Online detection of Botnets on Network Flows using Stream Mining

Victor G. Turrisi da Costa[1], **Bruno Bogaz Zarpelão**[1],
Rodrigo Sanches Miani[2], **Sylvio Barbon Junior**[1]

[1]Computer Science Department – State University of Londrina (UEL)
Londrina – PR – Brasil

[2]School of Computer Science (FACOM) – Federal University of Uberlândia
Uberlândia – MG – Brazil

{victorturrisi, brunozarpelao, barbon}@uel.br,

miani@ufu.br

Abstract. *The threat posed by botnets of infecting a large number of devices and using them together to perform several malicious actions has become a growing issue to the Internet security. One way to deal with it is to have methods able to correctly identify those botnets and then run necessary countermeasures. Many approaches using machine learning (ML) have been proposed over the years to cope with botnet detection. Nonetheless, the algorithms commonly employed cannot adapt to new data without significant effort. In this sense, a ML research topic referred to as stream mining may be a solution. Stream mining algorithms are specially tailored to learn incrementally with new instances, without consuming significant memory or time. This work proposes an approach using the Very Fast Decision Tree, a classification algorithm used on stream mining that can learn incrementally when needed, to identify botnets by observing network flows. When evaluating the approach on multiple scenarios with different botnets, we were able to achieve high performance metrics on the majority of scenarios, while using a significantly low number of labelled instances.*

1. Introduction

The Internet security has been a growing concern with the popularisation of its services and the continuous increase in the number of active users. Among the major concerns, botnets are one of the most widespread and hazardous threats. They consist of a group of infected devices that work together to perform a vast range of malicious activities, such as, Distributed Denial of Service (DDoS) attacks, stealing sensitive information and generating spam [Silva et al. 2013, Kidmose et al. 2016, Costa et al. 2017]. A botnet has three main components that often characterises it, the bots, the botmaster and the Command and Control (C&C) infrastructure. The bots correspond to the devices that are compromised by a bot malware (a malware that takes control of this device). The botmaster consists of malicious users that own a botnet and control its bots. Lastly, the C&C infrastructure describes the way the bots and the botmaster communicate among themselves.

One common approach to deal with botnets consists of detecting suspicious activities using Intrusion Detection Systems (IDSs), and then perform the necessary security measures [Silva et al. 2013, Grill and Pevnýa 2016]. IDSs often rely on machine

learning (ML) algorithms to learn malicious behaviours and use the modelled patterns in detection task. In this sense, these algorithms need to be able to continuously adapt to identify new botnets [Hammerschmidt et al. 2017]. Over the years, many works have proposed solutions for botnet detection, e.g., [Livadas et al. 2006, Saad et al. 2011, Stevanovic and Pedersen 2014, Jianguo et al. 2016], but new botnets have developed new mechanisms to make them more robust and sophisticated [Silva et al. 2013], greatly increasing the difficulty of identifying and disrupting them. Hence, solutions capable of quickly adapting to changes in botnet behaviour are required.

Stream mining, as described in [Domingos and Hulten 2000, Gama et al. 2010, Krawczyk et al. 2017], is a branch of ML that views data as a possibly infinite stream and, by doing so, algorithms are specially designed to learn in an online fashion, one instance at a time. Additionally, memory and time limitations are specifically studied in this area, so that the algorithms also need to be able to cope with these challenges [Domingos and Hulten 2000, Gama et al. 2010, Krawczyk et al. 2017]. Many algorithms were specially developed to handle data streams, with the Very Fast Decision Tree (VFDT) [Domingos and Hulten 2000] being largely used across many fields. This algorithm is a modification of traditional decision trees with the ability to learn one instance at a time by using statistical properties, while it also consumes less memory compared to its batch counterparts, and is significantly faster than them.

With this in mind, we propose to detect botnets in an online manner by analysing network flows employing the VFDT. Nonetheless, other algorithms for data streams could also be employed, but we chose the VFDT because it is a very memory conservative algorithm, with the possibility of using it in low memory devices. Our approach can be easily updated when novel botnets become known, without any need to retrain the model from scratch. In other words, it is not required to store all labelled data used to train the model. Additionally, we also considered the cost of labelling instances, limiting the number of labels available for the proposed solution. The solution was evaluated on eleven different scenarios of the CTU dataset [García et al. 2014], which contains labelled network flows from legitimate and malicious botnet traffic.

The main contributions of this work are:

1. We provide a data stream based approach for botnet detection that is also capable of being updated online without training from scratch;
2. We perform empirical tests of the proposed approach on all scenarios of the CTU dataset, achieving high performance on the majority of scenarios, even with limited number of labelled instances.

The remainder of this paper is organised as follows. Section 2 presents botnet detection related work. Section 3 contains theoretical foundation about traditional machine learning and machine learning on data streams. In Section 4, the proposed approach is presented. Our experiments and results are contained in Section 5. The conclusion of this paper is presented in Section 6.

2. Related Work

This section presents previous work that performed botnet detection on the CTU dataset [García et al. 2014]. Wang and Paschalidis [Wang and Paschalidis 2016] proposed a two-phase approach for botnet detection. Their idea consists of first detecting and collecting

network anomalies associated with the presence of a botnet by using statistical measures, and then identifying the bots by analysing these anomalies. The detection phase quantifies and monitors flow-level data as histograms, which are then used to construct graphs of highly interactive nodes (hosts that communicate with each other). They evaluated their approach on scenarios 1, 2, 6, 8 and 9 of the CTU dataset, obtaining very low recall (0.0734 on mean) and satisfactory precision (0.7360).

Jianguo et al. [Jianguo et al. 2016] proposed to detect botnet behaviour data by using traditional machine learning (ML) algorithms and network flow features. They evaluated their work in two malicious datasets (scenario 2 of the CTU dataset and a dataset containing the Zeus botnet) and one dataset from the a laboratory in Berkeley that contained only legitimate data. They obtained very high performance metrics, even while using a logistic regression, but they employed a fully labelled dataset, in the sense that a large number of labelled instances were used. Additionally, they did not use legitimate nor background traffic from the CTU dataset, so a direct comparative link between the performance achieved in their work and ours is impossible to make.

Hammerschmidt et al. [Hammerschmidt et al. 2016] created a solution using finite state machines and network flow features to detect devices infected by bot malwares. The authors state that recent methods that deal with botnet detection work in a batch setting, which creates time and memory constraints. In this sense, they propose to adequate their approach to deal with network data in a stream setting. When evaluating their approach on scenarios 10, 11 and 12 of the CTU dataset, they achieved high host identification rates, however, their solution does not identify malicious botnet flows, requiring a high number of flows to perform the detection task. Hammerschmidt et al. [Hammerschmidt et al. 2017] is an extension of the same paper with additional theoretical links to the data stream mining field.

Ijaz et al. [Ijaz et al. 2017] proposed a genetic algorithm based solution to detect malware attacks. They evaluated the performance of their approach at detecting each type of attack present on the KDD dataset [Bay et al. 2000] and two attack types from scenario 2 of the CTU dataset. On the CTU dataset, they obtained satisfactory results on both attacks, however, their approach lacks online capacity, needing many round of optimisation and batch data for the genetic algorithm.

Chen et al. [Chen et al. 2017] evaluated three unsupervised machine learning algorithms, self-organising map (SOM), local outlier factor (LOF), and k-NN outlier, to build a normal behaviour profile to be used for botnet detection. They evaluated their solution on the CTU dataset. By adding derived features to the original ones, they could obtain a very high detection rate (91.3%), but with inherited high false positive rates due to the nature of the algorithms employed. Nonetheless, a very significant performance, in the sense that the algorithms employed did not have access to any labelled data.

Khanchi et al. [Khanchi et al. 2017] proposed an approach using genetic programming and ML on data streams to detect botnet flows. The authors also imposed a budget to restrict the amount of labelled data their approach had access to. They evaluated the approach on the CTU dataset and obtained a relatively high performance using a considerable small number of labelled instances. Nonetheless, authors imply that labelling legitimate instances have the same cost than botnet instances, which does not hold true

in reality, since it is easy to gather network data from proven legitimate servers, such as, from Google. In this sense, more legitimate data could be used without greatly increasing labelling costs.

Unlike solutions that use traditional ML algorithms, our solution is able to learn continuously, without the need to store all labelled data and use it to retrain the solution when new data becomes available, adapting to novel botnets more easily. Likewise, the prediction model is updated on the fly, different from the traditional ML that process past data, even obsolete, to obtain a model. Additionally, ML algorithms for data streams consume significant less memory than their traditional batch counterparts, which makes possible to deploy it in different network components. Some works proposed solutions that work in an online setting, but there is still large room for improvement.

3. Machine Learning and Data Stream Mining

Machine learning techniques are able to model knowledge based on data. Afterwards, those models can be applied in a range of tasks such as classification problems. In recent scenarios, the massive amount of data available demands specialised algorithms capable of learning from it [Gama et al. 2010, Krawczyk et al. 2017]. To address this issue, data stream mining algorithms, in contrast to traditional ML algorithms, are specially tailored to be able to learn incrementally without the need to store all data, in fact, processing data in the form of a continuous and infinite stream. Additionally, the evolving nature of data streams throughout the time produces a phenomenon called concept drift. Concept drifts are related to changing the behaviour of patterns formed along the time, consequently, old data becomes obsolete and can negatively impact the performance of the predictive model [Gama et al. 2010, Krawczyk et al. 2017].

When dealing with network traffic analysis, a huge amount of information is available and so, the constraints to deal with it align with the ones of ML on data streams. One of the most used algorithms in data stream ML is called Very Fast Decision Tree (VFDT), proposed by [Domingos and Hulten 2000]. It is a modification of the standard decision trees applied in traditional ML tasks to address incremental learning, as well as time and memory limitations. Unlike traditional decision trees, the VFDT can learn one instance at a time by storing sufficient information by taking advantage of a statistical property called Hoeffding Bound (HB). Moreover, the prediction model obtained by this algorithm is based on lightweight statistics about the instances instead of the instances themselves.

These statistics are a simple counting procedure to handle nominal attributes. On the other hand, numerical attributes are computed by more sophisticated lightweight techniques. The most applied strategy is the Gaussian estimators [Pfahringer et al. 2008], which uses very few memory and result in a predictive performance as close as if the VFDT was trained using all instances at the same time. The aforementioned characteristics support the VFDT to compute metrics normally used across all decision trees, as information gain (IG).

Given a variable x, whose value fluctuates between a range R, that was independently observed n times and presented an observed mean of $\overline{x}$, the HB states that the true mean of this variable is at least $\overline{x} - \epsilon$ when n grows to infinity with statistical probability

$1 - \delta$, where

$$\epsilon = \sqrt{\frac{R^2\, ln(\frac{1}{\delta})}{2\, n}}. \quad (1)$$

We can check if the best feature is indeed the best if its IG value is greater than the second highest IG value by at least ϵ (computed using Equation 1).

To increase the performance of the VFDT, Gama et al. [Gama et al. 2003], proposed the use of functional leaves. Functional leaves implement an additional prediction layer, where each leaf applies an adaptive Naive Bayes (NB) to perform predictions. This adaptive NB algorithm uses the prediction of a NB when it has superior accuracy than a most common classical approach. Otherwise, it is assigned the prediction as the class most present in the leaf. It is important to notice that this accuracy is computed incrementally and that the same leaf may alter between using NB or most common. At last, the VFDT has a competitive performance in comparison to decision trees fashioned to batch training from conventional datasets [Domingos and Hulten 2000]

4. Proposed Approach

We propose a network-based approach to detect botnets by analysing bidirectional network flows in an online manner using the VFDT. In this sense, the approach is able to be incrementally updated when new instances become available. Additionally, memory and time constraints are also considered, resulting in a very lightweight solution. The approach can be divided into two main steps: instance creation and botnet detection.

The objective of the instance creation step is to generate, from network packets, the data instances that will be processed by VFDT. The instance creation starts with a stream of network packets that is organised into bidirectional flows by a router. In practice, any router that supports RFC 5103 [Trammell and Boschi 2008], which defines implementation details for routers to be able to generate bidirectional network flows, can be used. Bidirectional flows differ from conventional flows in the sense that the equipment is capable of inferring the direction of the communication among hosts. The features that compose each instance are extracted from each outputted flow. In this way, each flow is transformed into a single data instance to be processed by VFDT. The data instances are then sorted by their end times.

All features employed in this work are present in Table 1. We also ignored features related to IP addresses, since they are not related to the behaviour of the botnet and would generate some bias. In this case, the solution would just associate malicious traffic to the IP addresses of the infected devices and not detect botnet actions based on network behaviour. Additionally, we derived a new feature based on the mean amount of bytes per packet. It is interesting to note that this new feature is independent of data from other network flows and can be easily implemented.

At first, in the botnet detection step, the VFDT is initialised with the first instance available. After this point, the algorithm is able to perform predictions at any time. However, based on a single sample, all predictions will concern to the single class learned. For each new instance processed, the VFDT tries to predict its class, and outputs this prediction. In this sense, the VFDT will answer if that network flow corresponds to a botnet instance or not. Additionally, this prediction is used to computed the performance statis-

Table 1. Description of features used.

Feature	Description
Duration	Duration of the flow in seconds.
Protocol	Type of protocol used on the flow, e.g., TCP and UDP.
Source port	The source port of the connection.
Destination port	The destination port of the connection.
Direction	Direction of the flow.
State	Transaction state.
sTos	Type of service from source to destination.
dTos	Type of service from destination to source.
Total packets	Total number of packets on the flow.
Total bytes	Total bytes on the flow.
Source Bytes	Total bytes from source to destination.
Extended Feature	
Mean bytes per packet	The mean amount of bytes per network packet

tics of the approach. If the ground truth label of that instance is available, the instance is used to update the VFDT model. The update works in the following manner. First, the new instance is sorted to a leaf of the VFDT. Then, the statistics about that instance are incorporated into the node. Lastly, the tree checks if it is possible to split that leaf, according to the HB, and splits it if needed. Since the VFDT is able to learn in an online fashion, there is no need to retrain it from scratch, and so, each instance processed is discarded after the VFDT learned from it. The ground truth labels are provided by a human expert, who can insert new knowledge into the VFDT. Given the fact that the VFDT can be updated with a single instance at a time, labelled data can be provided at any rate. Likewise, there is no specific order in which instances should be provided.

A full overview of the approach can be seen in Figure 1.

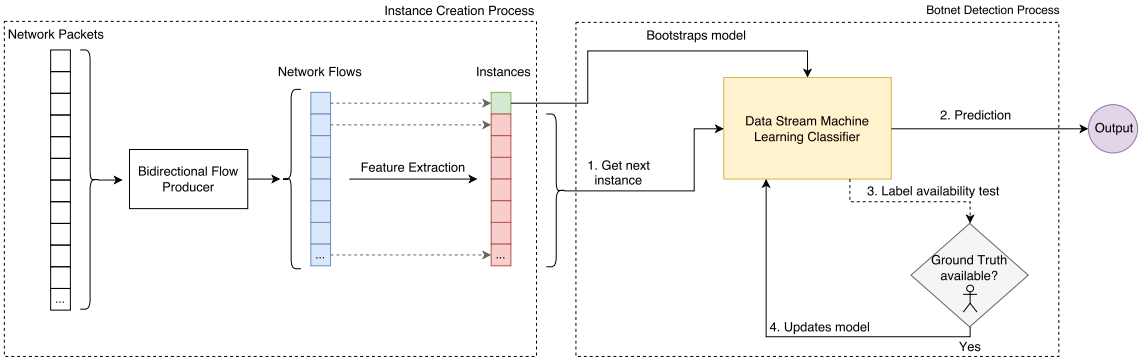

Figure 1. Proposed Approach.

5. Experiments and Evaluation

To evaluate our approach in a real-world scenario capable to support the data stream with incidence of botnet behaviour, we chose the CTU dataset [García et al. 2014]. Many recent proposals were also evaluated with this dataset [Wang and Paschalidis 2016, Jianguo et al. 2016, Hammerschmidt et al. 2016, Hammerschmidt et al. 2017, Ijaz et al. 2017]. It contains 13 different network scenarios that vary in quantity and type of botnets. Each scenario contains legitimate and back-

ground traffic in the form of labelled bidirectional network flows. The last two differ in the sense that the first is traffic between confirmed uninfected machines and trustful sources, i.e., a Google's server, while the other refers to traffic that was not confirmed as legitimate nor botnet.

In Table 2, the amount of flows per type, for each scenario, are presented.

Table 2. Description of scenarios from the CTU dataset. Adapted from [García et al. 2014]

Scenario	Total Flows	Botnet Flows	Legitimate Flows	Background Flows
CTU 1	2,824,636	40,961 (1.44%)	30,387 (1,07%)	2,753,290 (97.47%)
CTU 2	1,808,122	20,941 (1.15%)	9,120 (0.50%)	1,778,061 (98.33%)
CTU 3	4,710,638	26,822 (0.56%)	116,887 (2.48%)	4,566,929 (96.94%)
CTU 4	1,121,076	2,580 (0.23%)	25,268 (2.25%)	1,093,228 (97.51%)
CTU 5	129,832	901 (0.693%)	4,679 (3.60%)	124,252 (95.7%)
CTU 6	558,919	4,630 (0.82%)	7,494 (1.34%)	546,795 (97.83%)
CTU 7	114,077	63 (0.05%)	1,677 (1.47%)	112,337 (98,47%)
CTU 8	2,954,230	6,127 (0,21%)	72,822 (2.46%)	2,875,282 (97.32%)
CTU 9	2,753,884	184,987 (6.68%)	43,340 (1.57%)	2,525,565 (91.70%)
CTU 10	1,309,791	106,352 (8.11%)	15,847 (1.20%)	1,187,592 (90.67%)
CTU 11	107,251	8,164 (7.60%)	2,718 (2.53%)	96,369 (89.85%)
CTU 12	325,471	2,168 (0.66%)	7,628 (2.34%)	315,675 (96.99%)
CTU 13	1,925,149	40,003 (2.07%)	31,939 (1.65%)	1,853,217 (96.26%)

Additionally, we also present the botnet types per CTU scenario with a brief description of their C&C protocols and their main malicious actions in Table 3.

Table 3. Botnet types per scenario. Adapted from [Khanchi et al. 2017]

Botnet Family	Scenarios	C&C Protocol	Main Malicious Actions
Neris	1, 2 and 9	IRC	Spam, fraud, and network scanning
Rbot	3, 4, 10 and 11	IRC	DDoS
Virut	5 and 13	HTTP	DDoS, spam and data theft
Menti	6	IRC	Data theft
Sogou	7	HTTP	Spam and data theft
Murlo	8	IRC	Network scanning
NSIS.ay	12	P2P	Data theft

The CTU dataset divides botnet flows into two different categories: botnet flows and C&C flows. The first corresponds to network data from infected devices which were being used as bots. The latter consists of communication between the infected devices and the C&C infrastructure of the botnet. We treated both types as data with a botnet label. Additionally, we tailored another class, called background, which is formed by CTU's legitimate and background traffic. It is important to note that to create a reliable evaluation focused on botnet detection, our background class was not tagged as legitimate, since some botnet instances may have gone unnoticed by the filters employed to the CTU dataset and been classified as background.

We did not evaluate the scenarios 5 and 7 since they presented an unfit number of botnet flows, 901 and 63, respectively. Indeed, different from some prior work [Wang and Paschalidis 2016, Jianguo et al. 2016, Hammerschmidt et al. 2016, Hammerschmidt et al. 2017, Ijaz et al. 2017], we proposed to test our approach on all

scenarios with a significant number of instances (more than one thousand).

When carrying out the experiments, we sorted the flows by their end time. We computed this time by using their start time and adding the duration of the flow.

To measure the performance of our approach, we used precision, recall and false positive rate (FPR). These metrics can be easily computed by the following equations:

1. $precision = \frac{TP}{TP+FP}$, which represents the percentage of botnet-classified flows that are indeed botnets;
2. $recall = \frac{TP}{TP+FN}$, which measures how effective the classifier is in identifying all botnets instances;
3. *False positive rate*: $FP/(FP + TN)$

where, TP, TN, FP, FN represent the true positives, true negatives, false positives and false negatives, respectively.

Accuracy is normally used across many works, but was not employed here due to the high imbalance rate among background and botnet instances, which would result in an accuracy of around 95% if the approach predicted all instances as background. Additionally, to measure the memory consumption of the approach, we computed the size in bytes of the VFDT constructed for each scenario.

When evaluating data stream algorithms, unlike traditional ML algorithms, some validation strategies such as hold-out and cross-validation cannot be employed. Thus, we employed a method called prequential testing [Gama et al. 2010, Domingos and Hulten 2000, Krawczyk et al. 2017]. This consists of presenting an instance, without its label, to the classifier being evaluated and asking for a prediction. After this, the prediction is stored and used to compute the performance metrics desired. When all performance statistics are updated, the true label of that instance is presented to the classifier, which is then able to learn from it. Due to the fact that labelling instances is very costly for a human expert, we limited the number of instances with ground truth that were available for the classifier. To do this, we presented the ground truth for all instances (both botnet and background) until a number of n botnet instances. After that, no instance had its true label presented to the classifier and, in this sense, the classifier stopped updating. In other words, the classifier learned from all instances until it had learned from a number of n botnet instances. We varied this n value from 100 to 10000 with a step of 100 on scenarios 1, 2, 3, 4, 6, 8, 10, 11 and 12. For scenario 9, the range of n was from 100 to 40000, and for scenario 13 from 100 to 20000, both with the same step of 100. We used different values for these two scenarios because the performance achieved by using the other range was not satisfactory and presented room for improvement, since there was still a lot of botnet instances on the scenarios. However, it is important to note that the classifier was evaluated using all instances.

All modules of the solution were implemented in Python version 3.6 and the experiments were carried on a desktop PC with a core Intel i7-6700K and 16 GB of RAM.

5.1. Results and Discussion

At first, we will present the mean numerical performance of our approach on the scenarios of the CTU dataset. After that, performance along the training process is presented. Then,

we present a discussion about the performance of the solution according to botnet family. Lastly, a summarising discussion about the results are presented.

In Table 4, the numeric results for the best performing limit of n botnet instances are presented. We denote best as the combination of a low value of n together with the predictive performance achieved. It is possible to observe that the number of instances used to achieve a significant predictive performance greatly varies among datasets. This means that some botnets are inherently difficult to identify using very few instances, while others, scenarios 4, 6, 8, 10 and 11, presented very distinct behaviour from background data. We can observe that in all scenarios the FPR was very low, with scenario 9 having the highest FPR among them, with a value of 0.7%. Lastly, since the memory costs of a solution are important, we report the number of bytes each tree consumed after all training was done. Since the VFDT presents no pruning mechanisms, which means that no leaves are ever removed from the tree, the size reported corresponds to the maximum size. The trees presented a mean size of 0.751 MB, with the lightest tree having 0.272 MB and the most memory consuming tree having 1.508 MB in size. The memory consumed is very low and can even be ignored in most applications. Even if we hold all trees in memory, they would use mere 8.262 MB, being suitable for lightweight or embedded devices.

Table 4. Performance of the VFDT for the best maximum limit of botnets.

Dataset	Labelled botnet instances (n)	Labelled background instances	Precision	Recall	FPR	Size(MB)
CTU 1	16.3% (6700)	49.9% (1389655)	0.927	0.917	0.0011	1.104
CTU 2	41.1% (8600)	25.8% (461868)	0.675	0.696	0.0039	0.633
CTU 3	31.7% (8500)	75.6% (3545024)	0.962	0.967	0.0000	1.197
CTU 4	65.9% (1700)	90.1% (1007538)	0.967	0.940	0.0000	0.623
CTU 6	28.1% (1300)	30.1% (166930)	0.955	0.959	0.0003	0.272
CTU 8	31.0% (1900)	43.0% (1269715)	0.934	0.946	0.0001	0.707
CTU 9	15.2% (28200)	58.5% (1114439)	0.924	0.921	0.0072	1.139
CTU 10	0.4% (500)	40.5% (487595)	0.999	0.996	0.0000	0.380
CTU 11	2.4% (200)	86.6% (85846)	0.992	0.995	0.0006	0.295
CTU 12	78.4% (1700)	72.6% (234991)	0.664	0.422	0.0014	0.404
CTU 13	49.7% (19900)	40.2% (758694)	0.635	0.451	0.0055	1.508

In Figures 2 and 3, the final mean precision and recall values obtained for each CTU scenario are presented. In the x-axis, we have the limit number n, in percentage, according to the number of botnet instances on that dataset, of botnets instances that limited the update process. Likewise, the percentage of background instances which had their ground truth labels presented to the VFDT is also reported in the same axis. On the y-axis, we have the percentage values. The green line in the plots corresponds to the best performing limit of n botnets in each scenario.

It is possible to see that for scenarios 1, 3, 4, 6, 8, 9, 10 and 11 the approach was able to achieve very high values of precision and recall (greater of equal to 90% on both metrics). A interesting behaviour is present in scenario 3. Around a limit of 5000 botnet instances, the VFDT presented high precision and recall values, but increasing this value negatively impacted its performance until a maximum of around 8500 instances, in which the performance stabilised. From the limit of 5000 to 8500 botnets, some patterns in botnet actions may have changed and by learning from it, the performance of the solution was compromised. In this sense, further analysis of the scenario may be interesting to

detect those patterns and study them.

Scenario 2 had a reasonable performance, achieving a metrics around 70% when using a limit around 8200 and 8600 botnet instances. Lastly, scenarios 12 and 13 presented the worst performances. On the first, the low number of botnet instances may have provided too little information for the VFDT. On the latter, even using a very high limit of instances, performance was still very low and further analysis on the characteristics of the botnets in that scenario may assist at increasing performance. Additionally, scenario 12 contains the only P2P botnet in the CTU dataset. Using the P2P protocol for its C&C infrastructure, provide that botnet great flexibility and robustness [Le et al. 2016], obfuscating its activities. Likewise, the usage of a HTTP C&C infrastructure in the botnet in scenario 13, difficult detection, as the actions of this botnet can be hidden amongst the huge number of normal web traffic, as HTTP is a common and widely-used protocol for home, educational and corporate networks [Eslahi et al. 2015].

By taking into consideration the botnet types of each scenario, we are able to infer additional information about the performance of our approach. The solution was able to identify Neris botnet after a few thousand instances for scenarios 1 and 9, however, due to background data in scenario 2, the performance was greatly compromised. On the other hand, Rbot botnet was very easy to detect on all scenarios. Virut botnet was only present in scenario 13, since we removed scenario 5 due to the lack of enough botnet instances, however, its behaviour was the hardest to identify, together with NSIS botnet. Lastly, Menti botnet also presented a very distinguishable behaviour. In general, the approach was able to easily identify IRC botnets despite their very diverse set of malicious actions, struggling to identify botnets based on other C&C protocols.

The approach consumes very few memory and also obtained high performance metrics across the majority of scenarios. Additionally, it can be updated in real time when new labelled instances become available, increasing its applicability in the real world. If we had employed traditional ML algorithms, the solution would be limited to the labelled instances used at the time of training. Likewise, if new labelled instances become available, the algorithm would have needed to be retrained from scratch.

6. Conclusion and Future Work

Botnets are an existing and growing threat to the Internet security and so, approaches to deal with them are needed. One way to tackle this issue is to detect botnet actions and infected devices to then provide the necessary security measures.

In this paper we presented an approach to deal with botnet detection in an online manner, which can even be extended using other algorithms or updating techniques. The approach is able to classify unknown network flows as botnet or not, being lightweight and with the additional property that it can be updated on the fly.

To evaluate this approach, we performed empirical evaluations on the CTU dataset, which contains different scenarios with network traffic data from botnets, limiting the maximum number of instances the approach had access to its ground truth label. We obtained high prediction performance metrics (precision, recall, and FPR) in the majority of scenarios. Likewise, the memory consumption of the VFDT was very low, with a mean size of 0.272 MB on all scenarios.

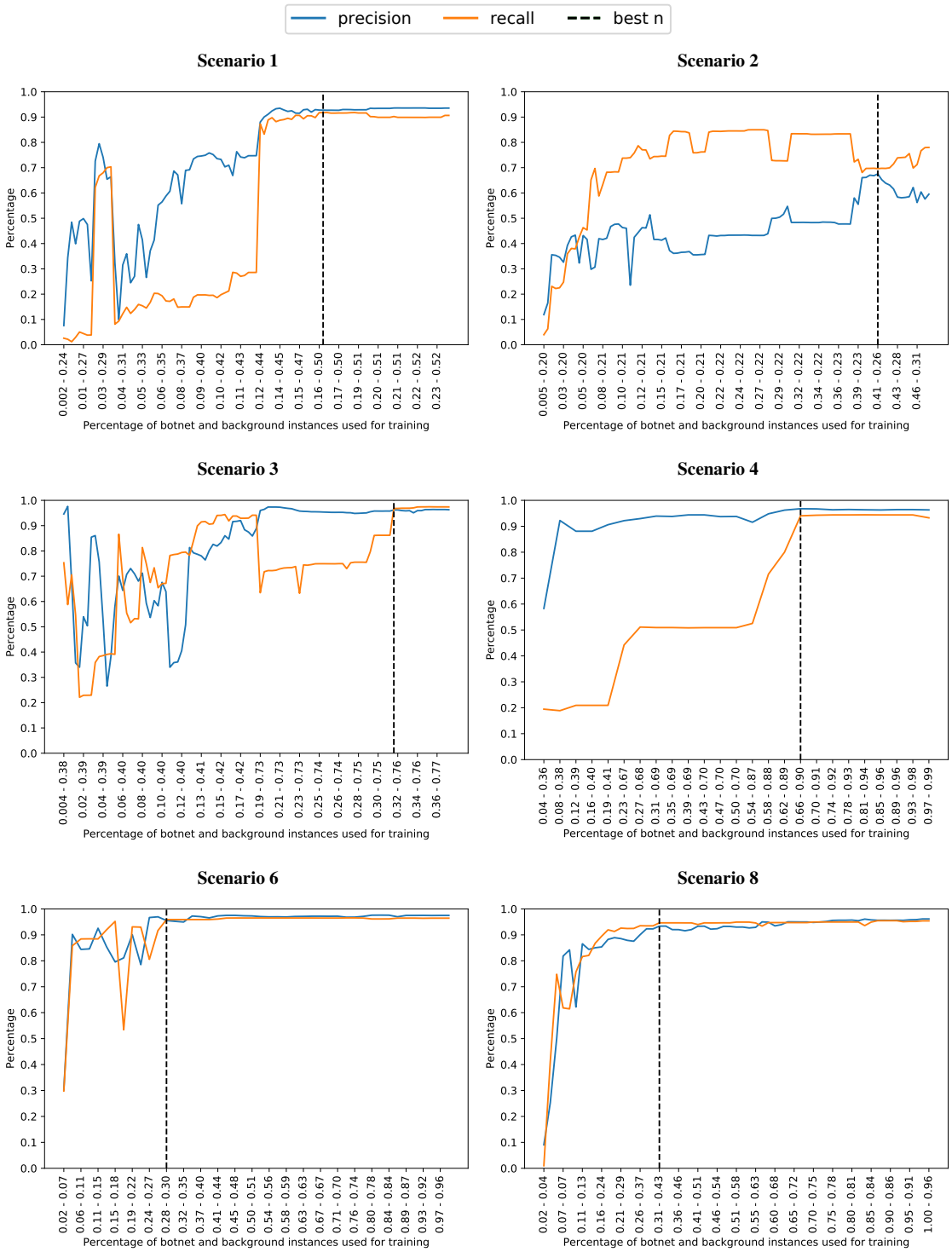

Figure 2. Mean precision and recall after prequentially learning until a maximum number of botnets instances are reached. The x-axis represents the total amount of botnet and background instances used to update the VFDT in percentage.

When used in the real-world, the proposed solution would require a router which implemented the RFC 5103 to generate the bidirectional flows. These flows could be processed in any device on the same network, or even remotely, generating alerts for the network specialist.

Figure 3. Continuation of Figure 2.

As future work, we intend at testing ensembles solutions to tackle the same problem. Additionally, we expect to use active learning to make labelling process more efficient.

References

Bay, S. D., Kibler, D., Pazzani, M. J., and Smyth, P. (2000). The uci kdd archive of large data sets for data mining research and experimentation. *SIGKDD Explor. Newsl.*,

2(2):81–85.

Chen, W., Luo, X., and Zincir-Heywood, A. N. (2017). Exploring a service-based normal behaviour profiling system for botnet detection. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 947–952.

Costa, V. G. T., Barbon Jr, S., Miani, R. S., Rodrigues, J. J. P. C., and Zarpelão, B. B. (2017). Detecting Mobile Botnets Through Machine Learning and System Calls Analysis. *Proceedings of the 2017 IEEE International Conference on Communications (ICC)*, pages 917–922.

Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80.

Eslahi, M., Rohmad, M. S., Nilsaz, H., Naseri, M. V., Tahir, N. M., and Hashim, H. (2015). Periodicity classification of http traffic to detect http botnets. In *2015 IEEE Symposium on Computer Applications Industrial Electronics (ISCAIE)*, pages 119–123.

Gama, J., Rocha, R., and Medas, P. (2003). Accurate decision trees for mining high-speed data streams. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03*, (January 2003):523.

Gama, J., Rodrigues, P. P., Spinosa, E., and Carvalho, A. (2010). Knowledge Discovery from Data Streams. *Web Intelligence and Security - Advances in Data and Text Mining Techniques for Detecting and Preventing Terrorist Activities on the Web*, pages 125–138.

García, S., Grill, M., Stiborek, J., and Zunino, A. (2014). An empirical comparison of botnet detection methods. *Computers & Security*, 45(Supplement C):100 – 123.

Grill, M. and Pevnýa, T. (2016). Learning combination of anomaly detectors for security domain. *Computer Networks journal*, 107:24–43.

Hammerschmidt, C., Marchal, S., State, R., Pellegrino, G., and Verwer, S. (2016). Efficient Learning of Communication Profiles from IP Flow Records. *Proceedings - Conference on Local Computer Networks, LCN*, pages 559–562.

Hammerschmidt, C., Marchal, S., State, R., and Verwer, S. (2017). Behavioral clustering of non-stationary IP flow record data. *2016 12th International Conference on Network and Service Management, CNSM 2016 and Workshops, 3rd International Workshop on Management of SDN and NFV, ManSDN/NFV 2016, and International Workshop on Green ICT and Smart Networking, GISN 2016*, pages 297–301.

Ijaz, S., Hashmi, F. A., Asghar, S., and Alam, M. (2017). Vector Based Genetic Algorithm to optimize predictive analysis in network security. *Applied Intelligence*.

Jianguo, J., Qi, B., Zhixin, S., Wang, Y., and Lv, B. (2016). Botnet Detection Method Analysis on the Effect of Feature Extraction. *Trustcom/BigDataSE/ISPA, IEEE*, pages 1884–1890.

Khanchi, S., Vahdat, A., Heywood, M. I., and Zincir-Heywood, A. N. (2017). On botnet detection with genetic programming under streaming data label budgets and class imbalance. *Swarm and Evolutionary Computation*, (August):1–18.

Kidmose, E., Stevanovic, M., and Pedersen, J. M. (2016). Correlating intrusion detection alerts on bot malware infections using neural network. *2016 International Conference on Cyber Security and Protection of Digital Services, Cyber Security 2016*.

Krawczyk, B., Minku, L., Gama, J., and Stefanowski, J. (2017). Ensemble learning for data stream analysis: A survey. *Information*, pages 1–86.

Le, D. C., Zincir-Heywood, A. N., and Heywood, M. I. (2016). Data analytics on network traffic flows for botnet behaviour detection. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7.

Livadas, C., Walsh, R., Lapsley, D., and Strayer, W. T. (2006). Using machine learning techniques to identify botnet traffic. *Proceedings - Conference on Local Computer Networks, LCN*, (1):967–974.

Pfahringer, B., Holmes, G., and Kirkby, R. (2008). Handling numeric attributes in hoeffding trees. In *Proceedings of the 12th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD'08, pages 296–307, Berlin, Heidelberg. Springer-Verlag.

Saad, S., Traore, I., Ghorbani, A., Sayed, B., Zhao, D., Lu, W., Felix, J., and Hakimian, P. (2011). Detecting P2P botnets through network behavior analysis and machine learning. *2011 9th Annual International Conference on Privacy, Security and Trust, PST 2011*, pages 174–180.

Silva, S. S. C., Silva, R. M. P., Pinto, R. C. G., and Salles, R. M. (2013). Botnets: A survey. *Computer Networks*, 57(2):378–403.

Stevanovic, M. and Pedersen, J. M. (2014). An efficient flow-based botnet detection using supervised machine learning. *2014 International Conference on Computing, Networking and Communications (ICNC)*, pages 797–801.

Trammell, B. and Boschi, E. (2008). Bidirectional flow export using ip flow information export (ipfix). RFC 5103, RFC Editor.

Wang, J. and Paschalidis, I. C. (2016). Botnet Detection based on Anomaly and Community Detection. *IEEE Transactions on Control of Network Systems*, 5870(c):1–1.