BACKORDERS: Using Random Forests to Detect DDoS Attacks in Programmable Data Planes

Bruno Coelho
blcoelho@inf.ufrgs.br
Federal University of Rio Grande do Sul
Porto Alegre, Rio Grande do Sul, Brazil

Alberto Schaeffer-Filho
alberto@inf.ufrgs.br
Federal University of Rio Grande do Sul
Porto Alegre, Rio Grande do Sul, Brazil

ABSTRACT

Networks and the services they support form the communication backbone of our society, and it is important that potential Distributed Denial of Service (DDoS) attacks are detected quickly, in order to avoid or minimize the impact they may have on the availability of services. Recent technological advances in programmable networks – specifically the programmability of data planes in switches and routers, have made available new ways of detecting such attacks. By relying on this newfound possibility, this paper proposes the utilization of a Random Forest (RF) to aid in quickly and accurately detecting DDoS attacks in a programmable switch. Random forests utilize several classification trees, each of them for independently classifying an input as one of a set of classes. Here, each decision tree will classify a network flow as potentially malicious, i.e. part of a DDoS attack, or a legitimate user flow. Despite utilizing multiple classification trees to improve accuracy, random forests are relatively lightweight, with each tree requiring few and simple computations to arrive at a classification. Our results show that even small RFs, requiring as few as 63 match+action table entries, can achieve F1-Scores of over 90%.

CCS CONCEPTS

• **Networks** → *Programmable networks*; **In-network processing**; *Denial-of-service attacks*; • **Computing methodologies** → Classification and regression trees.

KEYWORDS

Random forest, programmable data plane, DDos attacks

ACM Reference Format:
Bruno Coelho and Alberto Schaeffer-Filho. 2022. BACKORDERS: Using Random Forests to Detect DDoS Attacks in Programmable Data Planes . In *P4 Workshop in Europe (EuroP4 '22), December 9, 2022, Roma, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3565475.3569074

1 INTRODUCTION

Distributed Denial of Service (DDoS) attacks remain an issue to network operators, as even brief disruptions in service can lead to economic losses [17]. Further, attacks have become more sophisticated, making it harder to differentiate between legitimate and malicious traffic [2]. Flow-based analysis tries to classify a group of packets belonging to the same connection as malicious or not. In order to do this, classifiers may calculate statistical values such as the average, maximum, and minimum packet size or the time in between the arrival of consecutive packets of a specific flow [13].

Current techniques for DoS/DDoS detection generally lack in at least one of accuracy, detection speed, or scalability [19]. This can be partially attributed to systems choosing between utilizing the limited but efficient interfaces provided by traditional switches or utilizing more software-based approaches, allowing more customization at the cost of throughput [16]. However, recent breakthroughs in the field of programmable networks have allowed for further programmability of switches. As opposed to fixed-function switches, programmable switches allow researchers to propose and evaluate new ideas in sufficiently realistic settings [1]. As such, performing the detection of DDoS attacks in programmable data planes is a promising alternative to traditional approaches.

By capitalizing on programmable data planes, in this paper we propose a system capable of efficiently processing individual packets in order to classify their respective flows into likely being part of an attack or genuine user traffic. To achieve this, we design and evaluate BACKORDERS, a system that combines a well studied technique in artificial intelligence, namely Random Forest [5], with recent developments in programmable switches attending to the P4 language specification [1]. Our system is capable of inserting a pre-trained model of Random Forest (RF) into a programmable switch, calculating statistical values and utilizing the RF to accurately classify flows at line rate. As low-bandwidth DDoS attacks tend to be harder to detect [10], our system trains a Machine Learning (ML) model to efficiently learn and detect the patterns present in these attacks, in order to achieve higher accuracy than many of the simple methods used in DDoS detection and defense.

The remaining of this paper is organized as follows. Section 2 presents background information about random forests. Section 3 describes BACKORDERS, detailing the intuition behind the use of random forests and the system architecture. Section 4 presents the conducted experiments. In Section 5, we discuss the related work, and finally Section 6 presents the concluding remarks.

2 RANDOM FORESTS

In this section, we present theoretical background on Random Forests, the Machine Learning technique employed for DDoS detection.

Decision Tree (DT) is a technique that performs a series of chained comparisons ("tests") of values in order to arrive at a decision. The

EuroP4 '22, December 9, 2022, Roma, Italy
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9935-7/22/12…$15.00
https://doi.org/10.1145/3565475.3569074

algorithm branches to a different node depending on the result of the test, repeating this process until it arrives at a leaf node, where a decision is obtained [11]. *Classification Trees* (CTs) are a subset of DTs where the attribute the algorithm is trying to predict (known as the target attribute) is a discrete value, i.e., a class.

The C4.5 [12] algorithm can be utilized to generate classification trees based on the normalized information gain of each node. The value of normalized information gain utilizes the entropy of the set of samples, calculated as:

$$S(D) = -\sum_{i \in M} p_i \log_2(p_i) \tag{1}$$

where $S(D)$ is the entropy of the set D and p_i is the probability of a sample to belong to class i in the set of possible classes M. Along with the entropy of the whole set D, given the set V of possible values j of the attribute A, the entropy $S_A(D)$ of the set D after splitting it into subsets based on the value of the attribute A is calculated as a weighted average of the entropy $S(D_j)$ of each subset D_j, using the proportion of samples in the subset as weight. Given the entropy of the whole set ($S(D)$) and the subsets ($S_A(D)$), the normalization factor is calculated as:

$$SplitS_A(D) = -\sum_{j \in V} \frac{|D_j|}{|D|} \times \log_2(\frac{|D_j|}{|D|}) \tag{2}$$

where $|D_j|$ is the number of samples in the subset D_j and $|D|$ is the number of samples in the set D. Given this last partial value, the normalized information gain is calculated by dividing the information gain, defined as the difference of entropy between set ($S(D)$) and subset ($S_A(D)$), by the normalization factor (as shown in (2)). This value is calculated for each attribute A, with the algorithm selecting the attribute with the highest normalized information gain.

Random Forest is an ensemble system that combines several classification trees at the learner level [5]. For the other levels, a number of techniques can be used, resulting in different forests. In this work, we utilize simple *majority voting* at the combination level, $\sqrt{n}$ features examined per node (split) out of the n features available at the feature level, and *bootstrapping* at the data level.

3 THE BACKORDERS SYSTEM FOR DDOS DETECTION IN THE DATA PLANE

In this section, we detail our proposed system, BACKORDERS.

3.1 Approach Overview

BACKORDERS is a system for classifying network traffic within the data plane, with the main purpose of detecting DDoS attacks. To achieve this goal, we utilize a Random Forest model to aid in the process of classifying flows.

The training of a Random Forest (RF) is costly, both in terms of CPU and memory utilization. Thus, we elect to perform this step in the control plane, as it has more available resources. Despite the training of an RF being computationally heavy, the evaluation of a sample is relatively simple. The process of evaluating a sample consists of several comparisons being made, comparing pre-calculated values, called *thresholds*, against values calculated for each sample, called *features*. While a Classification Tree can have a large number of nodes, the number of comparisons done is logarithmic. As such, we propose the use of an RF model in P4-enabled switches to perform the classification of flows at line-rate.

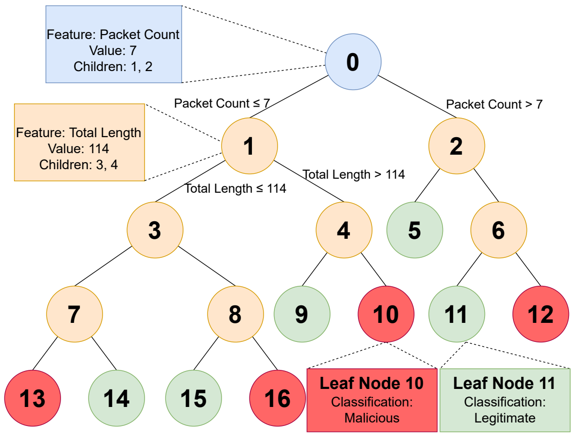


Figure 1: Example of nodes of a Classification Tree.

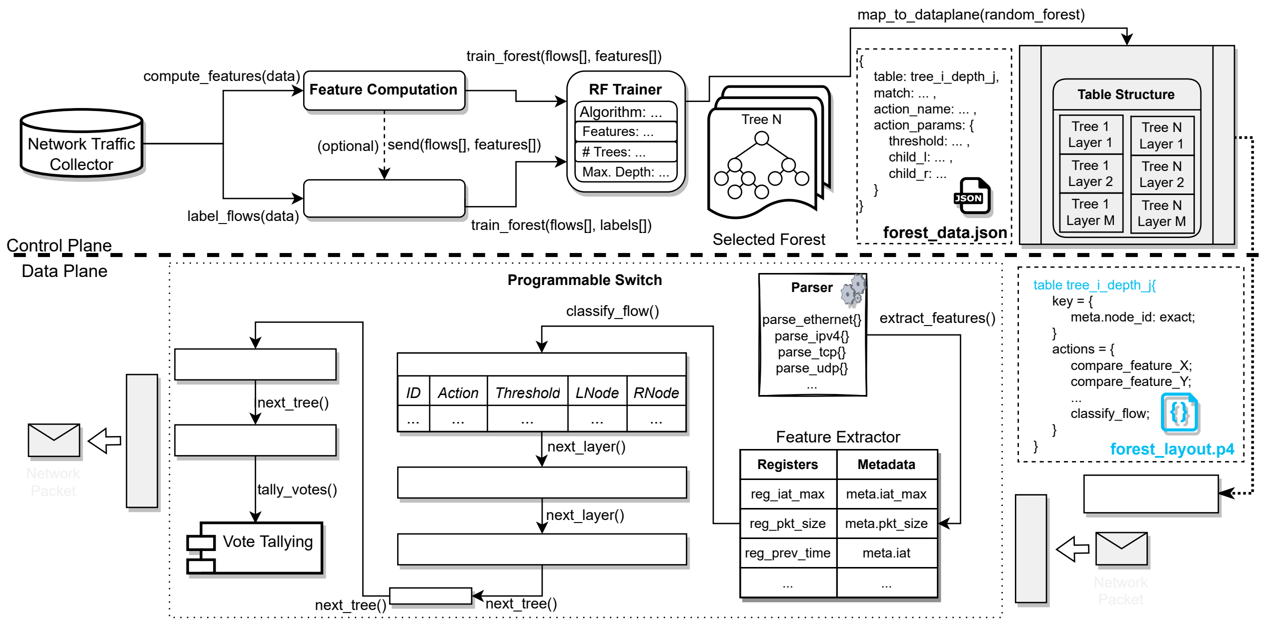
3.2 Random Forest Classification in the Data Plane

Given a Random Forest model, the first step towards performing online classification in the data plane is to map the forest's structure and operations to the constructs available in the P4 language, while also considering the limitations imposed by the hardware of programmable switches. Specifically, programmable switches do not support floating-point numbers, complex arithmetic operations (e.g., divisions), or loops (e.g., for, while, recursion). Finally, programmable switches have a limited amount of memory, around a few tens of megabytes of available SRAM [18].

A Classification Tree (CT) is a collection of nodes. Each of these nodes can be either an internal node or a leaf node. To evaluate an **internal node**, it is necessary to first perform a comparison against a threshold value or a series of class values. The algorithm branches to a different node depending on the result of the comparison. Evaluating a **leaf node** is a bit simpler, as all the information they hold is a *classification* for samples whose path leads to that *leaf*.

Classification Trees are commonly implemented utilizing recursion. However, P4 does not allow recursion, nor does it allow structures to reference other instances of that same structure. That is, a CT node cannot directly reference another CT node. Thus, we propose a way to map the information that each node must hold into entries of a match+action table.

Given an internal node, we map its relevant *feature*, *threshold*, and *children* to a match+action table. In our approach, each node has a unique identifier, such as '0' and '1'. Figure 1 shows an example of a classification tree. A representation of the match+action table entry that *node* '0' would be mapped to can be seen in Table 1. As such, this mapping would match its *identifier* '0', invoking the action *compare_pkt_count* (*i.e.,* compare packet count of that flow), passing as parameters the *threshold* 7 and its children, '1' and '2'. Once invoked, the action would compare the appropriate feature



```
{
  table: tree_i_depth_j,
  match: ... ,
  action_name: ... ,
  action_params: {
    threshold: ... ,
    child_l: ... ,
    child_r: ...
  }
}
```

```
table tree_i_depth_j{
    key = {
        meta.node_id: exact;
    }
    actions = {
        compare_feature_X;
        compare_feature_Y;
        ...
        classify_flow;
    }
}
```

Figure 2: The Architecture of BACKORDERS: the main modules run in the control plane (*RF Mapper*) and in the data plane/programmable switch (*feature extractor* and *online classifier*).

Table 1: Example of the mapping of internal nodes to a match+action table

Match Value	Action	Parameters		
Node ID		Threshold	Child 1	Child 2
0	compare_pkt_count	7	1	2
1	compare_total_length	114	3	4
2	compare_feature_B	y	5	6
8	compare_feature_H	z	15	16

to the threshold passed as parameter, indicating that the next node to be evaluated is '1' if *pkt_count* has a value less than or equal to 7, or node '2' otherwise.

3.3 BACKORDERS Architecture

The architecture of BACKORDERS comprises several components, which are shown in Fig. 2. (1) *RF Mapper* is a module that runs in the control plane and is responsible for converting the RF into BACKORDERS's internal representation. In order to perform the classification of network traffic at line-rate, BACKORDERS implements two additional modules in the data plane, (2) a *feature extractor* module, and (3) an *online classifier* module, responsible for orchestrating the multiple Classification Trees implemented in the data plane. These components will be discussed in detail next[1].

3.3.1 RF Mapper. This module is responsible for taking an input describing the structure of a Classification Tree (CT) and mapping it to a series of match+action tables to be inserted into the switch. As Random Forests contain multiple CTs, we apply this process over each tree in the forest. Each node in a tree, either internal or leaf, has a unique *identifier*, which is used to perform the matching in match-action tables. **Internal nodes** also specify the *feature* to be used, *threshold*, and *children*. The **feature** field is mapped into a specific P4 action. Ergo, every node that utilizes a specific feature will invoke the same action. Table 1 shows an example of this mapping, where a node that utilizes the number of packets of that flow will invoke the corresponding *compare_pkt_count* action, comparing the *pkt_count* feature of that flow to the *threshold* parameter in the match+action table entry. This will be true for every node with *pkt_count* as its feature. The **threshold**[2] of a node is passed as a parameter to the action that will perform the comparison. This is configured by the controller, responsible for inserting the corresponding match+action table entry in the switch. As a comparison against a threshold can only produce two results (i.e., greater than the threshold or not), each internal node has exactly two children. A node's **children** is mapped by passing their respective identifiers as parameters to the same action.

Leaf nodes only contain an identifier and a *classification*, having a different format for match+action table entries. Table 2 shows an example of the mapping of leaf nodes. Every leaf node utilizes the same action, *classify_flow*, passing as parameter the *classification* predicted by this leaf node.

3.3.2 Feature Extractor. This module is located in the programmable data plane of a P4-enabled switch and performs feature computation for each incoming packet. It uses packet metadata and header values to calculate flow features to be used by the RF.

[1]BACKORDERS also relies on a set of off-the-shelf *external components*, such as network traffic collector and random forest training algorithms. Since these are not part of the core of our solution, they will not be detailed here.

[2]Our system only implements numeric features at this time. We believe this is appropriate for network traffic classification, as network protocol headers are generally interpreted as either a number or a boolean flag (0 or 1).

Table 2: Example of the mapping of leaf nodes to a match+action table

Match Value Node Identifier	Action	Parameters Classification
5	classify_flow	LEGITIMATE
9	classify_flow	LEGITIMATE
10	classify_flow	MALICIOUS
11	classify_flow	LEGITIMATE
12	classify_flow	MALICIOUS
13	classify_flow	MALICIOUS

Features. Some features are based on values from the IPv4 header, such as *PSH* and *URG* flags. Other features utilize values from the TCP or UDP headers, such as *destination port*. Features also include *flow duration*, *packet count*, and *header length sum*, the sum of the length of headers of each packet of that flow. The *initial window* is extracted from the window size field of the TCP header of the first packet of a flow if the header is present. *ACT Data Count* is the number of packets of that flow that had at least one byte of payload.

Finally, we also calculate several metrics related to *packet length*, calculated based on the number of bytes of each packet of that flow, *inter-arrival-time* (IAT), the time elapsed between the arrival of the previous packet of that flow and the next, at the current time, and *segment size*, the length of the payload of each packet of that flow. For these last three types of information, we calculate total, minimum, maximum and mean values, resulting in 12 features. In total, our system currently supports twenty features (Table 3).

Table 3: Features implemented

Destination Port	
Flow Duration	
Packet Count	
Header Length Sum	
Initial Window	
ACT Data Count	
PSH	Flag Count
URG	
Packet Length	Total
	Minimum
	Maximum
	Mean
Inter-Arrival-Time	Total
	Minimum
	Maximum
	Mean
Segment Size	Total
	Minimum
	Maximum
	Mean

Approximating Means. As programmable switches must ensure line-rate processing, the time budget to process each packet is restricted.

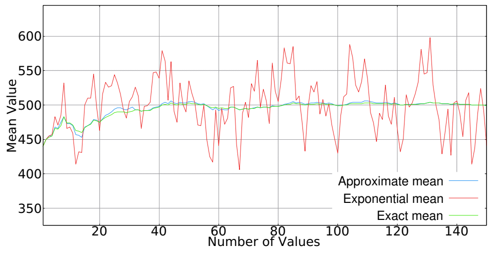


Figure 3: Mean values for different approaches.

Due to the complexity of the division operation, the P4 language does not support divisions. Thus, calculating the average values is not a trivial task, as it requires the computation of the division of a sum by its number of elements. Due to this restriction, existing work [18] typically relies on Exponentially Weighted Moving Averages (EWMA), by setting a weight of $\frac{1}{n}$, with n being a power of two, so that it can be simulated with bit-shifts. In order to minimize the difference between the computed approximation and the real moving average, we implemented a new technique to be employed in lieu of EWMA. Considering the other implemented features, our algorithm requires only a single additional value to be computed and stored. The computation of the moving average differs depending on the number of elements, i - if i is a power of two, we can calculate the exact division by utilizing bit-shifts. However, in every other case, we calculate an auxiliary value, $S_a(i)$, as shown in Equation (3), based on the auxiliary value calculated in the previous iteration and the approximate mean, $(M_a(i-1))$ of the previous iteration, along with the new value V. The approximate mean of this iteration is, then, calculated by dividing this auxiliary value $(S_a(i))$ by the highest power of two that is lesser than i, by utilizing bit-shifts. Finally, we define the approximated mean as $M_a(i) = \frac{S_a(i)}{\max 2^n \leq i,\ n \in \mathbb{N}}$. Thus, for example, to approximate the mean of 9 elements, we calculate $S_a(9)$ and divide it by 8, through a bit-shift of 3 bits. To calculate $S_a(9)$, we must calculate $S_a(8)$, as in the second case of Equation (3), and $M_a(8)$, which can be done by dividing $S_a(8)$ by 8. Fig. 3 compares the values calculated by our implemented approach (approximate mean) and EWMA with $\alpha = 0.5$, where we can see the value calculated by our algorithm being closer to the real value of the moving average.

$$S_a(i) = \begin{cases} \sum_{i=1}^{N} V_i, & \exists n \in \mathbb{N}, 2^n = i \\ S_a(i-1) - M_a(i-1) + V, & \nexists n \in \mathbb{N}, 2^n = i \end{cases} \quad (3)$$

3.3.3 Online Classifier. This module is responsible for orchestrating the multiple Classification Trees that compose our Random Forest in order to perform the classification of a network flow. The *online classifier* is located in the ingress processing block of the P4-enabled switch, where the logic for utilizing multiple classification trees is done. The first step towards classifying a network flow is to calculate the features to be used by the classification trees. After requesting the computation of features, in order to efficiently approximate means, we utilize an auxiliary match+action table with static entries. Along with invoking this table, the *online classifier* is responsible for invoking the tables that compose the multiple random forests. As the *RF mapper* module maps the trained RF into several match+action tables per tree[3], we must invoke multiple match+action tables per classification tree. By separating nodes into different match+action tables based on their depth, we ensure that invoking the same table multiple times will never be necessary, as only one node is evaluated for each level of the tree. This is done because P4 does not allow the control block to invoke the same table multiple times.

4 IMPLEMENTATION AND EVALUATION

This section presents the implementation and evaluation of BACKORDERS. We first describe the prototype that was implemented. Next, we present the methodology employed in the experiments and the results. Finally, we perform a theoretical analysis of the cost of different RF configurations.

4.1 Prototype

We utilized *scikit-learn* [9] to implement the *RF trainer* module. The *RF mapper* generates JSON configuration files for the insertion of trees and parts of P4 code for the definition of the multiple tables that contain the trees. In the data plane, we utilized the P4 language in order to implement the *feature extractor* and *online classifier* modules on the BMv2 software switch. In order to calculate every feature described in Section 3.3, we utilized 23 registers for values, along with 5 registers to store the 5-tuple that defines a flow, Source and Destination IP Addresses, Source and Destination Ports and Protocol. In our prototype, we aimed to provide an implementation for the calculation of every feature, regardless if it was utilized by the inserted forest or not. Potential optimizations include tailoring the set of features calculated for specific scenarios.

4.2 Methodology

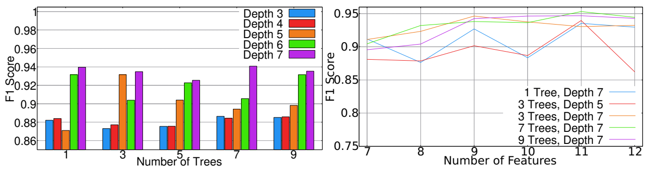
In order to evaluate BACKORDERS, we utilized the CICIDS2017 dataset [14]. This dataset contains a large number of labeled flows with over seventy features each. In particular, we utilized the subset of samples collected on Wednesday, July 5, 2017. This subset contains 692,703 labeled flows, with 440,031 flows labeled as legitimate, corresponding to over 63.52% of the total number of flows. The remaining samples include five different types of slow DDoS attacks. While the dataset contains over 70 features, we considered many of them to be too complex to be implemented in a programmable switch, and thus we selected a subset of features that were simpler to implement in a P4-enabled switch, as described in Section 3.3. As the algorithm for induction of Classification Trees automatically tries to select the best features as early as possible in the process, we simply provided the list of implemented features to each RF, allowing the algorithm to detect and select the most significant features. We compared random forests with 1, 3, 5, 7, and 9 trees, and trees with a maximum depth of 3, 4, 5, 6, and 7 levels. Additionally, we employed K-fold cross-validation with 5 folds per model, averaging the F1-Score[4] obtained with each fold.

[3]Implementing a decision tree within a single match+action table would require packets to be resubmitted/recirculated, thus reducing throughput.

[4]F1-Score is a popular metric for measuring the efficacy of ML learners, which combines precision and recall.

4.3 Evaluation Results

4.3.1 Learner Scores. We compare the F1-Score obtained by each trained model, considering legitimate traffic as the positive class. Fig. 4a shows the scores obtained by each model, where we can see that every model has an F1-Score higher than 0.85, including the forests with fewer trees and lower depth. We can also observe that, by increasing the maximum depth, the trained forests obtain a considerably increased F1-Score, even with a small number of classifiers per forest. Thus, even a forest with a small number of trees and limited maximum depth can obtain reasonable results.



(a) Different RF configurations **(b) Subsets of features**

Figure 4: F1-Scores.

Additionally, we have selected a few top-scoring combinations from the previous experiment and performed further analysis based on those configurations. In a first step, a hierarchical clustering of features was done, in order to identify correlated or multicollinear features. After selecting only one feature per cluster, we iteratively aimed to identify and remove the least contributing feature from the set of available features. In the results shown in Fig. 4b, we notice that after a certain amount of features, adding more features does not significantly increase the F1-Score of the examined models, while sometimes decreasing it. Intuitively, this might be due to the fact that not every feature has the same contribution to the model's predictive ability.

4.3.2 Scalability Analysis. As only one node is processed at each level, the number of comparisons done by a tree is proportional to its maximum depth. Thus, the number of comparisons done by the random forest is limited by $O(NM)$, where N is the number of trees in the forest and M is the maximum depth of each forest. Additionally, as each node is mapped into a single match+action table entry, the number of match+action table entries used by each tree is limited by $O(2^M)$, where M is the maximum depth of the tree. Thus, a random forest may take up to $O(N(2^M))$ match+action table entries, where N is the number of trees in the forest. In Table 4 we calculate the number of comparisons and match+action table entries for a few example configurations. As each entry matches on an exact key (the node's ID), they can be mapped to SRAM. Programmable switches tend to have a few tens of megabytes of SRAM [18]. Therefore, even the largest RF would incur negligible SRAM usage, as it would only take 1143 entries. Finally, as the memory cost per monitored flow is proportional to the number of features tracked for each flow, based on the results shown in Fig. 4b, we can find a compromise between memory usage and accuracy of the model.

Table 4: Cost analysis of different RF configurations

# Trees	Max. Depth	Processing per tree	Total processing	Table entries	Total table entries
1	6	6	6	63	63
	7	7	7	127	127
3	5	5	15	31	93
	6	6	18	63	189
	7	7	21	127	381
5	5	5	25	31	155
	6	6	30	63	315
	7	7	35	127	635
9	6	6	54	63	567
	7	7	63	127	1143

5 RELATED WORK

In this section, we discuss related work that propose techniques for detecting DoS and DDoS attacks in the data plane. Lapolli et al. [6] detect DDoS attacks based on estimates of the entropy of source and destination IP addresses. This approach assumes that there will be a large number of attackers, but the effectiveness for low-bandwidth attacks is unknown. Additionally, their system can detect the occurrence of attacks but not classify traffic into malicious or legitimate.

Simsek et al. [15] identify hosts from where DDoS attacks originate by detecting spoofing. Their work focuses on volumetric attacks, that is, attacks with high-bandwidth volume and a massive number of packets. Thus, slower DDoS attacks may be undetected by such a system, whereas our proposed system can be used for the detection of this type of attack.

Febro et al. [3] utilize deep packet inspection (DPI) to identify DDoS attacks that target the SIP protocol in the application layer. However, DPI is unable to handle encrypted traffic, which has become very popular. Additionally, it is limited to DDoS attacks that target SIP, whereas our proposed system can be used to detect a wider range of attacks.

Musumeci et al. [8] utilize and compare several ML classifiers in order to detect DDoS attacks. In their approach, statistical features can be computed in the data plane, but an ML module is responsible for the classification of traffic. ORACLE [7] utilizes the data plane to extract per-flow features. These are used by an ML model in the control plane in order to classify flows. While these systems can calculate features in the data plane, their ML modules are not implemented in the data plane. This introduces a longer delay before classification, on top of the forwarding device not being able to take any immediate action based on the classification.

pForest [18] implements several RF classifiers in the programmable data plane. This system classifies network flows in order to detect potential DDoS attacks. For mean values, which are more complex to calculate, they replace the moving average with exponentially weighted moving average, with a weight of $\frac{1}{2}$, which simplifies the computation with the limited capabilities of programmable switches. Despite their positive results, our proposed system utilizes a single RF, diminishing resource usage. Additionally, our proposed mechanism better approximates means with an algorithmic approximation of moving averages.

pHeavy [20] offloads decision trees to programmable switches in order to detect heavy flows at line-rate. The decision trees are trained with algorithms that reduce the imbalance in the distribution of classes, i.e., heavy and non-heavy flows. Further, pHeavy utilizes several decision trees in succession in order to accurately identify heavy flows. Similarly to our approach, pHeavy utilizes registers to calculate and store features for each flow. However, differently from our approach, pHeavy relies solely on decision trees. Random Forests are more refined models, as they are an ensemble of decision trees, being less prone to some of the issues that decision trees are known for [5].

Planter [21, 22] is capable of offloading several machine learning models into programmable network devices, including P4-enabled switches. The framework supports a variety of machine learning models, including Random Forests, XGBoost, K-Nearest Neighbors, and Neural Networks. Planter supports alternative ways to encode Random Forests (and decision trees) into programmable data plane devices. The first way to encode is similar to our approach (and the one in pForest [18]), while the second implementation requires a table for each tree and for each feature. The authors provide an evaluation of these alternative implementations, showing the trade-offs between them.

Stat4 [4] is a P4 library capable of computing and tracking statistical values in programmable switches. The statistics computed include moving means, variance, standard deviation, and percentiles of distributions. The authors show how features provided by the library can be used to detect anomalous traffic, as well as identify the recipient of said traffic. This can be used, for instance, to detect a volumetric DDoS attack and identify its victim, but it is not able by itself to differentiate between malicious and legitimate flows. However, Stat4 could be used as a complementary library to our approach, enabling the tracking of more advanced features to be used by Random Forests implemented in the data plane.

6 CONCLUDING REMARKS

In this paper we presented BACKORDERS, a system for classifying network flows in programmable data planes. Our system relies on a Random Forest classifier, mapping its structure to match+action tables to fit in a P4-enabled switch. Our system further calculates features for each flow entirely in the data plane. These features are utilized by the RF in order to provide a classification of network flows, identifying whether they are legitimate or malicious flows.

We believe that this work shows that it is possible to utilize machine learning models in the data plane. By implementing a random forest in the data plane, we can perform highly-accurate predictions at line-rate, meeting the restrictions imposed by the limited hardware of programmable switches. As future work, we plan on evaluating the trade-off between implementing more features and minimizing memory usage.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for the helpful feedback which significantly improved the quality of this paper. This work was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, CNPq (grant #311276/2021-0), FAPERGS (grant #19/2551-0001645-0) and FAPESP (grant #2020/05152-7 - PROFISSA, and grant #15/24494-8 - BigCloud).

REFERENCES

[1] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[2] Bruno L. Dalmazo, Jonatas A. Marques, Lucas R. Costa, Michel S. Bonfim, Ranyelson N. Carvalho, Anderson S. da Silva, Stenio Fernandes, Jacir L. Bordim, Eduardo Alchieri, Alberto Schaeffer-Filho, Luciano Paschoal Gaspary, and Weverton Cordeiro. 2021. A systematic review on distributed denial of service attack defense mechanisms in programmable networks. *International Journal of Network Management* 31, 6 (2021), e2163.

[3] Aldo Febro, Hannan Xiao, and Joseph Spring. 2019. Distributed SIP DDoS Defense with P4. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)* (Marrakesh, Morocco). IEEE Press, Marrakesh, Morocco, 1–8. https://doi.org/10.1109/WCNC.2019.8885926

[4] Sam Gao, Mark Handley, and Stefano Vissicchio. 2021. Stats 101 in P4: Towards In-Switch Anomaly Detection. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks* (Virtual Event, United Kingdom) *(HotNets '21)*. Association for Computing Machinery, New York, NY, USA, 84–90. https://doi.org/10.1145/3484266.3487370

[5] Tin Kam Ho. 1995. Random Decision Forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1 (ICDAR '95)*. IEEE Computer Society, USA, 278.

[6] Ângelo C. Lapolli, J. Marques, and L. Gaspary. 2019. Offloading Real-time DDoS Attack Detection to Programmable Data Planes. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE Press, Arlington, VA, USA, 19–27.

[7] Sebastián Gómez Macías, Luciano Paschoal Gaspary, and Juan Felipe Botero. 2021. ORACLE: An Architecture for Collaboration of Data and Control Planes to Detect DDoS Attacks. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE Press, Bordeaux, France, 962–967.

[8] F. Musumeci, V. Ionata, F. Paolucci, F. Cugini, and M. Tornatore. 2020. Machine-learning-assisted DDoS attack detection with P4 language. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. IEEE Press, Dublin, Ireland, 1–6. https://doi.org/10.1109/ICC40277.2020.9149043

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[10] Amit Praseed and P. Santhi Thilagam. 2021. Modelling Behavioural Dynamics for Asymmetric Application Layer DDoS Detection. *IEEE Transactions on Information Forensics and Security* 16 (2021), 617–626. https://doi.org/10.1109/TIFS.2020.3017928

[11] J. R. Quinlan. 1987. Simplifying Decision Trees. *Int. J. Man-Mach. Stud.* 27, 3 (Sept. 1987), 221–234. https://doi.org/10.1016/S0020-7373(87)80053-6

[12] J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[13] Anderson Santos Da Silva, Cristian Cleder Machado, Rodolfo Vebber Bisol, Lisandro Zambenedetti Granville, and Alberto Schaeffer-Filho. 2015. Identification and Selection of Flow Features for Accurate Traffic Classification in SDN. In *2015 IEEE 14th International Symposium on Network Computing and Applications*. IEEE Press, Cambridge, MA, USA, 134–141. https://doi.org/10.1109/NCA.2015.12

[14] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *ICISSP*. INSTICC, SciTePress, Funchal, Madeira, Portugal, 108–116. https://doi.org/10.5220/0006639801080116

[15] Goksel Simsek, Hakan Bostan, Alper Kaan Sarica, Egemen Sarikaya, Alperen Keles, Pelin Angin, Hande Alemdar, and Ertan Onur. 2020. DroPPPP: A P4 Approach to Mitigating DoS Attacks in SDN. In *Information Security Applications*, Ilsun You (Ed.). Springer International Publishing, Cham, 55–66.

[16] Nikhil Tripathi and Neminath Hubballi. 2021. Application Layer Denial-of-Service Attacks and Defense Mechanisms: A Survey. *ACM Comput. Surv.* 54, 4, Article 86 (may 2021), 33 pages. https://doi.org/10.1145/3448291

[17] Loïc D. Tsobdjou, Samuel Pierre, and Alejandro Quintero. 2022. An Online Entropy-based DDoS Flooding Attack Detection System with Dynamic Threshold. *IEEE Transactions on Network and Service Management* 19, 2 (2022), 1679–1689. https://doi.org/10.1109/TNSM.2022.3142254

[18] Coralie usse Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. 2019. pForest: In-Network Inference with Random Forests. https://doi.org/10.48550/ARXIV.1909.05680

[19] Hua Wu, Tingzheng Chen, Ziling Shao, Guang Cheng, and Xiaoyan Hu. 2021. Accurate and Fast Detection of DDoS Attacks in High-Speed Network with Asymmetric Routing. In *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE Press, Madrid, Spain, 1–6. https://doi.org/10.1109/GLOBECOM46510.2021.9685794

[20] Xiaoquan Zhang, Lin Cui, Fung Po Tso, and Weijia Jia. 2021. pHeavy: Predicting Heavy Flows in the Programmable Data Plane. *IEEE Transactions on Network and Service Management* 18, 4 (2021), 4353–4364. https://doi.org/10.1109/TNSM.2021.3094514

[21] Changgang Zheng, Mingyuan Zang, Xinpeng Hong, Riyad Bensoussane, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. 2022. Automating In-Network Machine Learning. https://doi.org/10.48550/ARXIV.2205.08824

[22] Changgang Zheng and Noa Zilberman. 2021. Planter: Seeding Trees within Switches. In *Proceedings of the SIGCOMM '21 Poster and Demo Sessions* (Virtual Event) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 12–14. https://doi.org/10.1145/3472716.3472846