized by incorporating neighborhood relativity. Thus, features relative to their neighborhood mean is given as

$$\mu_{i,m} = \frac{\sum_{v_k \in N_i} f_{k,m}}{|N_i|} \qquad \forall v_i \in V,\ 0 \leq m \leq j \quad (13)$$

$$f_{i,m} = \frac{f_{i,m}}{\mu_{i,m}} \qquad \forall v_i \in V,\ 0 \leq m \leq j. \quad (14)$$

After normalizing the features (with $D = 2$) for h_2 and h_4 with gateway, their IDs change from 2 to 0.8 and 3 to 1.1, respectively. Without the gateway, their IDs change from 4 to 1.6 and 3 to 1.1, respectively. As aforementioned, normalization attempts to make hosts of the same nature look similar, making the topological alterations less severe. Similarly, in situations where network data is recorded over varying time intervals, IDW and ODW tend to increase substantially with larger time intervals. By normalizing features, the effect of time also diminishes.

B. Model Training

The model accepts graph-based features as input and learns to distinguish between malicious and benign hosts. The two learning phases in BotChase are explained below.

1) Phase 1: The first ML phase performs unsupervised learning (UL) to cluster the hosts. Generally, benign hosts exhibit *similar* behavior that can be gauged by the graph-based features. These hosts exhibit resembling patterns in data, such as sending (OD/ODW) and receiving (ID/IDW) similar number of packets [39]. Since BC, LCC and AC are directly affected by these traits, their influence can be similar for all benign hosts. Therefore, maximizing the size of the "singleton" benign cluster is crucial.

This phase not only acts as a first filter for new hosts, but also significantly reduces the training data for the second phase. If a host is clustered into the benign cluster, then it is strictly benign. However, it is important to note that a malicious host can also be incorrectly clustered into a benign cluster, adversely affecting system performance. Therefore, though the system objective is to maximize the size of the benign cluster, it is essential to *jointly* minimize the number of bots that are co-located in this cluster.

Various UL techniques can be deployed in this phase. Some of these techniques include *k*-Means, Density-Based Spatial Clustering (DBScan) and SOM.

- ***k-Means***—The *k*-Means clustering algorithm attempts to find an optimal assignment of nodes to *k* pre-determined clusters, such that the sum of the pairwise distance from the cluster mean is minimized. *k*-Means is static, it results in the same cluster composition for a given dataset across different runs of the algorithm, with the same number of clusters and iterations. Assume *k* is set to the cardinality of the label set. Idealistically, there should be a clean assignment of hosts to corresponding clusters. However, in reality, some benign hosts exhibit an outlier behavior. For example, network nodes that host webservers and public APIs will depict a huge amount of data and connections, thus impacting ID, IDW, OD and ODW. Therefore, depending on the dataset, altering *k* may adversely affect clustering performance.
- ***Density-Based Spatial Clustering (DBScan)***—Unlike *k*-Means, DBScan does not require the parameter *k*, the pre-determined number of clusters. In contrast, it computes the clusters and assignment of nodes according to a rigid set of density-based rules. DBScan requires a pair of parameters: (i) *p*, the minimum number of points required to be assigned as core points, and (ii) *e*, the minimum distance required to detect points as neighbors. DBScan classifies points as core, edge or noise, where core points must have *p* points in their neighborhood with a distance less than *e*. Otherwise, if the point is reachable via *e* distance from at least one of the core points, it is considered an edge. The remaining points are considered noise and are not clustered. That is, points are not forcefully assigned to clusters as some points may just be noise. Therefore, DBScan is capable of detecting non-linearly separable clusters.
- ***Self-Organizing Map (SOM)***—A SOM is a special purpose artificial neural network that applies competitive learning instead of error-correction. It is frequently used for dimensionality reduction and clusters similar data. However, the notion of similarity in SOM is looser than that of *k*-Means and DBScan. In SOM, neurons are *pushed* towards the data points for a certain number of iterations. It uses the best matching unit to determine the winner neuron and updates its weights accordingly. Furthermore, SOMs also apply a learning radius that affects all the other neurons, when a close-by neuron is updated. The number of neurons also play an important role in clustering. Higher number of neurons result in dispersion of nodes away from a single cluster. Importantly, the same logic applies to *k*-Means, hence the classifier with the best assignment must be selected, according to the objectives outlined in this phase.

2) Phase 2: Phase 1 separates the dataset between nodes that are inside and outside the benign cluster. All the nodes, ideally small, that reside outside the benign cluster are input to Phase 2 for further classification. Optimally, all the bots should be outside the benign cluster, regardless of whether or not they are co-located in the same cluster. Depending on the amount of hosts outside the benign cluster, the supervised learning (SL) classifiers used in this phase will exhibit different results.

The primary objective in this phase is to maximize recall. Recall is a measure of how many bots are recalled correctly, i.e., do not go unnoticed. It is proportional to the number of true positives (TPs) and inversely proportional to false negatives (FNs). Various SL classifiers can be deployed in this phase to achieve this objective, such as logistic regression (LR), support vector machine (SVM), feed-forward neural network (FNN) and decision tree (DT).

- ***Logistic Regression (LR)*** and ***Support Vector Machine (SVM)***—LR focuses on binary classification of its input,

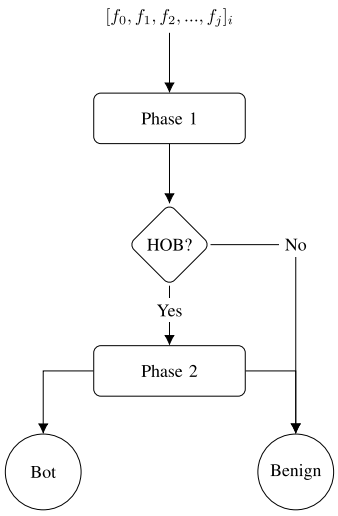

Fig. 4. Flowchart of node classification with i nodes and j features.

based on a sigmoid function. Input features are coupled with corresponding weights and fed into the function. Once a threshold p is defined, usually 0.5 for the logistic function, it establishes the differentiator between positive and negative points. Unlike LR, SVM is a non-probabilistic model for classification. It is not restricted to linearly separable datasets. There are various methods of computing SVM, including the renowned gradient-descent algorithm.

- ***Feed-forward Neural Network (FNN)***—FNNs are artificial neural networks that do not contain any cyclic dependencies. For a given feed-forward network with multiple layers, a feature vector is dispersed into the input layer, fed to the hidden layer of the network, and then to its output layer. While the input layer is constrained by the number of features exposed, the hidden and output layers are not. Every neuron may rely on a separate activation function that shapes the output. Popular activation functions for FNNs include identity, sigmoid, ReLU and binary step, among others. FNNs and the previously mentioned SL techniques are online classifiers. An online classifier is capable of incremental learning, as the weights associated with the deployed perceptrons are not static. This makes FNNs an attractive candidate for production-grade deployment.
- ***Decision Tree (DT)***—DTs rely heavily on information entropy (IE) and gain to conjure its conditional routing procedure. Generally, IE states how many bits are needed to represent certain stochastic information in the dataset. By using DT, information gain is maximized from the observed data and the taken path. After training a DT, newly observed data points can be predicted. However, unlike all the other classifiers, DTs are not online. That is, optimally retraining a DT must be done from scratch.

Recall the objective from Phase 1, i.e., minimize hosts outside the benign cluster (HOB), while maximizing bots outside the benign cluster (BOB). This results in a minimal training dataset for Phase 2. Also, it is expected that the resultant training dataset from Phase 1 would be unbalanced, with a bias towards benign hosts. This may prove problematic for LR, SVM and FNN in achieving high recall rates.

C. Inference

Once the models are trained, they are deployed in the system to perform bot detection. Ideally, the system must allow for two modes of execution: (i) model (re)training, to adjust to the dynamics of the network, and (ii) inference, i.e., for a given host predict whether or not it is a bot. In BotChase, the inference unfolds in two steps—presumable benign hosts get filtered out in Phase 1 as they get assigned to the benign cluster, while suspicious hosts that are assigned to a different cluster are further classified in Phase 2. Fig. 4 captures the inner workings of host classification. For consistency, the system must execute requests in order of observation.

IV. Experiments

We implement and evaluate the BotChase prototype bot detection system on a Hadoop cluster. In this section, we detail the experimental setup and the results of our evaluation.

A. Environment Setup

1) Hardware: The Hadoop cluster consists of a management node (2×Intel Xeon Silver 4114, 192 GB RAM), a compute node (2×Intel Xeon Gold 5120, 384 GB RAM) and four data nodes (2×Intel Xeon Silver 4114, 192 GB RAM). A 25Gbit and 10Gbit physical networks are deployed, interconnecting the nodes. The former network is primarily used for data and applications, while the latter is for administration.

2) Software: The software implementation is primarily based on Java. To ease dependency management, we incorporate Gradle [44]. JGraphT [45] graph library is used to construct the graph and extract graph-based features from network flows. Both Smile [46] and Encog [47] are used in tandem for ML. In order to support rapid prototyping, a custom in-house DataFrame (DF) library has been developed. DF conforms to the incremental streaming paradigms, data streams with well-defined source, pipelines and sinks. Furthermore, the underlying data structures are immutable, and all the basic stream-based transformations are available.

B. Dataset

The evaluation of BotChase is based on the CTU-13 [8] dataset. CTU-13 comprises of 13 different subset datasets (DS) that include captures from 7 distinct malware, performing port scanning, DDoS, click fraud, spamming, etc Every subset carries a unique network topology with a certain number of bots that leverage different protocols. CTU-13 labels indicate whether a flow is from/to botnet, background or benign. Known infected hosts are labeled as bots, while remaining hosts are tagged as benign. We leverage 12 datasets as base training data, while a single dataset, #9, is left out for testing purpose. This test dataset contains NetFlow data collected from a Neris botnet, 10 unique hosts labeled as bots, performing multiple actions including spamming, click fraud, and port scanning. We use this dataset configuration for training and testing, unless stated otherwise.

TABLE I
GRAPH TRANSFORM, BASE FEATURE EXTRACTION AND NORMALIZATION COMPUTATION

DS	GT (seconds)	Nodes	BC (hours)	FE (hours)	FE - BC (seconds)	F-Norm (seconds)
1	9	606829	24.12	24.121	3.6	11.3
2	6	441845	10.387	10.624	853.2	7.9
3	21	434489	9.463	9.713	900	13.755
4	5	185742	1.37	1.431	219.6	6.307
5	1	41548	0.057	0.06	10.8	0.556
6	3	107056	0.28	0.295	54	2.112
7	1	38081	0.021	0.022	3.6	0.488
8	13	383339	9.67	9.954	1022.4	9.617
9	10	366881	8.677	8.97	1054.8	7.879
10	7	197542	1.06	1.108	172.8	4.861
11	1	41809	0.055	0.057	7.2	0.627
12	2	94164	0.287	0.302	54	1.412
13	9	315343	3.667	3.852	666	6.824

C. Results and Discussion

1) Graph Transform, Feature Extraction & Normalization: For every subset in the CTU-13 dataset, BotChase first ingests all the network flows, creates the graph, extracts base features and normalizes them. For each dataset, Table I highlights the graph creation time, i.e., graph transform (GT), number of graph nodes ($|V|$), total runtime to extract only base BC feature and all base features (FE), and total runtime to normalize features (i.e., F-Norm).

It is evident that there is a non-linear relationship between BC and the number of nodes in the graph. BC alone consumes a large portion of the FE time. Furthermore, the inconsistent variation between GT and the number of nodes is due to the differing time windows across datasets. Also, dataset #3 has a much higher number of flows than #2, which increases the runtime of graph creation. This is primarily due to the repeated modification of exclusive flow tuples in set A. The system then normalizes the base features, and Table I depicts its total runtime with $D = 1$. Evidently, normalizing does not significantly increase the total runtime of the system, with 13.755 seconds for the most complex dataset.

We highlighted the limitations of a stand-alone supervised learning approach in [11]. Based on our evaluations, LR outperforms DT in precision, while DT shows a superior training time and robustness to unknown attacks. However, precision, training time and robustness are all crucial for our bot detection system. Can we achieve the best of all three? To investigate this, we set out to evaluate a two-phased system that employs an initial clustering phase (i.e., UL), followed by a classification phase (i.e., SL).

2) Phase 1 (UL): For the first ML phase in BotChase, we evaluate three UL techniques, namely k-Means, DBScan and SOM. However, DBScan results are inconclusive, where bots are co-located with benign hosts. DBScan is evaluated with varying minimum number of neighborhood points (minPts) and distance (ϵ). Multiple ϵ values are tested in the range of $[10^{-5}, 10^{-4}, \ldots, 10^{5}]$. Also, we infer ϵ values that correspond to the boundary of the bots themselves. We vary minPts in [1, 2,..., 25] depending on the number of bots in the aggregated training dataset. However, maximal separation of bots from benign hosts could not be achieved with the tested parameters. In essence, DBScan does not produce a single, prevalent benign cluster. On the other hand, both k-Means and SOM show appreciable results, with a learning rate of 0.7 for SOM.

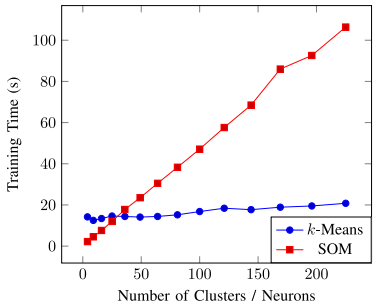

Fig. 5. Comparison of SOM and k-Means with respect to training time.

TABLE II
k-MEANS CLUSTERING WITH F-NORM

# of Clusters	HOB	HOB%	BOB	BOB%
4	5	0.0002	0	0
9	12	0.0004	0	0
16	36	0.0012	1	4
25	94	0.0033	6	24
36	170	0.0059	6	24
49	473	0.0164	8	32
64	1071	0.0371	10	40
81	1133	0.0392	10	40
100	3028	0.1049	21	87.5
121	26935	0.9327	24	96
144	27100	0.9384	24	96
169	27302	0.9454	24	96
196	27359	0.9474	24	96
225	28752	0.9956	24	96

3) Phase 2 (SL): Tables II and III highlight the evaluation metrics, including number of clusters or neurons, number of hosts outside the benign cluster (HOB), percentage of hosts outside the benign cluster relative to the total number of hosts (HOB%), number of bots outside the benign cluster (BOB), and percentage of bots relative to the total number of bots (BOB%). Recall, the dataset #9 is removed for testing, which includes 10 hosts labeled as bots and ≈366K hosts. Also, ≈3.2M hosts from the remaining datasets are used to train the classifiers. In comparison to the number of clusters for k-Means, SOM is able to alienate its first bot outside the benign cluster with a lower number of neurons (9 *vs.* 16). With 81 neurons, SOM has a recall of 92%, with k-Means achieving 42%. However, k-Means catches up with 121 clusters. Nevertheless, SOM outperforms k-Means by maximizing the number of bots isolated with a smaller number of neurons.

With a cluster size of 100, k-Means alienates 21 bots, while having an outside host sum of 3028 for the remaining non-benign clusters. In contrast, SOM removes 23 bots from the benign cluster with an outside host sum of 3675. The very next k-Means cluster size, i.e., 121, boosts HOB from 3028 to 26935, while SOM remains at a close 3894. However, k-Means isolates three extra bots, yielding 24 BOB for 26935 HOB. That is, three extra bots were detected for a ≈23K increase in HOB. Recall that our objective in this phase is to jointly minimize HOB while maximizing BOB. Therefore, SOM with 100 neurons becomes the natural choice.

TABLE III
SOM CLUSTERING WITH F-NORM

# of Neurons	HOB	HOB%	BOB	BOB%
4	10	0.0004	0	0
9	29	0.0010	1	4
16	49	0.0017	1	4
25	113	0.0039	6	24
36	286	0.0099	7	28
49	556	0.0193	8	32
64	1709	0.0592	10	40
81	3524	0.1222	23	92
100	3675	0.1274	23	92
121	3894	0.1350	23	92
144	27591	0.9647	24	96
169	27856	0.9740	24	96
196	28342	0.9912	24	96
225	28449	0.9950	24	96

TABLE IV
SUPERVISED LEARNING WITH F-NORM

Classifier	TP	FP	TN	FN	Recall	Precision
DT	10	1	366870	0	100	90.9
LR	0	0	366871	10	0	0
SVM	0	0	366871	10	0	0
FNN	0	0	366871	10	0	0

With respect to runtime, *k*-Means mostly outperforms SOM, as depicted in Fig. 5. With 100 clusters, *k*-Means took 16.8 seconds to train, in comparison to 47.1 seconds of SOM. We speculate that SOM's ever increasing training time is contributed to how it updates the surrounding neurons. As the number of neurons increase, the density of their neighborhood also increases. Eventually, more neurons will tend to be within the threshold radius. Nevertheless, with recall being our top priority, we leverage SOM as a UL classifier in Phase 1.

The training set for Phase 2 is determined by the number of hosts outside the benign cluster in Phase 1. These are the relevant hosts for this phase, as hosts that are assigned in the benign cluster never make it to Phase 2. With a 10×10 (i.e., 100 neurons) SOM and normalized features in Phase 1, the size of the dataset is significantly reduced. Therefore, we have 3675 HOB, including 23 bots, for further classification in Phase 2.

For this phase in BotChase, we evaluate four SL techniques, namely DT, LR, SVM and FNN. We use DT with Gini instance split rule algorithm, LR without regularization, and SVM with the Gaussian kernel and a soft margin penalty of 1. Moreover, NN is configured to use cross entropy as an error function and 10 hidden layers of 1000 units each. The DT classifier shows the best performance with the small dataset, as depicted in Table IV. It successfully detects *all* bots in the test dataset, with only a single FP out of the 366871 benign hosts. In contrast, all other classifiers are lackluster and unable to recall even a single bot from the dataset. We believe this is because all classifiers, except DT, rely on gradient-descent for error-correction. This implies that every single node in the dataset will affect the end-hypothesis function. Thus, with a dataset that is unbalanced, the hypothesis will be biased towards the benign hosts, which is the case for LR, SVM and FNN.

TABLE V
TRAINING TIME OF SUPERVISED CLASSIFIERS ON THE PRUNED DATASET

Classifier	Training Time (ms)
DT	88
LR	2454
SVM	864
FNN	3278

TABLE VI
SUPERVISED LEARNING WITH F-NORM ON THE BALANCED DATASET

Classifier	TP	FP	TN	FN	Recall	Precision
DT	1	0	366871	9	10	100
LR	10	9	366862	0	100	53
SVM	0	0	366871	10	0	0
FNN	0	0	366871	10	0	0

TABLE VII
SOM WITH NEWLY AGGREGATED DATASET

# of Neurons	HOB	HOB%	BOB	BOB%
100	3769	0.0011	32	94.12

Therefore, to balance the training dataset we follow a mixed sampling approach. The benign hosts are subject to downsampling to defined set size K in a range of [1K, 2K, 5K, 10K]. We then perform oversampling with replication for the bots at a 1:1 ratio with respect to the benign hosts. This provides a balance in between the dataset labels. The experiments are repeated per unique set size and the best overall outcome is then selected. Table VI shows the results with a balanced training dataset in this current scenario. SVM and FNN remain unfazed, not being able to classify a single bot. However, DT shows a significant drop in its classification performance. Since this is the pruned dataset, the number of unique data points present is minimal and the imbalance is not as significant as observed in [11]. However, balancing at this stage means balancing data points that were all outliers. This makes it more difficult for DT, which relies on information entropy, to converge to a proper bot classification state. As DT incurs a significantly less training time than LR, we proceed with the vanilla pruned dataset in the following analyses.

Table V highlights the training time for the supervised classifiers. For Phase 1, a 10×10 SOM incurs a training time of 47.1 seconds, while DT has the lowest training time of 88 milliseconds in Phase 2. Thus, the aggregate training time for both phases is $\approx$47.2 seconds, which is an 11 seconds improvement over the 58.2 seconds of stand-alone LR [11].

Using dataset #6 for testing, the robustness test harbors more hosts for training in Phase 2. Most importantly, there are more BOB, yielding a higher ratio of bots to hosts outside the benign cluster, as depicted in Table VII. The robustness results are portrayed in Table VIII. Though LR is able to recall the malicious bot while incurring only a single FP, DT exhibits perfect results on this specific test dataset. It is able to detect the previously unknown bot, as well as correctly classify all the benign hosts. Therefore, with SOM selected for Phase 1 and DT for Phase 2, the system ensures minimal training time and robustness to unknown attacks, with high recall and precision.

TABLE VIII
SUPERVISED LEARNING AGAINST PREVIOUSLY UNKNOWN BOT

Classifier	TP	FP	TN	FN	Recall	Precision
DT	1	0	107055	0	100	100
LR	1	1	107054	0	100	50
SVM	0	0	107055	1	0	0
FNN	0	0	107055	1	0	0

TABLE IX
SOM CLUSTERING WITHOUT F-NORM

# of Neurons	HOB	HOB%	BOB	BOB%
4	8	0.0003	0	0
9	39	0.0014	0	0
16	689	0.0239	0	0
25	935	0.324	0	0
36	2280	0.0790	9	36
49	3792	0.1315	11	44
64	4207	0.1459	14	56
81	6721	0.2333	15	60
100	8465	0.2940	22	88
121	12923	0.4495	24	96
144	20780	0.7248	24	96
169	22607	0.7890	24	96
196	23714	0.8280	24	96
225	42125	1.4803	24	96

4) Feature Normalization: Recall that aggregating datasets from different networks can negatively impact the base features, thus compromising system performance. Essentially, the topological structure of different networks affect the extracted graphical features, greatly skewing bot pattern and behavior. Thus, the intuition behind feature normalization is to make hosts, including bots, from different datasets look alike.

Table IX showcases the crucial depreciation of the SOM results without normalizing graph-based features. For example, with 81 neurons, SOM with and without F-Norm scores 92% and 60% on BOB, respectively. On average, the results without F-Norm have a higher HOB. This intrinsic observation signifies the lack of similarity between hosts of the same category. For example, benign hosts from different networks are not co-located due to the stark differences in their features. Conversely, with F-Norm, similarly labeled hosts are more frequently co-located, yielding better BOB and HOB. Hence, normalized graph-based features significantly improve the spatial stability of ML in BotChase.

For 100 neurons, SOM with F-Norm results in 23 BOB and 3675 HOB. Without F-Norm, it results in 22 BOB and 8465 HOB, as shown in Figures 6 and 7. Thus, for the same number of neurons, feature normalization was able to maximize BOB, while minimizing HOB. Therefore, we choose 100 neurons with F-Norm as our primary configuration for SOM.

5) Feature Engineering: It is important to gauge the significance and impact of the chosen graph-based features on bot detection. Albeit, different feature combinations may impact results, but are all features necessary? Table X shows the Pearson's feature correlation matrix for the normalized graph-based features. At a glance, we can determine that the first five features are highly correlated, with a correlation close to or greater than 0.9. Therefore, feature combinations that

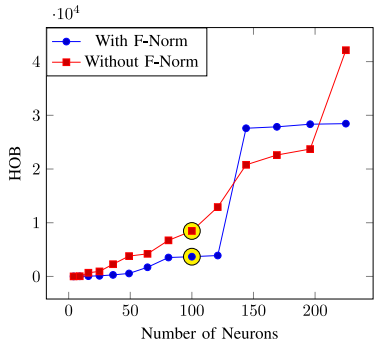

Fig. 6. Number of hosts outside the benign cluster (HOB) assigned by SOM with and without feature normalization.

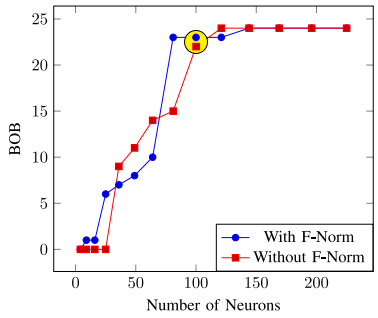

Fig. 7. Number of bots outside the benign cluster (BOB) assigned by SOM with and without feature normalization.

TABLE X
PEARSON'S FEATURE CORRELATION MATRIX WITH F-NORM

	ID	IDW	OD	ODW	BC	LCC	AC
ID	1	0.99	0.92	0.95	0.96	0.03	0.32
IDW	0.99	1	0.91	0.96	0.97	0.03	0.33
OD	0.92	0.91	1	0.89	0.90	0.08	0.37
ODW	0.95	0.96	0.89	1	0.97	0.04	0.43
BC	0.96	0.97	0.90	0.97	1	0.01	0.46
LCC	0.03	0.03	0.08	0.04	0.01	1	0.01
AC	0.32	0.33	0.37	0.43	0.46	0.01	1

TABLE XI
SOM CLUSTERING WITHOUT IDW AND ODW

# of Neurons	HOB	HOB%	BOB	BOB%
100	27404	0.958	24	96

exclude some of these features may not exacerbate classification accuracy. On the other hand, the last two features are highly uncorrelated, with LCC being the least correlated. Hence, we start with removing IDW and ODW, which decreases the benign cluster size but results higher on BOB, as shown in Table XI. However, Table XII shows the lackluster performance of the SL classifiers when we eliminate IDW and ODW features. Precision drops to 52.6% for DT from 90.9% (see Table IV). Also, LR now misclassifies two benign hosts as bots.

A weakness of the chosen features is the runtime of BC. For the first dataset, it took over 24 hours to compute BC. This will render any effort to expedite the learning process in vain. However, removing BC from the feature set adversely affects the performance of DT, but not for SOM, as depicted

TABLE XII
SUPERVISED LEARNING WITHOUT IDW AND ODW

Classifier	TP	FP	TN	FN	Recall	Precision
DT	10	9	366862	0	100	52.6
LR	0	2	366869	10	0	0
SVM	0	0	366871	10	0	0
FNN	0	0	366871	10	0	0

TABLE XIII
SOM CLUSTERING WITHOUT BC

# of Neurons	HOB	HOB%	BOB	BOB%
100	3622	0.125	23	96

TABLE XIV
SUPERVISED LEARNING WITHOUT BC

Classifier	TP	FP	TN	FN	Recall	Precision
DT	10	6	366865	0	100	62.5
LR	0	0	366869	10	0	0
SVM	0	0	366871	10	0	0
FNN	0	0	366871	10	0	0

TABLE XV
COMPARATIVE TRAINING AND TESTING DATASETS

Purpose	Dataset
Training	3, 4, 5, 7, 10, 11, 12, 13
Testing	1, 2, 6, 8, 9

in Tables XIII and XIV. SOM without BC performs identical to the use of the entire feature set. On the other hand, DT's precision is affected by the removal of BC, but it performs better than the removal of IDW and ODW from the feature set. While the precision deteriorates, *only* 6 and 9 benign hosts are misclassified out of the ≈367K hosts with the removal of BC and IDW/ODW, respectively. This reinforces the correlation matrix, i.e., these features are the most correlated. Since recall and precision are sought after metrics in BotChase, it is important to include these features for training and testing classifiers.

6) Comparative Analysis: Given the modularity of BotChase, in-place substitution of modules is possible. For example, rather than having graph-based features, the system can leverage flow-based features, while maintaining the two-phased bot detection. Therefore, we first compare the performance of our graph-based features with flow-based features from BotMiner and BClus *in* BotChase. Furthermore, we compare BotChase with the end-to-end system proposed for BClus. Finally, we provide a rough comparison against BotGM. In essence, BotGM and BotChase classify different entities in a network. The former predicts upon graphs of host-to-host communication, while BotChase classifies hosts based on a time frame. For a fair comparison, we reselect the training and testing datasets. This conforms to the selection in [8], where the test dataset contains multiple bot types and different network topologies. The dataset selection of these comparisons is depicted in Table XV.

BotMiner aggregates flows based on their source IP, protocol, destination IP and its corresponding port. These aggregated flows, called C-Flows, are processed in time epoch that lasts up to a full day of flow capture. After flows are aggregated, 52 features are extracted by first mapping every C-Flow into a discrete sample distribution of four random variables: (i) total number of packets sent and received in a flow, (ii) average number of bytes per packet, (iii) total number of flows per hour, and (iv) average number of bytes per second. These random variables are then binned into 13 slots according to pre-defined percentiles. Through this technique, every variable is converted into a vector of 13 elements, totaling 52 features per C-Flow.

TABLE XVI
SUPERVISED LEARNING WITH BOTMINER FEATURES WITHOUT F-NORM

Classifier	TP	FP	TN	FN	RCL	PRC
DT	0	1	2550094	34966	0	0
LR	7	36	2550059	34959	0.02	16.28
SVM	0	0	2550095	34966	0	0
FNN	0	0	2550095	34966	0	0

BClus undertakes a similar clustering approach by grouping flows into instances. These instances are identified by *unique* source IPs in a certain time window. Each instance is represented using 7 features: (i) source IP address, (ii) number of distinct source ports, (iii) number of distinct destination ports, (iv) number of distinct destination IPs, (v) total number of flows, (vi) total number of bytes, and (vii) total number of packets. These instances are then clustered using Expectation Maximization (EM). More features are then extracted from the clusters themselves to label clusters. These include: (i) total number of instances, (ii) total number of flows, (iii) number of distinct source IP addresses, and (iv) the average and standard deviation amount of distinct source ports, distinct destination IPs, distinct destination ports, number of flows, number of bytes, and number of packets. Hence, every cluster exhibits 15 features, which are then used by JRip (RIPPER), a propositional rule learner. After training, JRip is capable of classifying each cluster as malicious or benign. Ground truth label of clusters is determined through a bot flow threshold that is varied to find the best JRip model.

BotGM uses graph-based outlier detection to detect suspicious flow patterns. It starts off with extracting events, i.e., converting flows into a key-value entry. The key represents the source and destination IPs, while the value represents the source and destination ports. Then a sequence is extracted that tracks the source and destination port variations of two unique IPs. A directed graph is extracted from these variations, with vertices representing a port 2-tuple. The aggregate graphs are then mined for outlier detection. BotGM uses the graph edit distance to gauge how different a graph is from another. The inter-quartile method is then used to detect outliers.

- ***BotMiner Flow-based vs. Graph-based Features***—We start with the aforementioned flow-based features from BotMiner in BotChase. Table XVI showcases the outcome of classifying flows using BotMiner features, where only LR is able to detect a few malicious flows, misclassifying the majority of benign and malicious flows. To compare, we convert our host classification into flow classification in Table XVII. With a recall (RCL) of 0.02% and a precision (PRC) of 16.28%, BotMiner features

TABLE XVII
FLOW-BASED SUPERVISED LEARNING

Classifier	TP	FP	TN	FN	RCL	PRC
DT	211452	1037	20145929	47776	81.57	99.51
LR	67006	59657	20087309	192222	25.85	52.9
SVM	0	0	20146966	259228	0	0
FNN	0	0	20146966	259228	0	0

perform poorly against the graph-based features. The latter scores 81.57% and 99.51% on recall and precision, respectively. However, in comparison to host classification (see Table XXI), the precision is significantly higher as the flows originating from the identified FP hosts were relatively minimal. Likewise, the different number of flows per host may result in a lower or higher recall rate. While LR and DT highlight similar host classification results, DT is the more favorable flow classifier as it does not misclassify prominent benign hosts.

- ***BClus Flow-based vs. Graph-based Features***—Implementing BClus features in BotChase was an incremental process. Alongside choosing the optimal number of EM clusters, F-Norm has a major impact on the results. Unlike BotMiner, BClus strictly classifies instances pertaining to unique source IPs. An instance becomes a full representation of host behavior, using a large time window that fits the entire test dataset. As depicted in Table XVIII, our preliminary implementation of BClus without F-Norm has a zero recall rate across all the trained supervised classifiers. Therefore, to improve BClus we modify F-Norm to process all the hosts. Recall that BClus extracts features for source IPs only, thus features of destination IPs are missing from our data pipeline. The data pipeline only consists of hosts that have had their features extracted based on previous aggregations. Hence, a direct application of F-Norm, as implemented in BotChase, results in missing data elements for hosts that are present in the graph but not in the data pipeline. Therefore, we first naïvely modify F-Norm to handle non-existent data points with a zero vector. This improves the results of LR, which now captures a single bot out of 14, while the remaining classifiers still perform poorly, as depicted in Table XIX. This comes with no surprise, as a zero vector still affects the relative values of the host features. Finally, we transform BClus to account for both source and destination IPs as instances. This solves the issue with F-Norm, since all unique IPs in the network are mapped into a corresponding data point and host node in the graph. Using this two-way analysis, Table XX shows an appreciable improvement over the former iterations. DT manages a jump from 0% to 64.29% in recall and 81.82% in precision. However, even after the improvements to BClus features, it significantly under performs our graph-based features. Table XXI showcases the performance of the graph-based features on the new dataset selection. It exhibits convincing results for both DT and LR, with high rates of 80% and 85.71%

TABLE XVIII
SUPERVISED LEARNING WITH BCLUS FEATURES AND WITHOUT F-NORM

Classifier	TP	FP	TN	FN	Recall	Precision
DT	0	10	1651678	14	0	0
LR	1	3	1651685	13	7.14	25
SVM	0	0	1651688	14	0	0
FNN	0	0	1651688	14	0	0

TABLE XIX
SUPERVISED LEARNING WITH BCLUS FEATURES AND MODIFIED F-NORM

Classifier	TP	FP	TN	FN	Recall	Precision
DT	0	1	1651687	14	0	0
LR	1	4	1651684	13	7.14	20
SVM	0	0	1651688	14	0	0
FNN	0	0	1651688	14	0	0

TABLE XX
SUPERVISED LEARNING WITH BCLUS FEATURES AND TWO-WAY F-NORM

Classifier	TP	FP	TN	FN	Recall	Precision
DT	9	2	1905934	5	64.29	81.82
LR	7	8	1905928	7	50	46.67
SVM	0	0	1905936	14	0	0
FNN	0	0	1905936	14	0	0

TABLE XXI
SUPERVISED LEARNING WITH F-NORM

Classifier	TP	FP	TN	FN	Recall	Precision
DT	12	3	1905933	2	85.71	80
LR	12	3	1905933	2	85.71	80
SVM	0	0	1905936	14	0	0
FNN	0	0	1905936	14	0	0

on recall and precision, respectively. Interestingly, DT and LR have similar performance on host classification yet different on flow classification. Although both classifiers agree metrics-wise, the underlying sets of hosts tagged as bot or benign are different. Hence, it is possible to combine the classifiers into a single decision making entity. A simple rule to boost our classification results could be to flag a host as a bot if at least one classifier concurs. While this can increase the recall rate, precision is expected to decline as FPs are aggregated across classifiers.

- ***BClus Hybrid vs. Graph-Based Features***—So far, we have experimented with both BClus flow-based and BotChase graph-based features. How would these features fair together in the same ML model? With 6 flow-based and 7 graph-based features, every unique host in the network has 13 features to be processed by the system. In this experiment, we resort to BotChase as our base architecture, only changing the amount of features computed per unique source IP. Table XXII shows the result of the corresponding analysis. When compared to the graph-based baseline, LR is able to detect an additional bot, while incurring 6 additional FPs. This boosts the recall rate to 92.86%, while bringing down

TABLE XXII
SUPERVISED LEARNING WITH BCLUS HYBRID FEATURES AND F-NORM

Classifier	TP	FP	TN	FN	Recall	Precision
DT	12	3	1905933	2	85.71	80
LR	13	9	1905927	1	92.86	59.09
SVM	0	0	1905936	14	0	0
FNN	0	0	1905936	14	0	0

TABLE XXIII
BCLUS END-TO-END RESULTS

EM Clusters	Training Time (s)	TP	FP	TN	FN	RCL	PRC
12	423.9	14	219	1651469	0	100	6

TABLE XXIV
ACCURACY OF BOTGM VS BOTCHASE

Algorithm	Scenario ID (DS)				
	1	2	6	8	9
BotGM	0.91	0.78	0.95	0.89	0.83
BotChase	0.98	0.97	0.94	0.84	0.99

the precision to an unimpressive 59.09%. On the other hand, the metrics for DT did not change.

- ***BClus End-to-End vs. BotChase***—As part of our comparative analysis, we also implement the entire BClus approach and perform classification on their pre-defined selection of datasets. After optimizing the number of EM clusters (i.e., 12) and JRip folds, BClus performs a record 100% recall on the test datasets, with an extremely poor precision of 6%. Additionally, Table XXIII shows a high cost of 423.9 seconds to solely train EM. BClus significantly under performs BotChase that has a minimal 21.9 seconds in total training time, and $\geq$80% on recall and precision.
- ***BotGM vs. BotChase***—As the final comparison, we compare BotChase to BotGM. Both systems attempt to identify malicious and outlier behavior using graph methodologies. In [9], BotGM shows an impressive accuracy of up to 95% on one of the test datasets. Can BotChase do better? We perform a leave-one-out approach, targeting one test dataset and taking the rest for training. The results are depicted in Table XXIV. BotChase scores an aggregate high of 99% with a low of 84%. In contrast, BotGM shows a marginal improvement for scenarios 6 and 8 but lags behind in the remaining scenarios. We speculate that the traffic from the benign hosts overshadow that of the malware, which could be an issue for graph-based features, but not for BotGM's host-to-host traffic patterning.

7) Ensemble Learning: When multiple classifiers are used together for prediction, they are categorized under ensemble learning. Earlier, we have shown that DT and LR classify 12 out of the 14 bots with 3 FPs incurred in Table XXI. However, once we translate the results into flows in Table XVII, the numbers are completely different. This shows that the classifiers have successfully predicted a different set of bots. In essence, having them both act as a single classifier conservatively (i.e., bot detected, if inferred by at least one model) should result in a higher recall rate. We experimented with both of our two successful classifiers working in tandem, DT and LR.

TABLE XXV
ENSEMBLE LEARNING WITH GRAPH-BASED FEATURES

Classifier	TP	FP	TN	FN	Recall	Precision
DT + LR	13	6	1905930	1	92.86	68.42

TABLE XXVI
ONLINE SUPERVISED LEARNING

i	Time (mins)	TP	FP	TN	FN	Ing. (%)	RCL	PRC
1	5	0	0	1905936	14	0.00	0.00	0.00
2	10	0	0	1905936	14	0.00	0.00	0.00
3	15	0	21	1905915	14	1.10	0.00	0.00
4	20	0	22	1905914	14	2.24	0.00	0.00
5	25	0	23	1905913	14	2.28	0.00	0.00
6	30	8	19	1905917	6	3.61	57.14	29.63
7	35	8	19	1905917	6	3.61	57.14	29.63
8	40	8	19	1905917	6	3.61	57.14	29.63
9	45	8	19	1905917	6	3.62	57.14	29.63
10	50	8	19	1905917	6	3.62	57.14	29.63
11	55	8	3	1905933	6	4.26	57.14	72.73
12	60	0	14	1905922	14	5.07	0.00	0.00
13	65	1	5	1905931	13	5.87	7.14	16.67
14	70	1	3	1905933	13	7.15	7.14	25.00
15	75	12	5	1905931	2	10.32	85.71	70.59
16	80	12	2	1905934	2	10.73	85.71	85.71
17	85	12	2	1905934	2	11.86	85.71	85.71
18	90	12	2	1905934	2	13.53	85.71	85.71
19	95	12	2	1905934	2	15.81	85.71	85.71
20	100	12	2	1905934	2	16.72	85.71	85.71

The results in Table XXV reflect our hypothesis. The ensemble is able to correctly predict 13 bots, boosting recall to 92.86%. However, given the conservative approach, an FP incurred in any classifier will be accounted for. In this case, the classifiers on their own predict different hosts as FPs, resulting in an aggregate of 6 FPs. While this approach brings down the precision of the second phase to 68.52%, it enables us to successfully classify one more bot.

8) Analysis in an Online Setting: While our infrastructure fully supports streaming, our initial evaluation of BotChase was carried out on the entire CTU-13 dataset as a single data batch. However, it will be crucial to constantly retrain the ML models to account for change in network traffic patterns and host behavior. This will indeed be fundamental in realizing autonomic security management [48]. Most ML models are trained offline, and retrained from scratch. When this proves to be computationally intensive and time consuming, it prohibits the aspects of online deployment. The ability to retrain the model as new data becomes available, is fundamental to accommodate ML models' boundary changes after deployment.

To evaluate BotChase in an online setting, we iteratively retrain the ML models with new data and test the models. The training and testing datasets are inferred from Table XV. We assess the different ML models as we increase the amount of flows ingested into the system, from the aggregated training dataset. Therefore, a time window (w) of 5 minutes spawns N time windows, where N is the number of training sub-datasets. In the current dataset, a 5 minute time window is equivalent to 40 minutes of ingested data. Furthermore, the smallest training dataset has 15 minutes of flows, while the largest has 4011

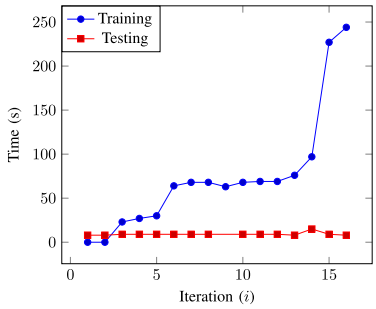

Fig. 8. The variation of the training and testing time as time progresses.

minutes. This is due to the nature of network flows. For example, the number of flows captured in one minute of a DDoS attack will outweigh that of an idle network. Therefore, the percentage of flows ingested into the system will increase in a non-linear manner. Also, it is ideal for the time window to be smaller than the smallest dataset.

With $w = 5$ and the same aggregated test dataset to assess the different ML models, we expose the elapsed time (t), percentage of flows ingested online (Ing.), and the classification metrics in Table XXVI.

- ***At t = 5 mins***, only a few flows are ingested into the system. There are a total of 9,697,049 flows in the dataset, while the percentage of ingested flows is minuscule, hence it shows as 0.00%. Since DT is the second phase classifier, having only benign data points does not suffice. Therefore, when the system detects this edge case, it defaults to classifying all data points as benign.
- ***At t = 15 mins***, 1.1% of flows are now ingested into the system. Given the early condition of hosts, it is expected to have high false alarms. In this case, the system results in 21 FPs and 14 FNs. The first bot flows appear at this time.
- ***At t = 30 mins***, we reach 3.61% of ingested flows. The number of TPs detected improves from 0 to 8, leading to the first model that is able to detect malicious hosts. Hence, it takes exactly 15 minutes for the system to exhibit its initial TPs, after the first bot flows start to appear. At this point, a baseline of malicious behavior is formed, matching the classification system's bot profile.
- ***At t = 60 mins***, the model's performance declines. The amount of flows ingested is 5.07%, while the model incurs 14 FPs and FNs. At this stage, the malicious hosts have camouflaged their features through benign communication and exhibit benign host-like behavior.
- ***At t = 75 mins***, the system reaches a state which is close to that of ingesting the full training dataset. At only 10.32%, it is able to detect 12 out of 14 bots. This achieves a recall of 85.71%. However, there are 2 additional FPs, resulting in a 70.59% precision.
- ***At t = 80 mins***, the system reaches its best outcome with only a tenth of the data ingested. Interestingly, one FP is shaved off the standard system metrics. As aforementioned, having more or less training data points may alter the constructed bot and benign profiles. A bot may initially behave like a benign host. Once more flows are ingested, a bot behavior can become more prominent and anomalous to that of benign. However, with further flows, the bot may also be able to disguise itself as benign. Since BotChase depends on graph-based features, an additional flow either adds weight to an existing edge or creates one. This will effectively change the neighbouring malicious and benign host features used in training, thus skewing performance.

TABLE XXVII
ONLINE SUPERVISED LEARNING USING HAT

i	Time (mins)	TP	FP	TN	FN	Ing. (%)	RCL	PRC
1	5	0	0	1905936	14	0.00	0.00	0.00
2	10	0	0	1905936	14	0.00	0.00	0.00
3	15	0	0	1905936	14	1.10	0.00	0.00
4	20	0	0	1905936	14	2.24	0.00	0.00
5	25	0	0	1905936	14	2.28	0.00	0.00
6	30	0	0	1905936	14	3.61	0.00	0.00
7	35	0	0	1905936	14	3.61	0.00	0.00
8	40	0	44	1905892	14	3.61	0.00	0.00
9	45	0	40	1905896	14	3.62	0.00	0.00
10	50	1	20	1905916	13	3.62	7.14	4.76
11	55	7	25	1905911	7	4.26	50.00	21.88
12	60	7	18	1905918	7	5.07	50.00	28.00
13	65	7	20	1905916	7	5.87	50.00	25.92
14	70	9	36	1905900	5	7.15	64.29	20.00
15	75	9	36	1905900	5	10.32	64.29	20.00
16	80	9	24	1905912	5	10.73	64.29	27.27
17	85	9	13	1905923	5	11.86	64.29	42.85
18	90	13	6	1905930	1	13.53	92.85	68.4
19	95	13	3	1905933	1	15.81	92.85	81.25
20	100	13	3	1905933	1	16.72	92.85	81.25

Fig. 8 showcases the different training and testing times observed as the time window progresses. Since we are dealing with flows in this scenario, the training time incorporates the time needed to extract the features from the flows. This requires building the graph incrementally with the given flows and extracting the features of every node. The plot takes on a positive slope for the training time, while the testing time remains at a plateau of around 9 seconds. At 75 minutes, the training time jumps to 227 seconds, slightly incrementing to 244 seconds at 80 minutes. Even at this optimal mark, the training time remains $< w$. This implies that the system is ready for classification prior to the processing of the next window interval.

Some variations in performance are observed beyond $t = 100$ minutes. This is because over-fitting can occur that contributes to additional FPs and FNs. The system reaches a steady state at t = 270 minutes, when only 37.22% of the flows are ingested. The sooner an online system becomes effective, the better. The amount of ingested flows and required training time dictates the total system overhead over time. The second phase supervised classifier used is DT, and it uses the ID3 algorithm [49]. This specific algorithm does not permit incremental learning, thus the tree is reinitialized at every epoch. Obviously, this is not efficient.

Therefore, we further experiment with a very fast decision tree (VFDT) variation, the HAT. This tree algorithm can be trained on-the-fly as new data becomes available. It maintains old branches, making sure to prune them as they become obsolete. An internal naïve bayes selector is used when the

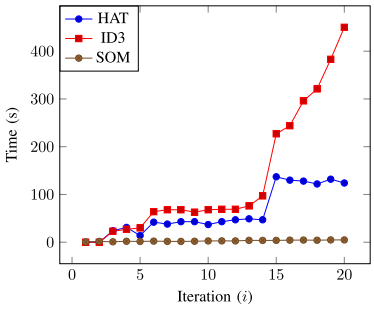

Fig. 9. The variation of the training time of HAT, ID3, and SOM.

tree leaves have the same values but different labels. Using MoA's HAT [50], we set out to experiment with BotChase using the Hoeffding algorithm. The results are depicted in Table XXVII.

We ran the algorithm with the same settings as that of the former experiment.

- ***At t = 5 mins***, HAT should be very similar to ID3 as both trees have the same exact data. This is exposed with minimal flow ingestion, and zero recall and precision.
- ***At t = 40 mins***, 3.61% of flows are now ingested into the system. This is the first epoch at which the system attempts to positively label hosts. However, this results in 44 FPs.
- ***At t = 55 mins***, we reach 4.26% of ingested flows. At this point, the system is able to successfully detect 7 out of the 14 bots. Compared to ID3, the system maintains its momentum to converge to a state that is capable of detecting bots. While the number of FPs have decreased, 20 still remain.
- ***At t = 70 mins***, the system is able to successfully detect 2 more bots, incurring alot more FPs along the way.
- ***At t = 90 mins***, the model becomes capable of capturing most of the bots, misclassifying only one. The amount of flows ingested is now 13.53%, The model still incurs a few FPs, only 1 more FP than that of the former algorithm.
- ***At t = 95 mins***, the system reaches a steady state. The recall rate does not change, but the precision increases to 81.25%. Throughout this progression, SOM maintains an average training time of 4 seconds.

Fig. 9 shows the stark difference in training time once an incremental classifier is deployed. Retraining from scratch results in an ever increasing training time. However, leveraging an incremental model allows the training time to only be restrained to new data. In general, HAT takes longer to achieve a good system state for detection, but will do so incrementally without retraining from scratch. There is a prominent compromise between convergence speed, training time, and efficacy of the model. When training time is paramount, one would favor HAT over ID3. Moreover, HAT seems more stable over time, as we see some dips in performance across intermediate iterations for ID3. With only ≈14% of the flows ingested and ≈2 minutes of training time incurred at every iteration, BotChase proves suitable for deployment in an online classification setting.

V. Conclusion

In this paper, we propose BotChase, a system that is capable of efficiently transforming network flows into an aggregated graph model. It leverages two ML phases to differentiate bots from benign hosts. Using SOM, the first phase establishes an acceptable compromise between maximizing the benign cluster and alienating the malicious bots. Furthermore, the results of the second phase favor DT, showcasing high TPs and low FPs. Without F-Norm, the results of the SOM were exacerbated, isolating less bots and decreasing the size of the benign cluster.

BotChase is also able to detect bots that rely on different protocols, proves robust against unknown attacks and cross-network ML model training and inference. Flow-based features employed in BotChase under perform in comparison to graph-based features. BotChase also outperforms an end-to-end system that employs flow-based features, and performs well against the graph-based BotGM system. In an online setting, BotChase leverages HAT for incremental learning to process data on-the-fly. While the model takes longer to converge, it exhibits superior classification performance in its final state. Further tuning the classifiers, exploring advanced ensemble learning and feature engineering, and extending F-Norm to higher degrees are candidates for future research.

References

[1] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, "Measuring pay-per-install: The commoditization of malware distribution," in *Proc. USENIX Security*, 2011, p. 13.

[2] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Inf. Warfare Security Res.*, vol. 1, no. 1, p. 80, 2011.

[3] R. Boutaba *et al.*, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *J. Internet Services Appl.*, vol. 9, no. 1, pp. 1–99, 2018.

[4] G. Creech and J. Hu, "A semantic approach to host-based intrusion detection systems using contiguous and discontiguous system call patterns," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 807–819, Apr. 2014.

[5] B. Venkatesh, S. H. Choudhury, S. Nagaraja, and N. Balakrishnan, "BotSpot: Fast graph based identification of structured P2P bots," *J. Comput. Virol. Hacking Techn.*, vol. 11, no. 4, pp. 247–261, 2015.

[6] Y. Jin *et al.*, "A modular machine learning system for flow-level traffic classification in large networks," *ACM Trans. Knowl. Disc. Data*, vol. 6, no. 1, p. 4, 2012.

[7] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering analysis of network traffic for protocol-and structure-independent botnet detection," in *Proc. USENIX Security*, 2008, pp. 139–154.

[8] 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.

[9] S. Lagraa, J. François, A. Lahmadi, M. Miner, C. A. Hammerschmidt, and R. State, "BotGM: Unsupervised graph mining to detect botnets in traffic flows," in *Proc. IEEE CSNet*, 2017, pp. 1–8.

[10] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *Proc. Int. Symp. Intell. Data Anal.*, 2009, pp. 249–260.

[11] A. A. Daya, M. Salahuddin, N. Limam, and R. Boutaba, "A graph-based machine learning approach for bot detection," in *Proc. IFIP/IEEE Symp. Integr. Netw. Manag. (IM)*, 2019, pp. 144–152.

[12] J. Goebel and T. Holz, "Rishi: Identify bot contaminated hosts by IRC nickname evaluation," in *Proc. HotBots*, vol. 7, 2007, p. 8.

[13] A. Ramachandran, N. Feamster, and D. Dagon, "Revealing botnet membership using DNSBL counter-intelligence," in *Proc. Steps Reducing Unwanted Traffic Internet*, vol. 6, 2006, pp. 49–54.

[14] M. Hagan, B. Kang, K. McLaughlin, and S. Sezer, "Peer based tracking using multi-tuple indexing for network traffic analysis and malware detection," in *Proc. IEEE 16th Annu. Conf. Privacy Security Trust (PST)*, 2018, pp. 1–5.
[15] S. Khattak, N. R. Ramay, K. R. Khan, A. A. Syed, and S. A. Khayam, "A taxonomy of botnet behavior, detection, and defense," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 898–924, 2nd Quart., 2014.
[16] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Comput. Netw.*, vol. 31, no. 8, pp. 805–822, 1999.
[17] J. R. Binkley and S. Singh, "An algorithm for anomaly-based botnet detection," in *Proc. Steps Reducing Unwanted Traffic Internet*, vol. 6, 2006, p. 7.
[18] A. Karasaridis, B. Rexroad, and D. Hoeflin, "Wide-scale botnet detection and characterization," in *Proc. HotBots*, vol. 7, 2007, p. 7.
[19] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, *Botnet Detection Countering the Largest Security Threat*, vol. 36, W. Lee, C. Wang, and D. Dagon, Eds. Boston, MA, USA: Springer, Jan. 2008, pp. 1–24. [Online]. Available: https://link.springer.com/chapter/10.1007/978-0-387-68768-1_1#citeas
[20] R. Villamarín-Salomón and J. C. Brustoloni, "Identifying botnets using anomaly detection techniques applied to DNS traffic," in *Proc. IEEE Consum. Commun. Netw. Conf.*, 2008, pp. 476–481.
[21] W. Lu, M. Tavallaee, G. Rammidi, and A. A. Ghorbani, "BotCop: An online botnet traffic classifier," in *Proc. IEEE Commun. Netw. Serveys Res.*, 2009, pp. 70–77.
[22] H. R. Zeidanloo, A. B. Manaf, P. Vahdani, F. Tabatabaei, and M. Zamani, "Botnet detection based on traffic monitoring," in *Proc. Int. Conf. Netw. Inf. Technol.*, 2010, pp. 97–101.
[23] S. Saad *et al.*, "Detecting P2P botnets through network behavior analysis and machine learning," in *Proc. IEEE 9th Annu. Int. Conf. Privacy Security Trust*, 2011, pp. 174–180.
[24] J. Zhang, R. Perdisci, W. Lee, U. Sarfraz, and X. Luo, "Detecting stealthy P2P botnets using statistical traffic fingerprints," in *Proc. IEEE/IFIP 41st Int. Conf. Depend. Syst. Netw. (DSN)*, 2011, pp. 121–132.
[25] W. Lu, G. Rammidi, and A. A. Ghorbani, "Clustering botnet communication traffic based on n-gram feature selection," *Comput. Commun.*, vol. 34, no. 3, pp. 502–514, 2011.
[26] H. Choi and H. Lee, "Identifying botnets by capturing group activities in DNS traffic," *Comput. Netw.*, vol. 56, no. 1, pp. 20–33, 2012.
[27] D. Zhao *et al.*, "Botnet detection based on traffic behavior analysis and flow intervals," *Comput. Security*, vol. 39, pp. 2–16, Nov. 2013.
[28] M. Antonakakis *et al.*, "Understanding the mirai botnet," in *Proc. USENIX Security*, 2017, pp. 1093–1110.
[29] M. P. Collins and M. K. Reiter, "Hit-list worm detection and bot identification in large networks using protocol graphs," in *Proc. Int. Conf. Recent Adv. Intrusion Detection*, 2007, pp. 276–295.
[30] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov, "BotGrep: Finding P2P bots with structured graph analysis," in *Proc. USENIX Security*, vol. 10, 2010, pp. 95–110.
[31] J. François, S. Wang, W. Bronzi, R. State, and T. Engel, "BotCloud: Detecting botnets using MapReduce," in *Proc. IEEE Int. Workshop Inf. Forensics Security*, 2011, pp. 1–6.
[32] H. Hang, X. Wei, M. Faloutsos, and T. Eliassi-Rad, "Entelecheia: Detecting P2P botnets in their waiting stage," in *Proc. IEEE/IFIP Netw.*, 2013, pp. 1–9.
[33] K. Henderson *et al.*, "RoLX: Structural role extraction & mining in large graphs," in *Proc. ACM Knowl. Disc. Data Min.*, 2012, pp. 1231–1239.
[34] Q. Ding, N. Katenka, P. Barford, E. Kolaczyk, and M. Crovella, "Intrusion as (anti) social communication: Characterization and detection," in *Proc. ACM Knowl. Disc. Data Min.*, 2012, pp. 886–894.
[35] J. François, S. Wang, R. State, and T. Engel, "BotTrack: Tracking botnets using NetFlow and PageRank," in *Proc. Int. Conf. Res. Netw.*, 2011, pp. 1–14.
[36] P. Jaikumar and A. C. Kak, "A graph-theoretic framework for isolating botnets in a network," *Security Commun. Netw.*, vol. 8, no. 16, pp. 2605–2623, 2015.
[37] 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. Comput. Intell.*, 2016, pp. 1–7.
[38] D. Zhuang and J. M. Chang, "PeerHunter: Detecting peer-to-peer botnets through community behavior analysis," in *Proc. IEEE Conf. Depend. Secure Comput.*, 2017, pp. 493–500.
[39] S. Chowdhury *et al.*, "Botnet detection using graph-based feature clustering," *J. Big Data*, vol. 4, no. 1, p. 14, 2017.
[40] D. Zhuang and J. M. Chang, "Enhanced peerhunter: Detecting peer-to-peer botnets through network-flow level community behavior analysis," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 6, pp. 1485–1500, Jun. 2019.
[41] 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.
[42] P. Mulinka and P. Casas, "Stream-based machine learning for network security and anomaly detection," in *Proc. ACM Workshop Big Data Anal. ML Data Commun. Netw.*, 2018, pp. 1–7.
[43] U. Brandes, "A faster algorithm for betweenness centrality," *J. Math. Sociol.*, vol. 25, no. 2, pp. 163–177, 2001.
[44] H. Dockter *et al.* (2007). *Gradle Build Tool*. [Online]. Available: https://www.gradle.org
[45] B. Naveh *et al.* (2013). *JGraphT*. [Online]. Available: https://jgrapht.org
[46] H. Li *et al.* (2016). *Statistical Machine Intelligence and Learning Engine*. [Online]. Available: https://haifengl.github.io/smile
[47] J. Heaton *et al.* (2013). *Encog Machine Learning Framework*. [Online]. Available: https://www.heatonresearch.com/encog
[48] S. Ayoubi *et al.*, "Machine learning for cognitive network management," *IEEE Commun. Mag.*, vol. 56, no. 1, pp. 158–165, Jan. 2018.
[49] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.
[50] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive online analysis," *J. Mach. Learn. Res.*, vol. 11, pp. 1601–1604, May 2010.

Abbas Abou Daya received the B.Eng. degree in electrical and computer engineering from the American University of Beirut, and the M.Math. degree in computer science from the University of Waterloo. He is a Senior Software Engineer with Arista Networks. His current research interests involve machine learning, cybersecurity, and networks and systems.

Mohammad A. Salahuddin received the Ph.D. degree in computer science from Western Michigan University in 2014. He is a Research Assistant Professor of computer science with the University of Waterloo. His current research interests include the Internet of Things, content delivery networks, network softwarization, cloud computing, and cognitive network management. He serves as a TPC Member for international conferences and a reviewer for various journals and magazines.

Noura Limam received the M.Sc. and Ph.D. degrees in computer science from University Pierre and Marie Curie, Paris VI, in 2002 and 2007, respectively. She is currently a Research Assistant Professor of computer science with the University of Waterloo. Her contributions are in the area of network and service management. Her current research interests are in network softwarization and cognitive network management. She is on the Technical Program Committees and Organization Committees of several IEEE conferences.

Raouf Boutaba (Fellow, IEEE) received the M.Sc. and Ph.D. degrees in computer science from the University Pierre and Marie Curie, Paris, in 1990 and 1994, respectively. He is currently a Professor of computer science with the University of Waterloo, Canada. His research interests include resource and service management in networks and distributed systems. He received several best paper awards and recognitions, including the Premiers Research Excellence Award, the IEEE ComSoc Hal Sobol, Fred W. Ellersick, Joe LociCero, Dan Stokesbury, Salah Aidarous Awards, and the IEEE Canada McNaughton Gold Medal. He is the Founding Editor-in-Chief of the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT from 2007 to 2010 and the editorial boards of other journals. He is a fellow of the Engineering Institute of Canada, and the Canadian Academy of Engineering.