Botnet Fingerprinting: a Frequency Distributions Scheme for Lightweight Bot Detection

Agathe Blaise, Mathieu Bouet, Vania Conan, Stefano Secci

Botnet Fingerprinting: a Frequency Distributions Scheme for Lightweight Bot Detection

Agathe Blaise, Mathieu Bouet, Vania Conan, Stefano Secci, *Senior Member, IEEE*

***Abstract*—Efficient bot detection is a crucial security matter and widely explored in the past years. Recent approaches supplant flow-based detection techniques and exploit graph-based features, incurring however in scalability issues, with high time and space complexity. Bots exhibit specific communication patterns: they use particular protocols, contact specific domains, hence can be identified by analyzing their communication with the outside. A way we follow to simplify the communication graph and avoid scalability issues is looking at frequency distributions of protocol attributes capturing the specificity of botnets behaviour. We propose a bot detection technique named *BotFP*, for *BotFingerPrinting*, which acts by (i) characterizing hosts behaviour with attribute frequency distribution signatures, (ii) learning benign hosts and bots behaviours through either clustering or supervised Machine Learning (ML), and (iii) classifying new hosts either as bots or benign ones, using distances to labelled clusters or relying on a ML algorithm. We validate *BotFP* on the CTU-13 dataset, which contains 13 scenarios of bot infections, connecting to a Command-and-Control (C&C) channel and launching malicious actions such as port scanning or Denial-of-Service (DDoS) attacks. Compared to state-of-the-art techniques, we show that *BotFP* is more lightweight, can handle large amounts of data, and shows better accuracy.**

***Index Terms*—Network security, data analysis, bot detection.**

I. Introduction

Back in September 2019, the French cyber police freed over 850,000 computers from a botnet named Retadup [2]. The worm spread through malicious email attachments, then installed cryptomining software on infected machines. Over nearly a million of infected hosts mined Minero cryptocurrency, reaping a huge amount of money – that is often the first reason for attackers to handle a botnet. Retadup is also suspected of being used in several ransomware attacks and data thefts. At the end of 2019, hackers also mass-scan for Docker vulnerability (Docker admin port TCP/2376) to mine Monero cryptocurrency [3]. 2019 saw an increase up to 55% of IoT malware attacks like Retadup [4]. Hence quickly detecting botnets is a major concern.

The word "botnet" comes from the combination of "robot" and "network". In this display, the attackers infect and control thousands of machines, then send them malicious commands to execute, like infecting, attacking or scanning other hosts. This large zombie network is then a major vector of large-scale attacks such as phishing DDoS, trojans, spams, etc. To communicate with bots, cybercriminals use Command-and-Control (C&C) channels implemented in different ways (the most popular ones are IRC, HTTP, P2P and Telnet [5]).

Botnet early detection is crucial to limit harms as soon as possible. However bots often mimic normal traffic and hide their payload characteristic by encryption. Recently they are also more likely to use HTTP rather than IRC to be confounded with classic web traffic. Also, HTTP being a widely-used protocol, firewalls seldom block it, contrary to IRC [6]. Furthermore, dynamic ports and runtime protocol changes enable botnets to bypass signature-based firewalls and intrusion detection systems (IDS). For robust detection systems, several flow-based botnet detection approaches [7], [8], [9], [10] were recently proposed, working without packet payload information. Differently than flow-based detection, other recently proposed botnet detection approaches consist in characterizing and analyzing relationships between hosts in the network, with techniques commonly referred to as graph-based anomaly detection [11], [12], [13]. However these techniques suffer from high time and space complexity, as they need to compute complex features over very large graphs.

In this paper, we propose a lightweight bot detection technique named *BotFP* that builds signatures modelling the behaviours of hosts in a network. These signatures reflect the communication pattern of each host, to highlight the differences between normal hosts and bots. In particular, we account for the fact that a botnet performs various kinds of actions; one can simultaneously infect and scan other hosts, perform click fraud, launch DDoS attacks, actions that can be qualified by finely analyzing IP addresses, TCP and UDP port numbers and ICMP types and codes. Then, we aim at accurately defining what constitutes bot and normal communications based on the signatures of labelled host; we propose a clustering variant (*BotFP-Clus*) - classifying new hosts based on their distances to labelled clusters - and a supervised machine learning variant (*BotFP-ML*) - for which we evaluate three different supervised ML classification approaches.

For our evaluation, we use the CTU-13 bot traffic dataset [14], containing 13 scenarios of different botnet samples. On each scenario a specific malware sample is executed, which used several protocols and performed different actions. We first learn from a training set what constitute normal communications and malicious behaviours, based on the distribution of IP addresses and port numbers used by hosts. We compare our two approaches, *BotFP-Clus* and *BotFP-ML*, in terms of bot detection performances and complexity. We

A. Blaise is with Thales, Gennevilliers, and Sorbonne Université, CNRS LIP6, Paris, France. Email: agathe.blaise@lip6.fr.

M. Bouet and V. Conan are with Thales, Gennevillers, France. Email: {name.surname}@thalesgroup.com

S. Secci is with Cnam, Cedric, 75003 Paris, France. Email: stefano.secci@cnam.fr

A preliminary version of this paper was presented at IEEE/IFIP NOMS 2020 [1].

demonstrate that the former enables to detect all bots with a better recall and a reduced complexity. Then, we show that, while having a comparable or lower time complexity than state of the art bot detection techniques, we outperform them all with a recall from 84% to 100% and a precision from 75% to 93%, depending on the method. We also show that using an adaptive quantification based on the volume of traffic enhances the results. For the sake of reproducibility and further research, we made the source code publicly available at [15].

The paper is structured as follows. Section II addresses related work in the field of bot detection and positions our work with respect to the state of the art. Section III introduces the dataset and metrics we used. Section IV presents the data processing methodology, as well as our rationale about looking at per-host fingerprints to describe host communications. In Section V, we introduce two signature-based algorithms to detect bots, namely *BotFP-Clus*, based on a clustering technique, and *BotFP-ML*, composed of various supervised machine learning techniques; while Section IV describes to the computation of attribute distribution signatures, in Section V we introduce two different techniques based on the aforementioned signatures. In Section VI, we numerically evaluate the proposals, discussing the performance in terms of precision and recall. In Section VII, we qualify the space and time complexity of *BotFP*, and we compare it to other recent bot detection methods. Finally, Section VIII concludes this paper.

II. Related work

Considering the importance of the matter, an extensive number of works exist in the field of bot detection. While traditional approaches rely on statistical and machine learning approaches over per-flow features, recently studied graph-based approaches analyze the relations between several hosts of a network.

A. Flow-based techniques

Flow-based techniques work by removing the packet payload and inspecting the packet header only [16]. Let us classify them as follows.

1) Statistical methods: BotHunter [7] aims to recognize the infection and coordination dialog that occurs during a successful malware infection. A similar approach, BotSniffer [8], focuses on the detection of C&C channels which are essential to a botnet. Therefore it exploits the underlying spatio-temporal correlation and similarity property of botnet C&C (horizontal correlation). The C&C server uses to contact every bot at the same time, then each of them uses to undertake some malicious actions following the C&C commands; these behaviours can be observed simultaneously in a network to spot a C&C channel, thus an underlying botnet.

BotHunter and BotSniffer perform their evaluation on their own honeynet or on traces authors built by executing malware binaries. However these traces are not publicly available and [9] highlighted the lack of suitable comparisons for botnet detection algorithms due to the lack of public botnet datasets. Hence they propose a labeled botnet dataset including botnet, normal and background traffic. In addition, the authors present two methods to identify botnets in these traces. The first one named BClus is a botnet detection approach. It creates models of known botnet behaviour by computing features per source IP address, then it uses them to detect similar traffic on the network. The second one named CAMNEP is a Network Behaviour Analysis system that combines various state-of-the art anomaly detection methods, such as MINDS, Xu, Lakhina volume and Lakhina entropy [17].

2) Machine learning methods: include artificial neural network, support vector machines (SVM), k-nearest neighbor (k-NN), decision trees and clustering. They can be divided into subcategories known as: *supervised*, *unsupervised* and *hybrid* techniques. *Supervised* ones learn from a labelled dataset what constitutes either normal traffic or attacks – there exists different techniques such as SVM-based classifiers, rule-based classifiers and ensemble-learning detectors [18]. Due to its excellent generalization performance, Support Vector Machines (SVM) are mainly used in many security applications [19], [20]. *Unsupervised* approaches learn by themselves what is normal or abnormal – among them, [21] proposes an unsupervised learning based ML solution to identifying known and unknown anomalies in IoT, more especially with auto-encoders; [11] also proposes an unsupervised approach, identifying the most dissimilar graphs. *Hybrid* approaches benefit from only a small part of labelled traffic, meant to be enough to learn from, as proposed in [22].

3) Other methods: use various entropy measures. For instance, [23] proposes a technique to detect large-scale anomalies in the network traffic, by measuring the deviation between the profiles of normal traffic and incoming flow records. [10] proposes a behavioral botnet detection method using Markov Chains to model the different states in the C&C channel. The proposed method is trained and evaluated using the CTU-13 dataset, and gives a F1-measure of 92% and a false positive rate of 0.05%. The authors in [24] focus on detecting bot infected machines at an entreprise level, by considering the complete DNS activity of a host per hour. They used an extensive set of features computed over campus DNS network traffic, and as a result identified suspicious DNS connections to detect infected machines.

However, flow-based techniques may miss some communication patterns between hosts that are quite specific to a botnet. Furthermore, working on a per-flow or per-host basis may incur a high computational overhead.

B. Graph-based techniques

Graph-based approaches [25] aim to model the relations between several hosts of a network. They are studied for various situations, for example to detect P2P bots [26], [27] or to recognize DNS traffic from malicious domains [28].

In [12], the authors distinguish between several kinds of traffic and make group of flows from: (i) the most frequent 11 destination port numbers used by TCP and UDP, (ii) all other TCP/UDP destination port numbers, and (iii) ICMP flows. They use plain and derived features for each of these categories, then they train three unsupervised learning algorithms on normal traffic with these features. As a result, with k-NN

they achieve over 91% detection rate with around 5% false positive rate.

BotGM [11] proposes an unsupervised graph mining technique to identify abnormal communication patterns and label them as botnets. The authors first construct a graph sequence of ports for each pair of source and destination IP addresses, then they compare each graph between them using the Graph-Edit Distance (GED). As a result, they reach a very good accuracy between 78% and 95%. However this technique is very costly as the GED is computed once for each pair of graph and its computation is known to be NP-complete.

The authors in [13] model network communications as graphs, where hosts are edges and communications between hosts vertices. They compute graph-based features such as In-Degree and Out-Degree and diverse centrality measures. They use a hybrid learning method and test various ML techniques to achieve a good detection rate. However this technique incurs in a high computational overhead as features are computed over a large communication graph, e.g., used by shortest paths algorithms computed for centrality measures.

Other graph-based detection methods [29], [30] seem promising, but their complexity is often high, NP-complete as for [30] and [11], or cubic for [13] (see Sect. VII).

C. Our contribution

Let us position our contribution with respect to the described related work. Our solution falls in the group of flow-based bot detection algorithms; *BotFP* computes statistics in a per-flow basis, yet considering a different flow definition such that a flow is identified only by the source IP address. In addition, contrary to most flow-based techniques, our solution models communications of hosts to other internal hosts and to the outside. Therefore it presents the advantages of a graph-based technique, analyzing the communications of an host with the outside. But compared to them it is more lightweight as it simply consists in multiple two-vector comparisons for *BotFP-Clus*, and in ML algorithms for *BotFP-ML*.

III. Dataset and evaluation indicators

In this section we first describe the dataset we leverage on for identifying characteristics inherent to botnets. We then introduce the metrics we used to assess the performance of our algorithm.

A. Dataset

We used the publicly available CTU-13 dataset [9] containing 13 scenarios of bot infections. Different botnet malware samples are executed in a virtual network to mimic the behaviour of an infection that is spreading. Each scenario contains between 294 and 508 hosts, including 1 to 10 bots, and there are 4923 hosts in total. Table I below draws the main characteristics for each scenario; it describes if hosts used IRC, P2P or HTTP protocols, if they sent spam, did Click-Fraud (CF), port scanning (PS) or DDoS attacks. This dataset is widely used for the already discussed recent bot detection methods [11], [12], [13].

Id	Duration (hrs)	#Bots	Bot	Activity
1*	6.15	1	Neris	IRC, SPAM, CF
2*	4.21	1	Neris	IRC, SPAM, CF
3	66.85	1	Rbot	IRC, PS
4	4.21	1	Rbot	IRC, DDoS
5	11.63	1	Virut	SPAM, PS
6*	2.18	1	Menti	PS
7	0.38	1	Sogou	HTTP
8*	19.5	1	Murlo	PS
9*	5.18	10	Neris	IRC, SPAM, CF, PS
10	4.75	10	Rbot	IRC, DDoS
11	0.26	3	Rbot	IRC, DDoS
12	1.21	3	NSIS.ay	IRC, P2P
13	16.36	1	Virut	HTTP, SPAM, PS

TABLE I: Characteristics of the botnet scenarios. The scenarios included in the test set are marked by the symbol *.

To evaluate the performances of our bot detection method, we used scenarios 1, 2, 6, 8 and 9 for the test set, and other scenarios for the training set, as recommended by the authors of the CTU-13 dataset [9]. Note that the bot species are different in the training and test sets, i.e., we do not test our algorithm on the same bot malware that we learned from.

The CTU-13 dataset is widely used for bot detection and contains malware samples from botnets still spreading, but we also found interesting to explore latest datasets from the stratosphere project [31]. In particular, the IoT-23 dataset [32] contains 23 samples of IoT network traffic, each one being either malicious or benign. In addition, the Malware Capture Facility Project [33] contains 342 long-term botnet captures, captured from 2015 until now. However, traces from these datasets are not labelled on a per-flow level, and in addition they contain either captures from botnet or from benign hosts, but no mixed captures needed to run the algorithm. Therefore, the application of our algorithm to IoT-23 is not pertinent, in particular its learning phase which normally trains on flows both from bots and benign hosts and labelled as such. Nevertheless, we report in the supplementary materials a visual comparison between three scanning events, one from each dataset, showing similarities and differences. In the following, we focus on the CTU-13 dataset because it has labelled flows and is used as reference dataset by many existing methods at the state of the art we can so compare to.

B. Evaluation metrics

A confusion matrix is a table often used to evaluate the performance of a classification model [34]. The basic terms are the following (expressed as whole numbers and not rates): True Positive (TP) is the number of bots correctly classified; True Negative (TN) is the number of benign hosts correctly classified; False Positive (FP) is the number of benign hosts incorrectly classified; False Negative (FN) is the number of bots incorrectly classified.

This is a list of rates that are often computed from the confusion matrix for a binary classifier:

Accuracy, computed as $ACC = \frac{TP+TN}{TP+TN+FP+FN}$, shows the percentage of true detection over total hosts. A high accuracy is required. However a bias may be introduced with an unbalanced dataset like the CTU-13 dataset with few botnet activity. Therefore we need to consider other metrics too.

Precision, computed as $P = \frac{TP}{TP+FP}$, refers to the ratio of incorrectly classified benign hosts versus all the benign hosts. A high P value is desirable.

Recall, computed as $R = \frac{TP}{TP+FN}$, also known as false alarm rate, refers to the ratio of incorrectly classified benign hosts versus all the benign hosts. A high R value is desirable.

F1-score, computed as $F1 = 2 \cdot \frac{P \cdot R}{P+R}$, also known as false alarm rate, refers to the ratio of incorrectly classified benign hosts versus all the benign hosts. A high $F1$ value is desirable.

IV. Bots Fingerprints

In this section we first describe some typical behaviours of bots, then we introduce our bot detection technique *BotFP*, detailing the different processing steps, including the flow records collection, the bot fingerprints computation and the signatures formatting.

A. Preliminary example

Fig. 1 gives an example of dissimilar histograms for a benign host and a bot, for three attributes namely $\text{Sport}_{\text{TCP}}$, $\text{Dport}_{\text{UDP}}$ and Dip_{UDP}; in this example, histograms are made of 32 regular bins (each bin aggregating multiple attribute values), and are normalized with respect to the number of packets. $\text{Sport}_{\text{TCP}}$ for the benign host are in the range [49152, 61000] and [1025, 5000] for the bot, which indicates a first difference in the ephemeral ports thus the OS (all bots from the dataset display this characteristic). Looking to $\text{Dport}_{\text{UDP}}$ does not show any anomaly, as both hosts show a high usage of UDP/53 which runs DNS. We thus expect to find one single IP address corresponding to the DNS server in Dip_{UDP}, but we found a multitude of them for the bot. Thus, it is not a classic DNS usage, but in fact an attempt of port scanning. This example illustrates why it is important to not only compare attributes one to one, but also to take into account correlations among attributes.

B. Methodology

The primary goal of our bot anomaly detection algorithm, BotFP, is to label bots as such, avoiding false positives. Let *Sip*, *Dip*, *Sport* and *Dport* represent respectively the source and the destination IP addresses, the source and the destination port numbers, of a layer-4 flow. Fig. 2 depicts the *BotFP* steps, through a trace example.

1) **Flow records collection**: flow records are first collected to form a dataset. We split the dataset into two distinct sets: one for training and one for testing.
2) **Host network** *Sip* **filtering and grouping**: from flow records, we select the ones whose *Sip* is in the host network and group them by such addresses.
3) **Quantification** (attribute frequency distributions computation): signatures of each host, denoted σ_{Sip}, are defined as the concatenation of the normalized frequency distributions of each attribute. TCP, UDP and ICMP flows are characterized separately to better take into account each protocol specificity.
4) **Offline training**: this phase consists in learning from the training set what constitutes either malicious or benign host signatures. Different methods can be used to do so, including clustering algorithms, supervised learning algorithms or neural networks. We further describe two approaches we propose, namely *BotFP-Clus* and *BotFP-ML*, in Section V. This step is optional and does not apply in case of an unsupervised learning algorithm.
5) **Online classification** (distances computation): finally we classify hosts from the test set either as benign or bot, through a learning algorithm consistent with the previous step. We compute the distance between one labelled host from the training set and one host to classify from the test set.

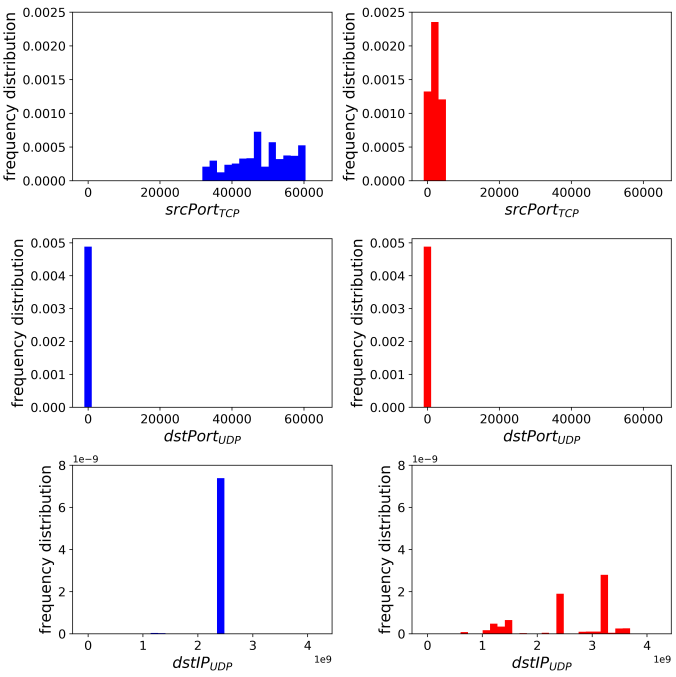


Fig. 1: Histograms showing the frequency distributions of attributes ($\text{Sport}_{\text{TCP}}$, $\text{Dport}_{\text{UDP}}$, and Dip_{UDP} respectively) for a benign host in blue/left (`147.32.84.17` from scenario #1) and a bot in red/right (`147.32.84.165` from scenario #1).

Table II defines the key parameters we use, as well as the notations for the variables of the algorithm.

Notation	Definition
m	Minimum number of packets per host for one protocol
b	Number of intervals (bins) in the frequency distributions
ϵ	Density in the clustering algorithm
a_i^j	Frequency distribution of attribute i for host j
σ_j	Signature of host j

TABLE II: Notations.

C. Flow records collection and formatting

Flow records are first collected to form a dataset (**step 1** in Fig. 2). We split the dataset into two distinct sets: one for training and one for testing. We name the training set as $\mathcal{T}$ and the test set as $\mathcal{E}$. We select only flows whose *Sip* belongs to the host network prefix and group them by such addresses as shown in Fig. 2 (**step 2**). As we search for internal bots,

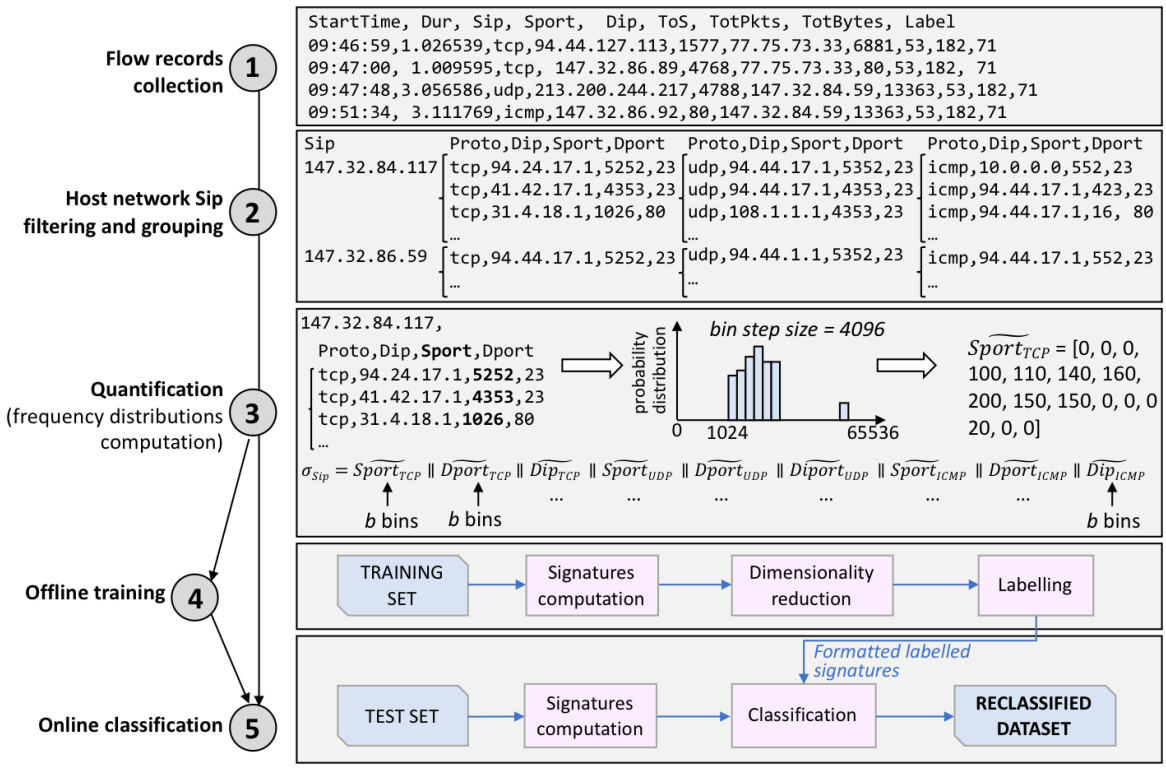


Fig. 2: Description of the processing steps of our solution. We first select flow records (**step 1**) that are in the host network and group them by such addresses (**step 2**). Signatures σ_{Sip} of each host are defined as the concatenation of the frequency distributions of each attribute (**step 3**). The training phase consists in learning what constitutes either malicious or benign signatures hosts noted σ_{Sip} (**step 4**). Finally we classify hosts from the test set, with a supervised learning algorithm that considers their distance to labelled hosts (**step 5**).

we exclude source IP addresses belonging to external Internet networks.

D. Quantification (attribute frequency distributions)

To characterize the host behavior, let A be the set of attributes used to characterize it. In this work, we consider 9 attributes in total, discriminating between TCP, UDP and ICMP packets, as follows (following CTU-13 notations): $\text{Sport}_{\text{TCP}}$, $\text{Dport}_{\text{TCP}}$, Dip_{TCP}, $\text{Sport}_{\text{UDP}}$, $\text{Dport}_{\text{UDP}}$, Dip_{UDP}, $\text{Type}_{\text{ICMP}}$, $\text{Code}_{\text{ICMP}}$ and Dip_{ICMP} [1].

Let a_i^j denote the attribute vector for attribute i and host j, representing the attribute frequency distribution, i.e., the ratio of packets received for attribute i over its attribute range. More precisely, each attribute vector contains b bins, where $a_i^j[k]$ is the value of the k^{th} bin of attribute i for host j. For each attribute, a bin aggregates the attribute occurrences over the possible attribute range (e.g., many successive port numbers grouped together in a bin) available for the specific attribute (e.g., TCP source port), in a way that depends on a bin aggregation policy as detailed hereafter.

In practice, to avoid statistically negligible attributes to influence the detection logic, it makes sense to set attribute vectors with a too low number of TCP packets exiting a host j to null values, i.e., $a_i^j[k] = 0\ \forall k$ and for all TCP-type attributes i. Let m denote such a minimum number of packets threshold, that we later numerically assess.

Let σ_j denote the signature of node j – keeping in mind that a host is uniquely identified by its Sip (we use σ_{Sip} instead of σ_j in the figures). It is built as the concatenation of all its attribute vectors; it can then be expressed as:

$$\sigma_j = \Big\|_{i=1}^{|A|} a_i^j = a_1^j \| a_2^j \| \cdots \| a_{|A|}^j \tag{1}$$

where $\|$ represents the concatenation operator between vectors. The result of the concatenation is therefore one single array σ_j of $|A| \times b$ entries.

1) Quantification technique: Let us further clarify how the attribute frequency distributions can be aggregated in a set of bins. To compute the attribute vector, b bins are used to cover the attribute range, say $[0, max]$; e.g., for source and destination port numbers, max is equal to 65536, and for destination IP addresses it is equal to 2^{32}. It makes sense to set b as a power of 2, as port numbers and IP addresses are typically organized into ranges of powers of 2 (e.g., reserved ports are in [0, 1023] and ephemeral ports in [49152, 65536], while IPv4 addresses are denoted by 4 Bytes).

We consider two different ways to aggregate bins.

Regular bins: attribute range intervals are uniformly distributed, of a fixed bin width set to max/b. Fig. 2 (**step 3**) shows an example of attribute frequency histogram for attribute $\text{Sport}_{\text{TCP}}$: the attribute range corresponds to the possible

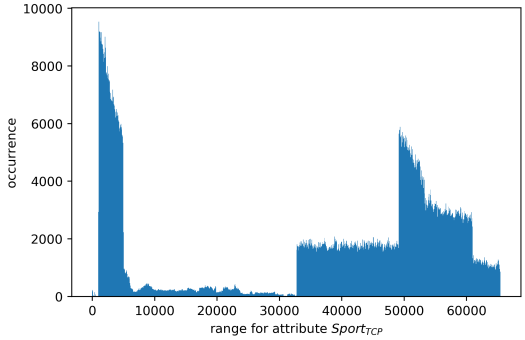
[1]Note that in the CTU-13 dataset, the used notation for the ICMP type is $\text{Sport}_{\text{ICMP}}$ and for the ICMP code is $\text{Dport}_{\text{ICMP}}$.

TCP source port numbers used by the *Sip* host. The example shows a regular partition of the attribute range; e.g., with a bin width set to 4096, the number of bins is 16.

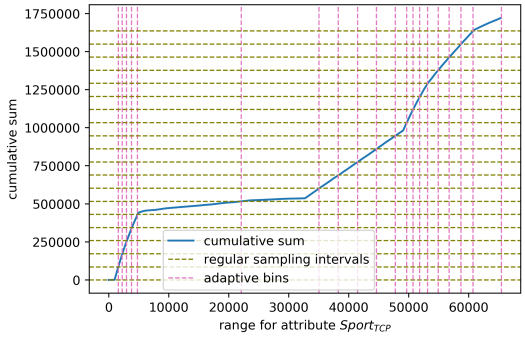
Adaptive bins: intervals are chosen depending on the amount of traffic. Intuitively, the more density of information there is, the more sensitive (small) the step should be. Thus we aim to define individual bin width so that we equalize the occurrences over the different bins, i.e., it is always the same for all the bins, each bin having potentially a different bin width. To do so, we first start with the highest attribute granularity (e.g., 65,536 for the port number), and compute the frequency distribution for all hosts. Then, we sum up the obtained vectors across all hosts. At this point, we are able to define individual bin steps so that the occurrences are evenly distributed across bins; this operation can be done for instance taking the cumulative distribution function and evenly dividing the probability range from 0 to 1 in the number of desired bins. We repeat this process for all the attributes.

Fig. 3 illustrates the computation of adaptive bins for the $\text{Sport}_{\text{TCP}}$ attribute. We first compute the number of unique values for very small bins as shown in Fig. 3a, then we divide the cumulative sum by a fixed number of regular sampling intervals and compute adaptive bins so that each of them contains the same number of packets, as shown in Fig. 3b.

The number b of bins and the bin aggregation strategy (regular vs. adaptive bins) are to be assessed experimentally.



(a) Number of unique values using 1000 bins.



(b) Cumulative sum and definition of 20 adaptive bins.

Fig. 3: Example partitioning in 20 adaptive bins based on the traffic load, for attribute $\text{Sport}_{\text{TCP}}$.

2) Observable bot behavior and attributes: Let us report on the observable behaviours for TCP, UDP and ICMP attributes from traces we could have access to. To get a visual representation of such behaviors, we propose in the supplementary materials fingerprints of infected hosts highlighting their specific malicious activities. First, we observe uncommon behaviours specific to a botnet for **TCP** flows:

- *destination ports* ($\text{Dport}_{\text{TCP}}$) usually range between 0 and 1023. These service ports are associated to given services by the Internet Assigned Numbers Authority (IANA) [35], e.g., TCP/80 typically runs HTTP and TCP/443 HTTPS. However, bots show different usage of destination ports: they are usually diverse and represent services often targeted by attackers such as TCP/25 (SMTP) or TCP/23 (Telnet), vulnerable to spam and attacks. We also observe some exotic destination port numbers used to access proxies that host the C&C server.
- *source ports* ($\text{Sport}_{\text{TCP}}$) are often ephemeral ports, allocated automatically from a predefined range by the IP stack software. The range recommended by IANA is 49152 to 65535, while many Linux kernels use the port range 32768 to 61000. FreeBSD has used the IANA port range since release 4.6. Previous versions, including the Berkeley Software Distribution, use ports 1025 to 5000 as ephemeral ports. Microsoft Windows Operating Systems (OS) until Windows XP used the range [1025, 5000] for ephemeral ports, while use the IANA range now.
  We observe that bots rarely use the IANA recommended range, but rather the [1025, 5000] range. This obviously depends on the OS of the infected host. A report from Kaspersky Labs [36] shows that Linux and Windows botnets represent respectively 95.75% and 4.25% of all botnets, which is very different from the OS distribution for regular devices (not bots), probably because bots infect vulnerable devices including connected objects.
- *destination IP addresses* (Dip_{TCP}): not all subnets are covered, but only some specific ones are contacted by normal hosts. Among them, it is common to observe addresses in the same range of the source IP address, private networks including `192.168.0.0/16`, and cloud service subnetworks, mostly Google ones, often contacted for Google Analytics and similar collateral services. Destination IP addresses cover a larger space for bots than for normal nodes, in case of a spam for example. Looking to the AS details in the *whois* database [37] also gives additional information, such as the age of the domain or its originated country.

There are specific botnet behaviours also for **UDP** flows:

- UDP *destination ports* ($\text{Dport}_{\text{UDP}}$) are associated to particular services by IANA, as for TCP. In the case of UDP, we often observe a fixed destination port set to 53. It represents connections to the local DNS server as UDP/53 typically runs DNS.
- UDP *source ports* ($\text{Sport}_{\text{UDP}}$) are used for ephemeral ports as for TCP, their range depends on the OS implementation. We notice that the range for ephemeral ports used by bots is often different than for common hosts.
- there is usually a fixed *destination IP address* (Dip_{UDP}) that represents the DNS server IP address.

Finally, **ICMP** flows also show specific botnet behaviours:

- the ICMP *type* ($\text{Type}_{\text{ICMP}}$) indicates the type of ICMP

message and gives a global information about the kind of message (e.g., 0 for Echo Reply and 3 for Destination Unreachable) as specified in RFC2780 [38]. We often observe only a small amount of ICMP packets. In case of a botnet, we sometimes observe many ICMP messages with uncommon types and codes, consisting in a Ping Flood or an ICMP DoS attack.

- the ICMP *code* ($\text{Code}_{\text{ICMP}}$) represents the ICMP subtype and gives additional context information for the message (e.g., if the type is 3, the code can be 0 if the destination network is unreachable or 1 if the destination host is unreachable, etc).
- the hosts frequently reply to *destination IP addresses* (Dip_{ICMP}) that targeted them, with messages like "port unreachable" if it was a port scanning. The number of such packets is low for benign hosts, and larger for bots.

Looking to attributes individually enables to retrieve some botnets behaviours, but it is even better to analyze these attributes together. Actually, sometimes it is the combination of two attributes that makes a host behaviour abnormal.

E. Signatures formatting

As shown in Fig. 2 (**step 4**), the training phase deals with flows from the training set through several modules. Data preparation is accomplished through pre-processing and dimensionality reduction. In the pre-processing phase, signatures of hosts belonging to the internal network are computed as described in the previous step. Finally, we reduce the dimensionality of the space by finding the directions of maximal variance, in order to reduce the spatial complexity. Thus we project the dataset into the new dimensional space.

1) Removing the less significant hosts: The threshold m is the minimum number of packets per host and protocol to consider it. We analyze the distribution of the number of packets per host to better understand its impact. The distribution is a long-tailed one, with hosts with a very high number of packets (up to 200,000 packets per host). Therefore, Fig. 4 shows the Probability Distribution Function (PDF) of the number of TCP packets per host, only from 0 to 1500 packets per host, avoiding very large outliers. The plot is about TCP, while UDP and ICMP exhibit the same distribution. Eliminating hosts with less than m packets has a minor impact on the results, as we notice that after removing hosts with less than 150 packets, the number of hosts goes from 4923 to 1933, which represents only 0.7% of removed traffic in terms of number of packets (from 13,342,675 to 13,295,640 packets). We could miss very stealthy bots using the filter, but we assume that bots have to be a minimum active to be efficient (including attacks, scans and communications with the C&C server).

To choose its exact value, we evaluate the impact of parameter m on the bot detection results in Section VI-A.

2) Dimensionality Reduction: After the pre-processing step, the signatures contain $|A| \times b$ columns, with b the number of bins and $|A| = 9$. Our purpose is to reduce the number of columns, by restricting the scope to the most meaningful ones. We reduce the dimensionality of the space working with

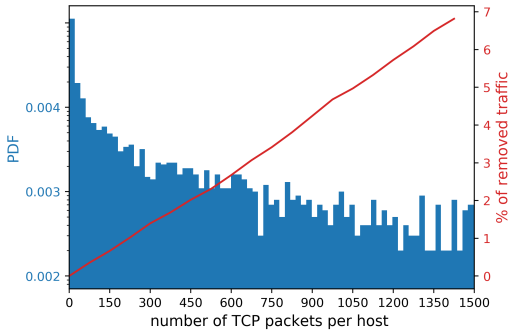


Fig. 4: PDF of the number of TCP packets per host, and (in red, right axis) possible m threshold values and corresponding traffic volume ratio.

the Principal Component Analysis (PCA) technique. PCA is applied on hosts from the training set $\mathcal{T}$, thus on a matrix of size $(|\mathcal{T}|, |A| \times b)$. PCA reduces the number of components of the already smallest dimension, then $|A| \times b$ must be smaller than $\mathcal{T}$ to reduce the number of features. Therefore, it is not applicable if $|A| \cdot b$ is larger than $|\mathcal{T}|$.

PCA finds the directions of maximum variance. The fraction of variance explained by a principal component is the ratio between the variance of that principal component and the total variance. Our goal is to reduce the dimensionality while keeping a good amount of information, so that the cumulative explained variance ratio is close to 100%. Fig. 5 hereafter shows the cumulative explained variance ratio vs. the number of factors, with $b = 128$. To get 99% of the variance ratio, we can reduce the factor number by ten, approximately, keeping around 150 and 200 factors respectively for regular and adaptive bins, over the 1152 original factors. These values vary with the number of bins: for each of them we need to choose the right number of factors. For all values of b, we notice that we can reach 99% of the variance ratio by using 9 times less factors.

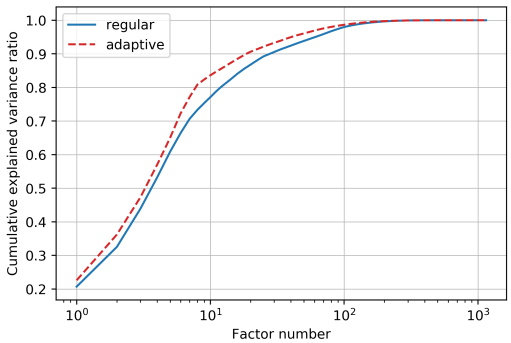


Fig. 5: Explained variance vs. number of factors.

V. Bot Detection

In this section we introduce two different bot detection techniques for *BotFP* that we designed for the training and the classification. *BotFP-Clus* relies on clustering as described in our previous work [1], and *BotFP-ML* relies on other supervised machine learning (ML) techniques.

A. BotFP-Clus

1) Training: Clustering algorithms are designed to group similar vectors into clusters and identify isolated ones as outliers. The similarity between two vectors is evaluated using a distance function like the Euclidean distance. Two vectors are defined as similar if they are close to each other, else dissimilar. We use as clustering algorithm DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [39], because it presents the advantage of discovering clusters without knowing the number of clusters in advance, which fits our needs. In addition, it works well on our data because clusters have close densities. DBSCAN uses two parameters:

- ϵ specifying the radius of a neighborhood with respect to some point. Every point situated within a distance ϵ from a point p is a neighbor of p; [2]
- $minPts$ which defines the minimum number of points in a radius ϵ to form a cluster.

DBSCAN defines a cluster as the maximal set of points where every pair of points p and q are within a distance ϵ from each other, and considers points that do not belong to any cluster as outliers. In our solution, DBSCAN is used in a slightly different manner as illustrated in Step 4 of Fig. 2. We set $minPts$ to 1 in order to consider singleton clusters as well. Then DBSCAN is applied on the vectors of σ_j from the training set to build clusters of similar host signatures. Using clusters instead of singular hosts enables to filter abnormal hosts and get more consistent data. This also reduces the number of coordinates to store.

Let C be the set of clusters obtained applying DBSCAN on the training set. Each cluster $c \in C$ contains several attributes:

- a set H_c of hosts belonging to the cluster;
- its position P_c defined as the centroid of the set of signatures $\{\sigma_1, \sigma_2, ..., \sigma_N\}$ of hosts in H_c, computed as $P_c = (\sigma_1 + \sigma_2 + ... + \sigma_N) \ / \ N$;
- a label identifying the nature of the cluster c, i.e., malicious or benign, denoted L_c.
  The nodes that are bots are known from the ground truth of the training set. The cluster is identified as a bot cluster if it contains at least one bot, else it is benign.

$$L_c = \begin{cases} \text{'bot' if } H_c \text{ contains a bot node} \\ \text{'benign' otherwise} \end{cases} \quad (2)$$

One may find the above condition to label bot clusters too strong. However, our tests showed that appropriately tuning ϵ, one can get a good clustering solution. A good setting we found is an ϵ set to 300 and 512 adaptive bins, giving that bot clusters always contain one bot at maximum, except one case with two bots, and hence they did not contain any normal host. Thus it appears sufficient to label a cluster as bot if it contains at least one bot. About the cluster density, we notice that other clusters contain up to 2400 hosts (all of them benign), but most of the time only one or several ones.

[2]The metric that we use for the computation of the distance between two hosts in DBSCAN is the $\ell 1$-norm defined as $||\sigma_h||_1 = |\sigma_h[1]| + ... + |\sigma_h[n]|$: this distance is robust and does not vary with the number of bins, as the cumulative sum of all elements stays equal. We consider it better than the $\ell 2$-norm – defined as $||\sigma_h||_2 = \sqrt{|\sigma_h[1]|^2 + ... + |\sigma_h[n]|^2}$ which increases with the number of bins.

2) Classification: We classify hosts from the test set based on their distance to the set of labelled clusters C. For a host $h \in \mathcal{E}$, if the closest cluster is labelled as bot, h will be classified as a bot. If the closest cluster is benign, h will be classified as benign too. Let $dist()$ be a function measuring the distance between a signature and a cluster, c^* is the closest cluster such that $dist(\sigma_h, P_{c^*}) = \min_c[\text{dist}(\sigma_h, P_c)]$. Then hosts are classified based on the label of c^*.

$$L_h = \begin{cases} \text{'bot' if } L_{c^*} = \text{'bot'} \\ \text{'benign' otherwise} \end{cases} \quad (3)$$

B. BotFP-ML

Several techniques can be used to learn from the training set what constitutes signatures either from benign hosts or from bots, then classifying hosts from the test set based on that knowledge. We evaluate four such techniques for BotFP: (i) Logistic Regression, used to predict the probability of a binary dependent variable, (ii) Support Vector Machines (SVM), which, given labeled training data, output an optimal hyperplane which categorizes new examples, (iii) Random Forest, which creates a forest with a number of decision trees, and (iv) Multilayer Perceptron (MLP) classifier, which can be thought as a deep artificial neural network, composed of an input and output layers, and an arbitrary number of hidden layers.

Supervised learning algorithms and neural networks take into account hyperparameters that must be tuned to obtain the best results of classification. Grid search [40] is used for model tuning, it builds and evaluates a model for every combination of hyperparameters specified, then selects the best one to improve a given evaluation metric. We used it by favouring the recall criterion. The hyperparameters are different according to the type of classifier. For instance, with SVM and logistic regression, the parameter C controls the sparsity: the smaller C, the fewer features selected. Another parameter common for both these algorithms is the penalty, which is used to specify the norm used in the penalization (regularization or noise variance).

VI. Evaluation

In this section, we evaluate the performance of BotFP. We first propose an evaluation of the method *BotFP-Clus*, tuning its parameters including DBSCAN ϵ, the number of bins b and the type of bins. We then analyze and compare *BotFP-Clus* and *BotFP-ML* performances and we select a set of best solutions. Finally, we compare them to other state-of-the-art detection techniques. Note that the source code for BotFP is available in [15]. The subnetwork address needs to be set up, then the reader can run the learning phase (learning the normal and malicious behaviors of the network and tuning the key parameters), and then launch the detection process.

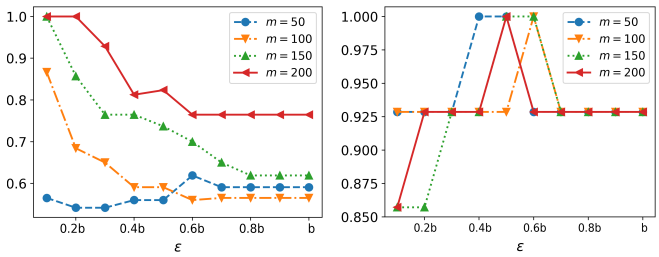
A. BotFP-Clus

BotFP-Clus is our proposal relying on labeled clusters of similar hosts behaviours. We apply it on the CTU-13 dataset, following the methodology presented in Sections IV and V.

We analyze the results as a function of the three key parameters to be chosen for *BotFP-Clus*: the minimum number of packets threshold (m), the number of bins (b), and the ϵ DBSCAN parameter. We proceed as follows: first we show precision and recall results as a function of m to find a reasonable choice; second, we elaborate on the impact of ϵ, and identify one good setting. We also show the benefit in using adaptive bins rather than regular ones.

Fig. 6a and 6b respectively show the precision and the recall of *BotFP-Clus*, for 512 adaptive bins, ϵ in $[0.1 \cdot b, 0.2 \cdot b, ..., b]$, and m varying in [50, 100, 150, 200]. We observe that for all m values, the recall is very high, reaching 100% in many settings. It is important to note that the precision is directly correlated with the minimum number of packets threshold: the higher m and the higher the precision.

For the following experiments, to favour the recall we choose $m = 150$, because the precision is almost as high as for $m = 200$, but the recall is more often equal to 100%. Fig. 7 shows for regular bins the precision (Fig. 7a), the recall (Fig. 7b), the F1-score (Fig. 7c) and the number of clusters (Fig. 7d). Fig. 8 shows for adaptive bins the precision (Fig. 8a), the recall (Fig. 8b), the F1-score (Fig. 8c) and the number of clusters (Fig. 8d). Multiple ϵ values (DBSCAN parameter) are tested in $[0.1 \cdot b, 0.2 \cdot b, ..., b]$.



(a) Precision. (b) Recall.

Fig. 6: Impact of parameter m on the precision and recall, for 512 adaptive bins.

1) DBSCAN ϵ choice: We use the $\ell 1$-norm as distance function; it increases linearly with the number of elements in the vector. Therefore, taking the parameter m as a constant, we can establish a general relationship between ϵ and b, that is why we chose ϵ as a fraction of b in Fig. 7 and 8.

A large value of ϵ may produce too large clusters resulting in false positives, while a too small value may overfit the data and miss bots. One way to choose a good value for ϵ can be to take the one for which the recall and the precision are the highest; overall, we favour the recall as we want to detect the most bots as possible. For regular bins, $\epsilon_{reg} = 0.4 \times b$ seems the best choice as the recall is high and we get an acceptable precision and F1-score. For adaptive bins, the best value is $\epsilon_{\text{ad}} = 0.5 \times b$, with the highest recall and a quite good F1-score, for all values of b.

2) Comparison between regular and adaptive bins: We observe that formatting the data by handling adaptive bins gives more consistent results and eases the process of clustering. Fig. 7b shows the recall for regular bins: we observe that the recall values are quite unstable even with high ϵ values. For adaptive bins (Fig. 8b) on the contrary, the recall oscillates between 85% and 100% (i.e., between 0 and 2 undetected bots) for ϵ starting from 150 and all values of b, which is quite stable.

Only looking to the recall is not sufficient, we also need to know the precision (number of false positives). We observe that using adaptive bins (Fig. 8a) presents a far higher precision compared to regular ones (Fig. 7a).

For these two reasons, we could confirm the intuition that using adaptive bins grants a more accurate view and therefore leads to better results.

3) General observations: Let us draw further observations from these preliminary results.

- *Benefits of clustering*

In Fig. 7 and 8, $\epsilon = 0$ is equivalent to not clustering the data, i.e., comparing each host from the test set to labelled hosts from the training set. Fig. 7b and 8b show that the recall never reaches 100% in this case, no matter b and the quantification technique, as the classification is too specific and we overfit the data. However, increasing ϵ (then forming larger and larger clusters) enables to detect all bots in some setups.

Clustering the data also reduces the complexity of the classification, by limiting the number of comparisons to do: we compare hosts from the test set to a limited set of clusters, rather than to all hosts from the training set.

- *Number of bins b*

The number of bins b is the third parameter to choose, which determines our suggested ϵ values as above discussed. The objective is to find a setup with a recall equal to 100% (i.e., all bots detected) and a precision as low as possible.

Using adaptive bins, the recall reaches 100% for nearly all values of b. However there is a strong correlation between b and the precision: the higher b, the higher the precision (thus the lower the number of false positives). Therefore the best solution is reached for a high number of bins ($b = 512$), for which the recall is equal to 100% and the precision is high.

- *Number of clusters*

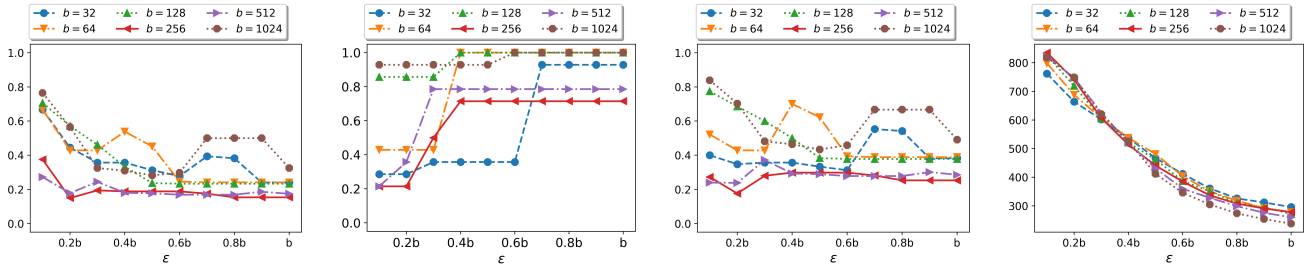
Finally, Fig. 7d and 8d show the number of clusters respectively for regular and adaptive bins. This shows the benefits in clustering the data: 550 clusters for $b = 512$ (Fig. 8d) is approximately 60% less than the 910 initial hosts.

B. Comparison between BotFP-Clus and BotFP-ML

We compare *BotFP-Clus* and *BotFP-ML* in terms of precision, recall and F1-score. For the former, we use the $\epsilon_{reg} = 0.4b$ and $\epsilon_{\text{ad}} = 0.5b$ settings identified in the previous section. For the latter, we tune the hyperparameters with Gridsearch by favouring the recall. In addition, we show in the supplementary materials the most relevant features in the classification process.

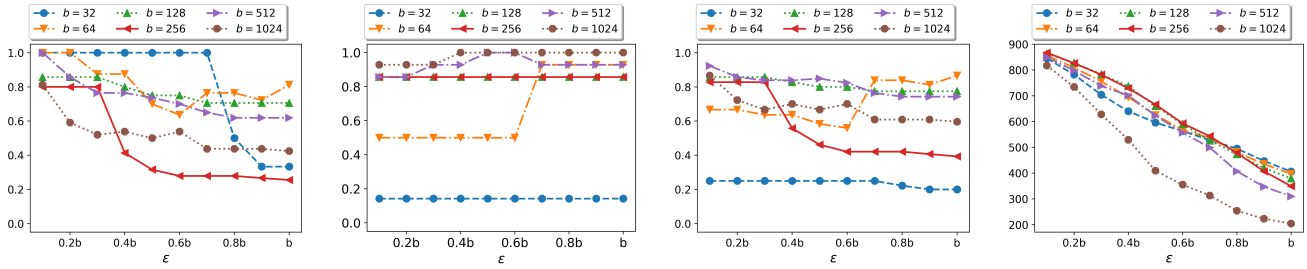
Fig. 9 and 10 compare the precision, the recall and the F1-score for *BotFP-Clus* and *BotFP-ML*, respectively for regular and adaptive bins, and for b between 8 and 1024.

Let us look first into *BotFP-ML* algorithms. For Random Forest, both for regular and adaptive bins, the precision and recall values are too unpredictable and varying with b. For SVM, it is also too varying for adaptive bins, but for regular



(a) Precision. (b) Recall. (c) F1-score. (d) Number of clusters.

Fig. 7: Regular bins: precision, recall, F1-score and number of clusters (*BotFP-Clus*).



(a) Precision. (b) Recall. (c) F1-score. (d) Number of clusters.

Fig. 8: Adaptive bins: precision, recall, F1-score and number of clusters (*BotFP-Clus*).

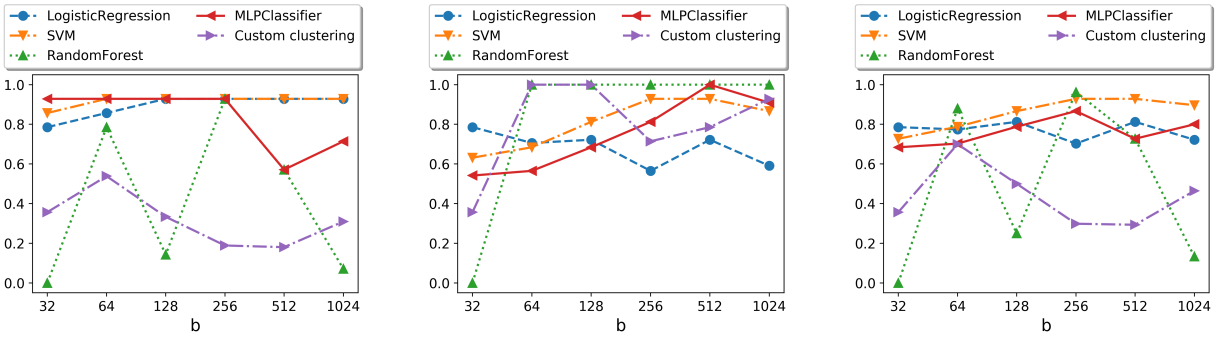
ones the precision and the recall are very high (both 93%) starting from $b = 256$. For the MLP classifier and Logistic Regression, the precision and recall values for both types of bins are quite high for all values of b. The recall for the MLP classifier is higher for adaptive bins, reaching 90% for 64 bins. For Logistic Regression, there is no notable difference between the use of regular and adaptive bins. In all cases, the recall never reaches 100% no matter b (or with a very low precision), which means that there are still some undetected bots.

For *BotFP-Clus*, the precision is very low for regular bins. We observe a correlation between b and the precision and recall: the higher b, the lower the precision and the higher the recall. Thus we may opt for the *BotFP-Clus* with a high value of b. In particular for 512 and 1024 bins, the recall reaches 100% (i.e., all bots detected) and 55 to 75% precision. Actually, a precision equal to 75% represents very few false positives: 4 benign hosts classified as bots, out of the 712 benign ones.

To sum up, we selected four best-performing solutions summed up in Table III according to the parameter we want to favour. Grey cases show the values for which a given parameter is the best one across all solutions. Let us further comment on the following goals:

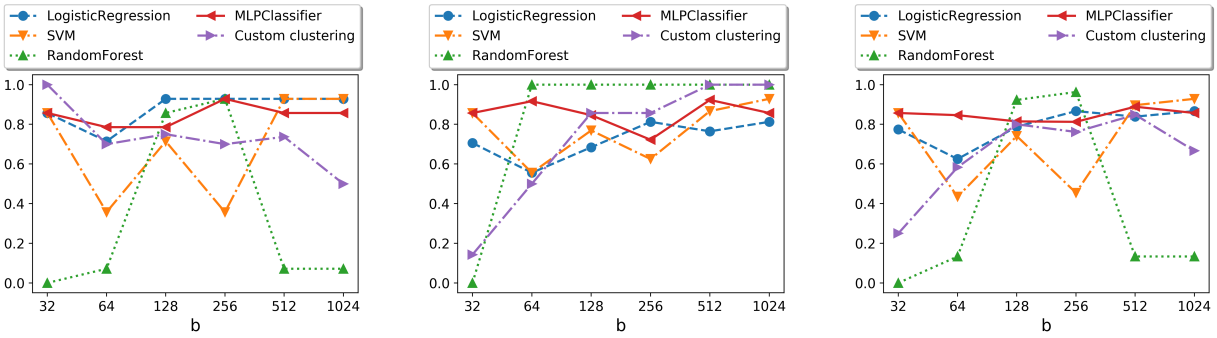
- *Maximize true bot detection*: We recommend *BotFP-Clus* with 512 adaptive bins, for which the recall is equal to 100% and the precision to 75%. Contrary to others, this method enables to detect all bots while keeping a good precision;
- *Balance the precision and the recall, thus maximize the F1-score*: *BotFP-SVM* with a linear kernel and 256 regular bins is ideal in this case. The recall and the precision are both equal to 93%. Moreover, this method does not require to compute adaptive bins, therefore is more lightweight than others in this respect;
- *Minimize the memory usage*: *BotFP-MLP* with 32 adaptive bins best suits this goal. As b is lower than 256 (see Section IV-E2), we can apply PCA beforehand to keep only 50 out of the 288 initial factors. The recall and the precision are still good, both equal to 84%. Note that we obtain exactly the same precision and recall values than without applying PCA. Therefore this solution is efficient and above all very lightweight;
- *Maximize precision:* One may choose to favour a high precision, i.e. a low false detection rate (FDR) defined as $FDR = 1 - precision$, especially in production environments where administrators want to receive as few alerts as possible. In this case, we recommend *BotFP-Clus* with 512 adaptive bins and $\epsilon = 0.1b$ instead of the ϵ value previously tuned to favour the recall. Using this setting, the precision is equal to 100%, the recall to 85% and the F1-score to 93%.

Table IV shows the confusion matrix for scenarios 1, 2, 6, 8, 9 from the test set, for the best cases that we identified, i.e. the *BotFP-Clus* algorithm with $b = 512$ adaptive bins and $\epsilon_{ad} = 0.5 \times b$, the *BotFP-MLP* algorithm with $b = 32$ adaptive bins, the *BotFP-SVM* algorithm with $b = 256$ regular bins, and the *BotFP-Clus* algorithm with $b = 512$ adaptive bins and $\epsilon_{ad} = 0.1 \times b$ For *BotFP-Clus* and ϵ_{ad}, we detected bots from all scenarios (1 bot for scenarios 1, 2, 6, 8, and 10 bots for scenario 9), which makes the recall equal to 100%. In total, we labelled 9 benign hosts as bots, which results in a very high precision equal to 74%. For *BotFP-MLP*, 2 bots have not been detected, while it remains 2 false positives. For *BotFP-SVM*, there are only 3 false positives, but it remains



(a) Precision. (b) Recall. (c) F1-score.

Fig. 9: Regular bins: precision, recall and F1-score for both approaches.



(a) Precision. (b) Recall. (c) F1-score.

Fig. 10: Adaptive bins: precision, recall and F1-score for both approaches.

Solution	Bins type	ϵ	b	Precision	Recall	F1-score	PCA	Complexity
BotFP-Clus	adaptive	$0.5b$	512	74%	100%	80%	X	low
BotFP-MLP	adaptive	$0.5b$	32	85%	85%	85%	✓	high
BotFP-SVM	regular	$0.4b$	256	93%	93%	93%	X	medium
BotFP-Clus	adaptive	$0.1b$	512	100%	85%	80%	X	low

TABLE III: Summary of the best solutions according to the detection performance to favour.

one undetected bot. For *BotFP-Clus* and $\epsilon_{ad} = 0.1 \times b$, there are no false positives as we favour the precision, but there are 5 undetected bots.

C. Comparison to state-of-the-art detection techniques

We now compare the four selected BotFP settings to other state-of-the-art detection methods, namely BClus [9], CAMNEP [9], BotHunter [7], *BotGM* [11] and [13] described in Section II.

To compare BotFP to other methods, we compute the accuracy for scenarios from the test set, as proposed in [9]. Table V reports the results for each solution and all scenarios from the test set, as proposed in [9]. Our results are very competitive as we reach an accuracy between 97% and 100% with the MLP classifier, SVM and *BotFP-Clus*, while other algorithms provide an accuracy between 30% and 95%. Only [13] achieves up to 100% accuracy for scenario #9 but it tested only that one and trained on the 12 other scenarios. Note also that our algorithm has a far lower complexity (cf. Section VII).

VII. Complexity

We qualify the space and time complexity of *BotFP*, considering its three steps: attribute frequency distributions computation, training and classification. We consider that the classification takes all the elements in the test set $\mathcal{E}$, even if hosts may also be processed individually in practice. We also compare it to other recent bot detection methods.

A. Attribute frequency distributions computation

First, we need to compute the fingerprint σ_j for all hosts.

1) Space complexity: given a host and $|A|$ attributes, we need to store arrays of b bins for all the attributes, then the per-host space complexity is equal to $O(|A| \cdot b)$. The overall process is a one-shot operation over all hosts, resulting in a complexity $O(|\mathcal{T} \bigcup \mathcal{E}| \cdot |A| \cdot b)$. In our setting we have $|A| = 9$, which can be reduced to $|A'| = 1$ when using PCA.

2) Time complexity: For a given host i, the computation of each attribute vector comes with $|a_i|$ entry readings, before bin aggregation, thus the worst-case time complexity is equal to $O(|A| \cdot max_i|a_i| \cdot |\mathcal{T} \bigcup \mathcal{E}|)$.

Id	TP	TN	FP	FN
1	1	164	3	0
2	1	131	0	0
6	1	112	0	0
8	1	163	4	0
9	10	133	1	0

(a) *BotFP-Clus*, $b = 512$ adaptive bins and ϵ_{ad}.

Id	TP	TN	FP	FN
1	1	166	0	0
2	0	131	0	1
6	1	111	1	0
8	1	164	1	0
9	9	134	0	1

(b) *BotFP-MLP*, $b = 32$ adaptive bins.

Id	TP	TN	FP	FN
1	1	166	0	0
2	1	131	0	0
6	1	111	1	0
8	1	165	2	0
9	9	134	0	1

(c) *BotFP-SVM*, $b = 256$ regular bins.

Id	TP	TN	FP	FN
1	1	166	0	0
2	0	131	0	1
6	1	111	0	0
8	1	165	0	0
9	7	134	0	3

(d) *BotFP-Clus*, $b = 512$ adaptive bins and $\epsilon = 0.1b$.

TABLE IV: Confusion matrix for scenarios from the test set, for four BotFP settings.

Metrics	Recall					Precision					Accuracy				
Algorithm	**1**	**2**	**6**	**8**	**9**	**1**	**2**	**6**	**8**	**9**	**1**	**2**	**6**	**8**	**9**
BClus [9] (2014)	0.4	0.3	<0.0	0.1	0.1	0.5	0.6	0.4	0.2	0.4	0.5	0.5	0.4	0.3	0.4
CAMNEP [9] (2014)	0	<0.0	<0.0	<0.0	<0.0	<0.0	0.8	0.9	0.9	0.9	0.5	0.4	0.4	0.5	0.5
BotHunter [7] (2007)	0.01	0.02	0.06	0	0.02	0.8	0.9	0.98	0	0.9	0.4	0.3	0.38	0.42	0.4
BotGM[3] [11] (2017)	X	X	X	X	X	X	X	X	X	X	0.91	0.78	0.95	0.89	0.83
Graph-based ML[4] [13] (2019)	X	X	X	X	1	X	X	X	X	0.91	X	X	X	X	1
***BotFP-Clus* (b = 512)**	**1**	**1**	**1**	**1**	**1**	**0.25**	**1**	**1**	**0.2**	**0.91**	**0.98**	**1**	**1**	**0.97**	**0.99**
***BotFP-MLP* (b = 32)**	**1**	**0**	**1**	**1**	**0.9**	**1**	**0**	**0.5**	**0.5**	**1**	**1**	**0.98**	**0.99**	**0.98**	**0.99**
***BotFP-SVM* (b = 256)**	**1**	**1**	**1**	**1**	**0.9**	**1**	**1**	**0.5**	**0.33**	**1**	**1**	**1**	**1**	**1**	**0.99**
***BotFP-Clus* (b = 512 - $\epsilon = 0.1b$)**	**1**	**0**	**1**	**1**	**0.7**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0.99**

TABLE V: Recall, precision and accuracy of different algorithms compared to *BotFP*. [3]: note that BotGM [11] did not provide per-scenario recall and precision values; however, they provided ROC curves showing a TPR (recall) equal to 80% for FPR=0, but we have no information about which scenario they used for this plot. [4]: the training was done on 12 scenarios (including 1, 2, 6 and 8) and the evaluation only on scenario 9.

B. Training

The training phase depends on the implemented supervised learning algorithm. For *BotFP-Clus*, the training consists in building host clusters from the training set, each host being characterized by its fingerprint σ_j.

1) Space complexity: For *BotFP-MLP*, a one-dimensional neuron input array grows linearly with the number of neurons, which send their outputs as inputs to a given neuron [41], thus $O(h)$ with h the number of neurons. The space complexity of *BotFP-SVM* is around $O(|\mathcal{T}|^2)$ [42]. For *BotFP-Clus*, DBSCAN presents a space complexity of $O(k|\mathcal{T}|)$, where k is a fixed memory cost to store the positions and labels of each among the $|\mathcal{T}|$ points, their labels and the neighbors of the current point.

2) Time complexity: For *BotFP-MLP*, the time complexity of the training (backpropagation) for a single iteration is $O(|\mathcal{T}| \cdot |A| \cdot b \cdot h^k)$, for k hidden layers containing h neurons. For *BotFP-SVM*, the time complexity of the training is $O(|\mathcal{T}|^2 \cdot |A| \cdot b)$ [43]. For *BotFP-Clus*, DBSCAN presents a worst-case time complexity of $O(|\mathcal{T}|^2)$ (without the use of an accelerating index structure). For each point of the database, we have to visit each other point to query their neighborhood.

C. Classification

For *BotFP-ML*, the classification technique depends on the implemented supervised learning algorithm. For *BotFP-Clus*, the classification determines the closest cluster to each host to classify, and assign its label to the host.

1) Space complexity: For testing as well, *BotFP-MLP* presents a worst-case space complexity of $O(h)$ with h the number of neurons. The test space complexity for a linear SVM is $O(|\mathcal{E}|)$ [44]. For *BotFP-Clus*, we have to store the positions of all clusters. Also for each host, we need to store the distance between its signature and each cluster. Therefore the total space complexity is $O(|C| \cdot |A| \cdot b + |\mathcal{E}| \cdot |C|)$.

2) Time complexity: For a trained MLP, the overall complexity of the classification (forward propagation) is $O(|\mathcal{E}| \cdot |A| \cdot b)$ [45]. For a trained SVM, the overall complexity of the classification is $O(|\mathcal{E}|^3)$ [43]. For *BotFP-Clus*, we need to parse all hosts from the test set, then to compare each of them to all clusters with a $\ell 1$-norm, thus the time complexity is equal to $O(|A| \cdot b \cdot |\mathcal{E}| \cdot |C|)$.

D. Comparison to other techniques

The time complexity of *BotFP* feature computation is therefore linear with the number of nodes, then the training is linear for *BotFP-Clus*, quadratic for *BotFP-SVM* and up to exponential for *BotFP-MLP*. For classification, if one considers that it would in practice run in runtime on a per host basis (i.e., $|\mathcal{E}| = 1$), *BotFP-Clus* and *BotFP-MLP* are linear with the number of hosts, knowing that $|C| < |\mathcal{T}|$, hence very competitive; and *BotFP-SVM* time complexity is cubic.

Let us compare the time complexity of our method to recent bot detection techniques [11], [13], for each main step:

Features computation: BotGM [11] creates graphs of communications between two hosts as features to feed their algorithms. The time complexity is thus $O(|\mathcal{P}| \cdot |\mathcal{E}_\mathcal{P}| \cdot ln(|\mathcal{V}_\mathcal{P}|))$ with $\mathcal{P}$ the set of IP addresses pairs, $\mathcal{E}_\mathcal{P}$ the set of edges per pair (i.e., the set of source-destination port pairs per IP pair) and $\mathcal{V}_\mathcal{P}$ the set of vertices per pair (i.e., each communication from one pair to another). For [13], graphs of communications for all possible source-destination IP addresses are computed, which generates a complexity of $O(|\mathcal{P}| \cdot ln(|\mathcal{V}|))$ with $|\mathcal{V}|$ the set of vertices between each edge in $|\mathcal{P}|$. Then the complexity needed to compute features over the graphs is different depending on the feature, and is up to quadratic

for features like Betweenness Centrality. Both [11] and [13] compute features for each pair of hosts, while we work on individual hosts which thus implies linear instead of quadratic processing.

Training: [11] uses an unsupervised method thus does not require training. For [13], techniques used for training are quite heavy as they use unsupervised (mainly clustering algorithms including k-Means which is NP-hard) then unsupervised (various classifiers) learning algorithms which are heavy too. Looking to the classification results of their algorithm, we assess that they better use k-Means followed by DecisionTree. k-Means is known to have a quadratic time complexity $O(|\mathcal{T}|^2)$, and the standard decision-tree has a time complexity of $O(|A|\cdot b\cdot k\cdot|\mathcal{T}|)$, with N the number of training examples and d the depth of the decision tree.

Classification: The classification step in [11] is very costly, they compute for each possible pair of graph ($O(|\mathcal{P}|^2)$ the GED which is NP-hard [46]. Thus the total time complexity is $O(|\mathcal{P}|^2\cdot\max(n_1,n_2)^3)$ with n_1 and n_2 the number of elements in the two graphs to compare. They compute these distances for each possible pair of graphs, then $|\mathcal{P}|^2$ times with n the number of graphs. Classification in [13] consists in the same process than training, i.e., unsupervised then supervised learning algorithms. The training and classification phases are simultaneous in k-Means and Decision Tree, thus their time complexity is simply $O(|\mathcal{E}|)$.

Hereafter, Table VI compares the time complexity for recent bot detection techniques, namely [11], [13] and ours.

Above all, *BotFP* is lightweight with respect to recent bot detection techniques: it deals with features consisting in vectors, easy to compute, and not graphs. For *BotFP-Clus*, the training is very lightweight, and the classification consists in computing inexpensive $\ell 1$-norm distances. For *BotFP-MLP* and *BotFP-SVM*, the training is a bit more complex, especially depending on the number of layers and nodes. However, as for other algorithms, the training is made only once and the classification phase is quite lightweight. It is hard to establish the exact time complexity of other algorithms [11], [13] because we do not know the details of their implementations, but [11] works on every possible pair of nodes then draws expensive graphs and once again make every possible graph comparison. [13] uses expensive centrality measurements.

VIII. Conclusion and perspectives

Botnet attacks are constantly more sophisticated, and this is expected to get even worse with the massive increase of connected objects and virtualised infrastructures. Therefore the quick identification of such bots is crucial to Internet security. Our technique *BotFP* focuses on the detection of botnets that infect thousands of machines and perform malicious actions such as launching port scanning and DDoS attacks. We propose an attribute frequency distribution design to characterize hosts communication, where bots exhibit specific behaviours. We design two *BotFP* variants for the training and classification. *BotFP-Clus* clusters similar host signatures of each host to group similar instances of traffic, hence avoiding data overfitting and reducing the complexity. *BotFP-ML* applies a supervised ML algorithm to learn from the signatures and detect new bots. The detection results are very promising, since *BotFP* detected all bots from the CTU-13 dataset with very few false positives, outperforming alternative techniques at the state of the art. With both techniques, *BotFP* achieves an accuracy close to 100% while being very lightweight compared to graph-based techniques. The said variant is chosen according to the parameter we want to favour, such as a high recall or precision, a low complexity, a small number of features.

We showed that using only the information about 4-tuple flows is a very insightful way to characterize the communications of an host and enables to efficiently detect bots. We plan as future work to consider additional features including: the payload size, as crafted packets from bots usually have a lower size than usual packets; the TCP flags, e.g. to detect SYN flood; the inter-packet time; the flows duration, e.g. the connection to the C&C server might be persistent; and DNS features inspired from [24]. We also plan to develop a real-time implementation of our algorithm, working with sliding time windows and incrementally updating the model, based on [47].

References

[1] A. Blaise, M. Bouet, V. Conan, and S. Secci, "BotFP: FingerPrints Clustering for Bot Detection," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2020.

[2] ZDnet. Avast and french police take over malware botnet and disinfect 850,000 computers. [Online]. Available: https://www.zdnet.com/article/avast-and-french-police-take-over-malware-botnet-and-disinfect-850000-computers/

[3] ZDNet. A hacking group is hijacking docker systems with exposed api endpoints. [Online]. Available: https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml

[4] Mid-year update: 2019 sonicwall cyber threat report. [Online]. Available: https://blog.sonicwall.com/en-us/2019/07/mid-year-update-2019-sonicwall-cyber-threat-report/

[5] M. Mahmoud, M. Nir, and A. Matrawy, "A survey on botnet architectures, detection and defences," *I. J. Network Security*, vol. 17, pp. 264–281, 2015.

[6] B. AsSadhan, A. Bashaiwth, J. Al-Muhtadi, and S. Alshebeili, "Analysis of p2p, IRC and HTTP traffic for botnets detection," *Peer-to-Peer Networking and Applications*, vol. 11, no. 5, pp. 848–861, jul 2017.

[7] G. Gu, P. Porras, V. Yegneswaran, and M. Fong, "BotHunter: Detecting malware infection through ids-driven dialog correlation," in *Proceedings of the USENIX Security Symposium*. USENIX Association, 2007.

[8] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting botnet command and control channels in network traffic," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2008.

[9] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security*, vol. 45, pp. 100–123, 2014.

[10] S. García, V. Uhlíř, and M. Rehak, "Identifying and modeling botnet c&c behaviors," in *Proceedings of the 1st International Workshop on Agents and CyberSecurity - ACySE '14*. ACM Press, 2014.

[11] S. Lagraa, J. Francois, A. Lahmadi, M. Miner, C. Hammerschmidt, and R. State, "BotGM: Unsupervised graph mining to detect botnets in traffic flows," in *Proceedings of the Cyber Security in Networking Conference (CSNet)*. IEEE, 2017.

[12] W. Chen, X. Luo, and A. N. Zincir-Heywood, "Exploring a service-based normal behaviour profiling system for botnet detection," in *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017.

[13] A. A. Daya, M. A. Salahuddin, N. Limam, and R. Boutaba, "A graph-based machine learning approach for bot detection," in *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019.

[14] Stratosphere Lab. The CTU-13 Dataset. A Labeled Dataset with Botnet, Normal and Background traffic. [Online]. Available: www.stratosphereips.org/datasets-ctu13

Algorithm	Features	Training	Classification
BotGM [11]	$O(\|\mathcal{P}\| \cdot \|\mathcal{E}_{\mathcal{P}}\| \cdot ln(\|\mathcal{V}_{\mathcal{P}}\|))$	X	$O(\|\mathcal{P}\|^2 \cdot \max(n_1, n_2)^3)$
Graph-based ML [13]	$O(\|\mathcal{P}\| \cdot ln(\|\mathcal{V}\|))$	$O((\|\mathcal{T}\| + k) \cdot \|A\| \cdot b + \|A\| \cdot b \cdot k \cdot \|\mathcal{T}\|)$	$O(\|\mathcal{E}\|)$
BotFP-Clus (512 adaptive bins)	$O(\|A\| \cdot max_i \|a_i\| \cdot \|\mathcal{T} \bigcup \mathcal{E}\|)$	$O(k\|\mathcal{T}\|)$	$O(\|A\| \cdot b \cdot \|\mathcal{E}\| \cdot \|\mathcal{C}\|)$
BotFP-MLP (32 adaptive bins)	$O(\|A'\| \cdot max_i \|a_i\| \cdot \|\mathcal{T} \bigcup \mathcal{E}\|)$	$O(\|A'\| \cdot b \cdot \|\mathcal{T}\| \cdot h^k)$	$O(\|A'\| \cdot b \cdot \|\mathcal{E}\|)$
BotFP-SVM (256 regular bins)	$O(\|A\| \cdot max_i \|a_i\| \cdot \|\mathcal{T} \bigcup \mathcal{E}\|)$	$O(\|A\| \cdot b \cdot \|\mathcal{T}\|^2)$	$O(\|\mathcal{E}\|^3)$

TABLE VI: Time complexity of different bot detection algorithms.

[15] "Source code for BotFP algorithm," 2020. [Online]. Available: https://github.com/BotFP/botFP-detection

[16] M. F. Umer, M. Sher, and Y. Bi, "Flow-based intrusion detection: Techniques and challenges," *Computers & Security*, vol. 70, pp. 238–254, sep 2017.

[17] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, p. 219, oct 2004.

[18] M.-Y. Su, G.-J. Yu, and C.-Y. Lin, "A real-time network intrusion detection system for large-scale attacks based on an incremental mining approach," *Computers & Security*, vol. 28, no. 5, pp. 301–309, jul 2009.

[19] X. Bai, T. Zhang, C. Wang, A. A. A. El-Latif, and X. Niu, "A fully automatic player detection method based on one-class SVM," *IEICE Transactions on Information and Systems*, vol. E96.D, no. 2, pp. 387–391, 2013.

[20] A. A. A. El-Latif, B. Abd-El-Atty, W. Mazurczyk, C. Fung, and S. E. Venegas-Andraca, "Secure data encryption based on quantum walks for 5g internet of things scenario," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 118–131, mar 2020.

[21] R. Bhatia, S. Benno, J. Esteban, T. V. Lakshman, and J. Grogan, "Unsupervised machine learning for network-centric anomaly detection in IoT," in *Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks - Big-DAMA '19*. ACM Press, 2019.

[22] W. Lu and H. Tong, "Detecting network anomalies using CUSUM and EM clustering," in *Advances in Computation and Intelligence*, 2009, pp. 297–308.

[23] J. Francois, C. Wagner, R. State, and T. Engel, "SAFEM: Scalable analysis of flows with entropic measures and SVM," in *2012 IEEE Network Operations and Management Symposium*. IEEE, apr 2012.

[24] M. Singh, M. Singh, and S. Kaur, "Detecting bot-infected machines using DNS fingerprinting," *Digital Investigation*, vol. 28, pp. 14–33, mar 2019.

[25] S. Chowdhury, M. Khanzadeh, R. Akula, F. Zhang, S. Zhang, H. Medal, M. Marufuzzaman, and L. Bian, "Botnet detection using graph-based feature clustering," *Journal of Big Data*, vol. 4, no. 1, may 2017.

[26] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov, "BotGrep: Finding P2P Bots with Structured Graph Analysis," in *Proceedings of the USENIX Security Symposium*, 2010, pp. 95–110.

[27] H. Jiang and X. Shao, "Detecting p2p botnets by discovering flow dependency in c&c traffic," *Peer-to-Peer Networking and Applications*, vol. 7, no. 4, pp. 320–331, jun 2012.

[28] F. Zou, S. Zhang, W. Rao, and P. Yi, "Detecting malware based on DNS graph mining," *International Journal of Distributed Sensor Networks*, vol. 2015, pp. 1–12, 2015.

[29] J. Wang and I. C. Paschalidis, "Botnet detection based on anomaly and community detection," *IEEE Transactions on Control of Network Systems*, vol. 4, no. 2, pp. 392–404, jun 2017.

[30] P. Kalmbach, A. Blenk, W. Kellerer, and S. Schmid, "Themis: A data-driven approach to bot detection," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018.

[31] Stratosphere Lab. Stratosphere Research Laboratory. [Online]. Available: https://www.stratosphereips.org/

[32] ——. Aposemat IoT-23. [Online]. Available: https://www.stratosphereips.org/datasets-iot23

[33] ——. Malware Capture Facility Project. [Online]. Available: https://www.stratosphereips.org/datasets-malware

[34] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, 2018.

[35] (2013) Service name and transport protocol port number registry. [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

[36] Kaspersky. DDoS attacks in Q2 2019. [Online]. Available: https://securelist.com/ddos-report-q1-2019/90792/

[37] Whois domain lookup. [Online]. Available: www.whois.com/whois/

[38] IANA. Internet control message protocol (icmp) parameters. [Online]. Available: https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml

[39] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 226–231.

[40] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research (JMLR)*, 2012.

[41] G. Serpen and Z. Gao, "Complexity analysis of multilayer perceptron neural network embedded into a wireless sensor network," *Procedia Computer Science*, vol. 36, pp. 192–197, 2014.

[42] Y. Hou and X. F. Zheng, "SVM based MLP neural network algorithm and application in intrusion detection," in *Artificial Intelligence and Computational Intelligence*. Springer Berlin Heidelberg, 2011, pp. 340–345.

[43] A. Abdiansah and R. Wardoyo, "Time complexity analysis of support vector machines (svm) in libsvm," *International Journal of Computer Applications*, 2015.

[44] G. Sharma and F. Jurie, "A novel approach for efficient SVM classification with histogram intersection kernel," in *Procedings of the British Machine Vision Conference 2013*. British Machine Vision Association, 2013.

[45] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotun, "Map-reduce for machine learning on multicore," in *Proceedings of NIPS*, 2006.

[46] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou, "Comparing stars," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 25–36, 2009.

[47] A. D'Alconzo, I. Drago, A. Morichetta, M. Mellia, and P. Casas, "A survey on big data for network traffic monitoring and analysis," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 800–813, sep 2019.

Botnet Fingerprinting: a Frequency Distributions Scheme for Lightweight Bot Detection Supplementary Materials

Agathe Blaise* Mathieu Bouet† Vania Conan† Stefano Secci‡

1 Observation of bots fingerprints

Fig. 1a shows a typical example of a bot (`147.32.84.165`) from scenario #1 of CTU-13, where each point is representing one flow (bots from scenarios #2 and #6 show a similar behaviour); through the graphs we can infer its actions: scanning or spamming (looking to the whole range of IP addresses targeted), infection of other hosts by searching for their vulnerabilities (looking to the used destination ports corresponding to many vulnerable services), communication with the C&C server via a proxy. We also observe regular connections to the C&C server, using the exotic TCP/65000 port and the IRC protocol. Finally, the range for ephemeral ports is different from quite unusual, neither the range recommended by IANA or the typical Linux range.

Fig. 1b is an example of an infected host (`192.168.100.111`) from scenario #17 of IoT-23, performing several malicious activities; its behavior is close to hosts from scenarios #5, 7 from CTU-13 which performed port scanning. For this host, we observe usual TCP and UDP connections, but also port scanning: targeting port 8081 (alternative HTTP port) with the hardcoded source port number 17576, 80 and 8080 with source port 18088, 52869 (service Universal Plug an Play) with source port 18344, and 37215 (Huawei HG532 router port) with source port 17832, targeting the whole range of IP addresses. We observe that both hosts are performing network scanning: targeting specific ports known for their vulnerabilities and targeting the whole range of IPv4 addresses. We notice also some differences between both hosts: they did not run the same Operating System, as the range for ephemeral ports is different. Also, we do not observe C&C communications for the second host.

*A. Blaise is with Thales, Gennevilliers, and Sorbonne Université, CNRS LIP6, Paris, France. Email: agathe.blaise@lip6.fr.

†M. Bouet and V. Conan are with Thales, Gennevillers, France. Email: {name.surname}@thalesgroup.com

‡S. Secci is with Cnam, Cedric, 75003 Paris, France. Email: stefano.secci@cnam.fr

Fig. 1c shows the fingerprinting of an infected host (`10.0.2.111`) from capture 112_2 from Malware Capture Facility Project. The host has normal TCP and UDP connections, but we also observe a ICMP DoS using a large variety of ICMP codes (noted destination ports in the graph) and targeting the DNS servers hosted at `8.8.4.4` and `8.8.8.8`. Its behavior is similar to hosts from scenarios #4, 10, 11 from CTU-13 performing ICMP DoS.

2 Importance of features selection

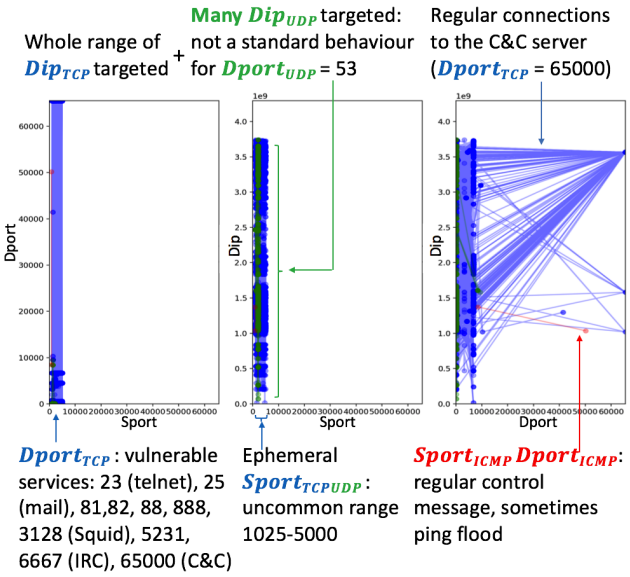
To better understand the specificities of bots communications that enable to detect them, we observe the most meaningful features in the classification process.

Once the linear SVM is fit to the data, with 256 regular bins (and the C parameter set to 100 according to Gridsearch), it creates a line or a hyperplane which separates the data into classes. It uses support vectors to maximize the distance between two classes, and the weights obtained represent the vector coordinates which are orthogonal to the hyperplane and their direction indicates the predicted class.

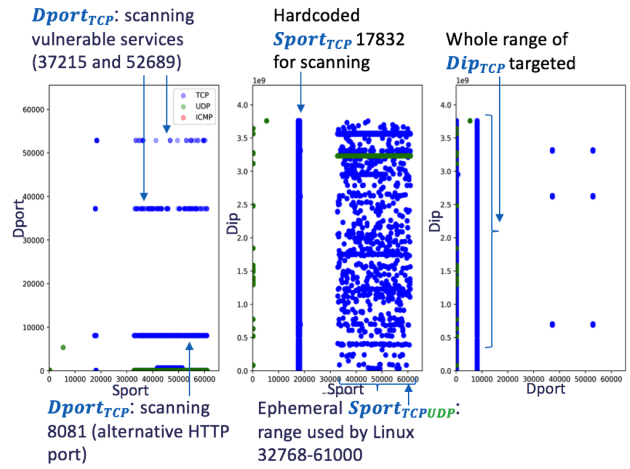
Table 1 shows the ranking of the most meaningful attributes with, for each of them, the number of bins which have a weight not null, the sum of all per-bin weights, and the mean weight per bin. $\text{Type}_{\text{ICMP}}$ has the most significant impact of all attributes. It represents the type of ICMP message which, for a botnet, is often set to 3 for "Destination Unreachable" or to 8 for "Echo Request". The second and the third most important ones are Dip_{TCP} and Dip_{UDP}, as during a botnet spam or scan, nearly all destination IP addresses are targeted instead of some selected ASes. $\text{Sport}_{\text{UDP}}$ and $\text{Dport}_{\text{UDP}}$ follow, actually UDP is often used only for DNS, and in case of a botnet is very differently used. Finally, the values of $\text{Sport}_{\text{TCP}}$, $\text{Dport}_{\text{TCP}}$, $\text{Dport}_{\text{ICMP}}$ and Dip_{ICMP} have nearly no impact on the results.

Attribute	**Number of bins with a weight > 0**	**Sum of weights for all bins**	**Mean weight per bin**
$\text{Type}_{\text{ICMP}}$	148	0.8517	0.0058
Dip_{TCP}	60	0.5328	0.0089
Dip_{UDP}	52	0.3371	0.0058
$\text{Sport}_{\text{UDP}}$	41	0.1663	0.0041
$\text{Dport}_{\text{UDP}}$	100	0.1264	0.0013
$\text{Sport}_{\text{TCP}}$	43	0.0046	0.0001
$\text{Dport}_{\text{TCP}}$	10	0.0327	0.0032
$\text{Code}_{\text{ICMP}}$	50	0.0014	$2.8843 \cdot 10^{-5}$
Dip_{ICMP}	5	0.0013	0.0003

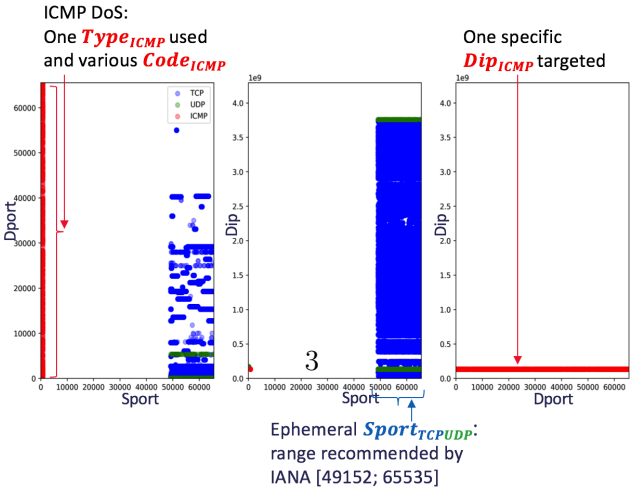
Table 1: Ranking of the attributes according to their importance in the classification, for *BotFP-SVM* with 256 regular bins.



(a) Host `147.32.84.165` from scenario #1 of CTU-13 performing network scanning.



(b) Host `192.168.100.111` from scenario #17 of IoT-23 performing port scanning.



(c) Host `10.0.2.111` from capture 112_2 of Malware Capture Facility Project.

Figure 1: Fingerprinting of infected hosts.