Attack Detection and Mitigation using Intelligent Data Planes in SDNs

Aparna Ganesan
Department of Computer Science
The University of Texas at Dallas
Richardson, TX 75080
Aparna.Ganesan@utdallas.edu

Kamil Sarac
Department of Computer Science
The University of Texas at Dallas
Richardson, TX 75080
ksarac@utdallas.edu

***Abstract*—Despite its significant advantages over distributed control in traditional networks, the centralized control used in software defined networks (SDN) introduces potential security vulnerabilities. The controller-switch bandwidth, flow tables in data plane switches, and the controller itself could become overwhelmed by potential denial of service attacks in SDN.**

In this work, we present a machine learning (ML) based approach to defend SDNs from such attacks. We use decision tree and logistic regression based ML models to identify decision boundaries at the controller site. We then translate these decision boundaries into range compressed match-action table rules. Next, we dynamically communicate these rules to the data plane switches using P4 language primitives enabling switches to filter out attack traffic without needing to consult with the SDN controller for each new packet. Our solution allows us to dynamically update the match-action rules based on the changing behavior of the attack traffic without causing any downtime for the data plane switches.

Keywords– Machine learning, decision tree, logistic regression, programmable data plane, semi-smart switches

I. Introduction

A Software Defined Network (SDN) consists of a control plane and a data plane. An SDN controller that resides in the control plane acts as the brain of the network. Switches that reside in the data plane mainly focus on basic packet forwarding and are typically configured and controlled by the SDN controller. When a data plane switch receives a packet that it does not know how to handle, the switch sends a *PacketIn* message to the controller asking for instructions on how to handle the packet. With this programmability, network management and event handling become logically centralized at the controller. The controller then provides a platform to implement several complex solutions like traffic measurement, monitoring, intrusion detection, load balancing in the control plane and translate the instructions to the data plane switches in the form of flow rule updates. This centralized control of the network, despite having significant advantages over traditional distributed control, introduces new security vulnerabilities. The controller-switch bandwidth, flow tables of the switches, and the controller itself could become overwhelmed with flood of incoming packets.

Traditional firewalls typically utilize static features of packets, e.g., IP addresses, port numbers, flags, etc., to detect and filter out attack packets. Attackers can often evade such detection mechanisms by spoofing various static features to succeed in their attacks. With the advent of efficient and robust machine learning (ML) algorithms and the availability of rich network datasets, we can use ML models for packet classification. Once we identify the important features, we can train ML models with those features to classify packets as benign or attack packets.

With the recent introduction of switch programmability, in-network computing, performing certain applications like packet classification in existing network devices, has gathered attraction. However,the current state of art approaches for switch programmability does not quite allow us to implement ML based model training for packet classification at the switches. However, once the ML model is learned at the controller site, the corresponding decision boundary, i.e., the imaginary $(N-1)$ dimensional surface that separates the N dimensional feature space into different classes for N chosen features, can be interpreted and the intelligence can be incorporated into the programmable switch with the available tools. The controller can proactively feed the decision boundary if there is any change in the traffic pattern and the switches can filter the packets as per the loaded rules. This prevents the attacker from bringing down the controller as the attack packets will be dropped by the data plane before they reach the controller.

In this paper, we present two ML based packet classification approaches that can be implemented at the data plane to detect attack packets with high accuracy while making sure that the match-action table sizes in the switches do not grow drastically. In addition, our solution does not require downtime for the switches to update rules due to the changing behavior of the attack traffic. The prospect of using ML based packet classification implemented on data plane has been examined in [1]. The authors considered four ML algorithms to implement packet classification on switches' match-action pipeline. However, they reported poor accuracy and significant scalability challenges in their implementation. Similarly, in [2], the authors implemented an intelligent data plane for packet classification using a *decision tree classifier*. The model updates are performed by reinstalling a new P4 code onto the target every time there is a change in the decision boundary.

Our main contribution in this work is translating linear ML

models in the form of match-action table rules that can be updated via control plane during network runtime in a scalable way and without having to restart data plane switches. Here, we demonstrate this using *logistic regression* and *decision tree* classifiers and transfer the decision boundary of the classifiers onto the switch. This is done in the form of feature range values rather than exact feature values as we do not want the size of the tables to overwhelm the resources in the switch. These two classifiers were chosen as a first step because we can interpret the decision boundaries and express it mathematically as a linear equation when compared to complex classifiers like random forests or neural networks. The main idea is to utilize the range tables and the compression of exact value tables to range type tables. In this way, very large tables with several thousand entries generated for a logistic regression model can be substantially compressed to have fewer values such that the resulting table size is manageable for the switches. The packets will be processed in a pipeline, progressing each stage of the pipeline based on a feature value and ultimately getting classified as benign or malicious. Depending on the number of features and the type of features, the number of stages in the pipeline are determined. The accuracy value of the experiments show promising results while giving the opportunity for the control plane to add new rules or retract unused rules dynamically without any network downtime for the switches.

In this work, (1) we perform packet classification based on dynamic packet features that will bear the signature of attack packets; (2) we then use decision tree and logistic regression models to train our packet classifiers in the control plane and transform them into match-action rules into the data plane; (3) we train our models on well-known intrusion detection datasets including CICIDS [3] and CICDDoS [4]; and (4) based on the trained model, we generate the ranges of the features that are deemed malicious and install the match-action rules into the switch tables in the form of ranges dynamically. Our evaluations using the match-action table rules show that the accuracy of classification based on these rules is on par with the accuracy of the ML model based on *sci-kit learn* [5] run offline. This demonstrates that the accuracy of the models are maintained even when they are interpreted in the form of a compressed match-action table.

The rest of the paper is organized as follows. Section II presents the related work. Section III provides a brief discussion of the state of the art in switch programming. Section IV presents the theory behind the proposed solution. Section V discusses the implementation details. Section VI presents our evaluations. Section VII concludes the paper.

II. Related Work

With the advent of programmable switches, several works have been proposed to improve the manageability of SDNs by capitalizing the programming capabilities of data plane devices. In case of network security, several studies have been published for heavy hitter detection using programmable switches [6],[7]. The ML algorithms open the avenue for developing better intrusion detection systems that self learn the changing nature of the attack vector [8], [9].

Combining the capabilities of programmable switches and the efficiency of ML based methods for intrusion detection, the idea of implementing ML algorithms in data plane was explored in [1]. In this work, the authors explore the possibility of implementing different ML algorithms on a prototype of their implementation based on both software and hardware approaches. The authors state that their approach is limited in classification accuracy. In our work, we focus on working with decision trees and logistic regression and strive to achieve high accuracy while performing in-network classification.

In [2], the authors used the decision tree classifier to perform intrusion detection. The decision tree classifier is implemented using a *ML-to-P4 compiler* that takes in the decision tree as input and generates a P4 program. This program is then compiled for a specific target by a *firmware builder*. The compiled code gets loaded on to the physical device by an *agent deployer* every time a new model is generated. The model updates need the entire P4 program to be reinstalled into the physical device. In our work, we use logistic regression and decision tree models and the resulting decision boundaries are installed via match-action rule updates. Once the basic P4 code with the essential tables are installed onto the switch, updates from the controller do not require downtime for the data plane switches.

In [10], the authors propose a new data plane architecture, called *Taurus*, to perform per packet ML. The proposed architecture aims at implementing non linear models like *deep neural networks (DNN)*, in the data plane. They extend the current *Protocol Independent Switch Architecture (PISA)*, by adding a *MapReduce* block, that works with the existing blocks like parsers and implements parallelism which is a major attribute of models like Neural Networks. The *Taurus* framework is implemented in hardware and end-to-end performance analysis is reported. While this work can be considered as an initial step towards using complex non linear models for in-network classification, we have aimed to achieve high prediction accuracy with linear models without making any adjustments to the existing architecture or the hardware ASICs.

We see from the current state of art that most of the methods described here either require complete reloading of software switch code in order to update the classification model or need a complex extension to the existing switch architecture. We aim to work on translating ML models in the form of match action tables so that it is scalable (not overloading the switch memory) and updating the model does not cause any downtime or any additional reprogramming delays in the switch operation.

III. P4 and Programmability

P4 [11] programming language was developed to make data plane programming flexible in terms of protocols it can handle and the hardware targets it can be deployed on. P4 attempts to handle the problem of limited programmability in the switches due to restrictions the OpenFlow protocol lays on packet processing and the headers on which it operates. Adding new

protocols and packet processing stages in the form of match-action tables in P4 gives the network operators the freedom to define the precise way in which they want the packets to be handled in the network independent of the underlying target. The components that are involved in deploying a reconfigurable data plane are (1) the P4 program that defines how the packet should be processed, (2) the switch model architecture and the software switch target used, and (3) the *p4runtime API* specification for the communication between the control plane and the data plane. In a recent survey on data plane programmability with P4 [12], authors elaborate on the language fundamentals along with the research advances using the programmable switches for network programming.

A. P4 Programming Language

The P4 programming language [11] gives the network programmers the flexibility to define algorithms using the constructs available. In addition to the usual packet headers for different protocols, programmers can generate intermediate data during the execution of the program, called the *metadata*. P4 offers support to define user defined metadata that helps programmers to store variables that are not in the packet header but are required during packet processing. Features like counters, registers, meters of the language help with the implementation of custom protocols based on the network requirement and to keep track of the network state. In this work, we capitalize the match-action tables offered by P4 to add the intelligence learnt from the trained models.

B. P4 Switch

In this work, we use the P4 software switch *Behavioral Model version2 (bmv2)* [13] that supports *v1model* architecture. The architecture defines the rules that tell the programmers the blocks that can be programmed. In the *v1model* architecture, the programmable blocks are parser, deparser, ingress and egress processing pipelines that have the match-action tables and a non-programmable traffic manager block. The *bmv2* is a reference P4 software switch that has several variations of switch implementations supporting different architectures. The *simple_switch_grpc* is a target based on *v1model*. Based on the definitions from the target architectures, P4 programs are written and compiled using the supported compilers. The programs written in P4 are compiled into JSON representations using the *p4compiler* and are loaded into the software switch during runtime. In this work, we use *simple_switch_grpc* [14] to emulate the data plane switches, with support to communicate with the p4runtime API.

C. P4Runtime

The P4 switches in the data plane are controlled and managed by a P4 capable SDN controller that uses *p4runtime* to communicate with the P4 switches. The *p4runtime* API [15] is the control plane specification that is used for managing a switch that is defined by a P4 program. In this work, we use the *p4runtime_lib* python library to write a custom python program that defines how we want to manage the data plane and write match-action rules emulating a controller that supports P4. The controller program, written in python here, resembles a full fledged SDN controller that is capable of communicating to the switches that support P4. The p4runtime control and data plane communication is facilitated by gRPC[16] framework with protobuf [17].

IV. Proposed Solution

Attacks like DDoS can cause significant damage to SDNs. Empowering the data plane switches with additional knowledge to successfully filter out malicious packets without having to consult with the controller for each such packet can be an effective solution to defend SDNs against these attacks. There have been several attempts in previous works to implement such solutions on the switches using static packet features. But, attackers with large compute resources can launch attacks in such a way that the required storage space for the solutions implemented with static features of the packets grows large overwhelming the switch resources.

In this work, we propose a method to use dynamic features like packet size *(src_bytes)*, number of packets in a flow *(pkts)*, etc., along with the static features like source and destination ports *(src_prt, dest_prt)*, etc., and translate ML models into the data plane in the form of match-action tables such that the number of rules required to represent the decision boundary is tractable. This makes the defense more effective and practical. In addition, we use P4 switches to implement the packet filtering rules in the data plane directly so that we can protect SDN controllers from flooding of *PacketIn* messages.

We make use of P4 switches due to the flexibility to design custom packet processing pipelines. The packet is passed down each block of the pipeline and at the end of the pipeline the packet is either dropped (if it is classified malicious) or passed on to the destination (if it is classified benign) depending on the packet features. With the available features in the P4 language, we focus on using ML algorithms that have linear decision boundaries only. We look for models that are easily represented in the form of look up tables. This restriction is posed because of the P4 language. The current version of P4 language does not support floating point operations making it difficult to implement ML classifiers that often require complex mathematical operations like logarithms, polynomials, etc. This shortcoming of P4 has been recorded in the literature [1], [2] resulting in a need to handle the knowledge transformation to switches in a smarter way.

Next, we want the implementation to be able to update the model without having to disrupt the operation of the network. The presented solution is designed in such a way that the intelligence from the ML classifiers are not hard coded in the data plane but can be updated from the control plane at run time. This is useful when we want to update the decision boundary when the attack signature changes.

In this work, decision tree (IV-A) and logistic regression models (IV-B) are used to classify benign and malicious packets. The workflow is depicted in Fig. 1. All the calculations are performed in the *machine learning and range compression*

block and the ML model is transferred to the *data plane* in the form of compressed match-action rules through the *control plane*. We train the ML classifier with the CICIDS2017 and CICDDoS2019 datasets and evaluate the performance of the models implemented as match-action tables in the data plane.

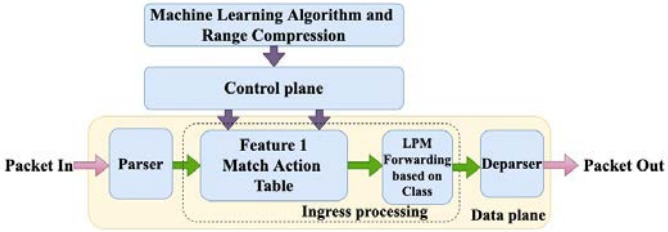

Fig. 1: Proposed solution

A. Decision Tree

Decision trees are supervised learning algorithms that classify the datapoints based on the threshold values of the features generated during training. The tree is constructed from the root and the feature based on which each node splits down is chosen based on a heuristic like the information gain. The next level of the tree is reached from the current node based on its threshold value, for example, all the datapoints that fall below the threshold value will belong to the left child whereas the datapoints above the threshold will propagate down the right child. The threshold value of the feature for each node is learnt from the training dataset such that it offers the greatest information gain for that split.

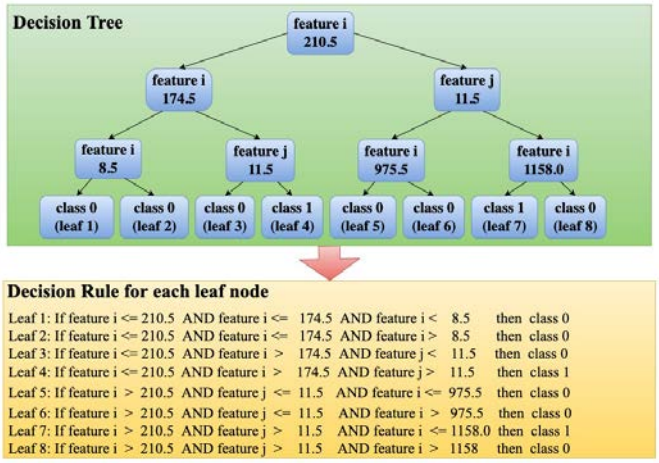

Fig. 2: Decision Tree to match-action rules

S.No	feature i	feature j	Target
1	$[0, 8]$	$[0, j_{max}]$	class 0
2	$[9, 174]$	$[0, j_{max}]$	class 1
3	$[175, 210]$	$[0, 11]$	class 0
4	$[175, 210]$	$[12, j_{max}]$	class 1
5	$[211, 975]$	$[0, 11]$	class 1
6	$[976, i_{max}]$	$[0, 11]$	class 0
7	$[211, 1158]$	$[12, j_{max}]$	class 0
8	$[1158, i_{max}]$	$[12, j_{max}]$	class 1

TABLE I: Rules to *Range* based match-action table

We use the decision tree classifier on the dataset and get the node values for the chosen features. In this work, we restrict the number of features to two and height of the generated tree to three with the leaf nodes being the classes to which a packet belongs. For a decision tree of height three, there can be up to eight leaf nodes. Each leaf node corresponds to a unique decision rule according to which a test point can be classified. We traverse the tree to each leaf node to get the corresponding decision rule. From each rule, we obtain the range values of considered features and the corresponding target value. This means that the prediction accuracy is achieved with at most eight table entries for a classifier.

We use pairs of features that provide good prediction accuracy for the considered attack and use it in the form of match-action rules to provide the intelligence for the switches. In case of decision trees, first we gather the decision rule corresponding to each leaf node as shown in Fig. 2. The tree shown in this figure is a sample tree with values for two features as *feature i* and *feature j*. We assume that, for each of the features, the total possible range is $[0, i_{max}]$ and $[0, j_{max}]$ respectively. From the decision rule, ranges of the feature corresponding to each leaf node is obtained as shown in Table I. The resulting match-action table is loaded on to the data plane switches to perform in-network classification.

B. Logistic Regression

Logistic regression computes the probability of the outcome to belong to each class for a given data point. It is a discriminative classifier model where the decision boundary is generated from the observed data. Model parameters are learnt by maximum likelihood estimation. The model learns weights and bias based on the training data and the probability that a given test point belongs to default class is estimated using the formula

$$y = \frac{1}{1 + \exp\left(-w_0 - w_1 x_1 - w_2 x_2\right)}$$

Here, y denotes probability that the test point belongs to the default class. In this example, the considered dataset has two features x_1 and x_2. The learnt model parameters are w_0, w_1 and w_2 where w_1 and w_2 are the weights and w_0 is the bias. In case of binary classification, the probability that the test point belongs to the other class is $(1 - y)$. A test data point is classified belonging to a class which has the higher prediction probability. In this work, since the P4 language does not support exponential operations, we perform the training and learning in the machine learning and data compression block in the control plane as shown in Fig. 1. For all possible values in the feature range, $[0, i_{max}]$ and $[0, j_{max}]$, we compute the prediction probability and the corresponding class value, generating a table with $i_{max} * j_{max}$ entries. From this table, we capture continuous feature range values with the same class value and aggregate them together as a single entry for the match-action table. This optimization is done to limit the number of entries in the match-action table and to manage the memory resource of the switches efficiently. Otherwise, if the rules are added with individual feature values, the size of the table will be large and intractable. Therefore, without loss of information, we reduce the size of the flow rule table substantially by installing them as *range* match type rules.

For example, Table II, shows how the compressed range table generated for any attack dataset will look like.

S.No	feature_1_range	feature_2_range	Target
1	$[i_1, i_2]$	$[j_1, j_2]$	class
2	$[i_3, i_4]$	$[j_3, j_4]$	class
3	...	...	class
n	$[i_{2n-1}, i_{2n}]$	$[j_{2n-1}, j_{2n}]$	class

TABLE II: Sample *range* based match-action table

V. Implementation

The proposed solution is implemented on P4 switches, running on *mininet* [18] environment. The control plane is implemented using the p4runtime specification. The *simple_switch_grpc* target, based on v1model architecture is used to emulate the data plane switches.

A. Dataset

The datasets used in this work are CICIDS2017 and CICDDoS2019. The CICDDoS2019 dataset includes several types of DDoS attacks including, *UDP, UDPLag, Syn Flood, PortMap, NetBIOS, MSSQL*. The collected dataset has flow features as well as packet level features. The DDoS attack files have proportionally much fewer benign datapoints. Therefore, we combine the benign datapoints from CICIDS2017 dataset to train the model. The ratio of benign to malicious points is kept approximately at $60:40$ uniformly for all categories of attacks. The ML classifiers are trained and the models are installed in the switches in the form of flow rules.

For this analysis, three pairs of features are considered. The selection of the features are based on the availability of support in P4 based switches to extract them and the quality of models that are generated using them. We ran several experiments with feature pairs and chose the ones that performed well across all the considered attacks. The parsers in the P4 architecture help us obtain the length of the incoming packets and counters help us keep the state for the number of packets between a pair of source and destination. Similarly, with the help of parsers, it is straightforward to extract the static features of the packet. The pairs of features used in the work include (1) *src_bytes, pkts*, (2) *src_bytes, src_prt*, and (3) *src_bytes, proto*. Here, *src_bytes* that denotes the packet size in the forward direction is a packet based feature whereas *pkts* denotes the number of packets in the flow and is a flow based feature; *src_prt* denotes the value of the source port value of the incoming packet; and *proto* denotes the protocol number of the incoming packet. For the chosen dataset, these feature sets are considered as the models trained with them resulted in high classification accuracy.

B. Match-action Table

The main goal of this work is to update the knowledge from ML models dynamically in such a way that changes to the model can be performed via the match-action table update without disrupting the existing system performance. The match-action rule updates are sent from the control plane as denoted by the code block in Fig. 3. The *feature_i, feature_j* values are given in the form of ranges to generate the table entry and it is written to the switch via the *INSERT* update in the *p4runtime*. The very first model that is used by the switch will have updates with the least priority value. Newer models are updated with higher priority values. This will allow us to have the newer rules to be elected over the older ones until the older rules are deleted.

```
def writeRules(p4info_helper, ingress_sw, feature_i,
                    feature_j, class_value, priority_value):
    table_entry = p4info_helper.buildTableEntry(
    table_name = "MyIngress.table_name",
    priority = priority_value,
    match_fields = {
    "meta.feature_1": feature_i,
    "meta.feature_2": feature_j,
    },
    action_name="MyIngress.set_class",
    action_params={
    "flag": class_value,
    })
    ingress_sw.WriteTableEntry(table_entry)
```

Fig. 3: Code block that writes the rules to the switch

```
action set_class(bit<14> flag)
{
    meta.class: flag;
}
table table_name{
    key = {
        meta.feature_i: range;
        meta.feature_j: range;
    }
    actions = {
        set_class;
        NoAction;
    }
    size = 4096;
    default_action = NoAction();
}
```

Fig. 4: P4 code to declare range tables

The corresponding tables and actions that need to be defined in P4 is given by the code block in Fig. 4. Here, the keys in the match-action tables are declared as type *range*. The *set_class* action sets the class value to be 0 or 1 (benign or malicious) for a particular entry. In the main processing block, the feature values are updated in the corresponding metadata variables. After the values are updated, the match-action rules are applied on the packet features and the class of the packet is set. Based on the set class value, the packets are either dropped or passed to the appropriate interfaces.

The major goal of this work is to renew the data plane knowledge with newer models that use most recent data to learn. We do this by means of match-action rule updates. There are two areas that need to be considered before performing model updates via table rules: (1) the table size grows drastically if we keep adding rules without purging them; and (2) if the packet feature matches more than one table entry with the same priority, this would be undesirable. We mitigate these issues by purging the outdated table entries by using the *DELETE* update from the *p4runtime*.

VI. Evaluation Results

The decision tree model was implemented using the *DecisionTreeClassifier* from the *sci-kit learn* library. The resulting decision tree is converted into corresponding match-action rules and is used to classify the packets in the data plane switch and report the results for considered six attacks in Table III.

Similarly, we run the *LogisticRegression* model based on *sci-kit learn* and obtain the weights and bias for the considered pairs of features of the dataset. With the obtained weights and bias, we generate a table with the predicted class values for all possible feature values and obtain the continuous ranges with same class value. With the range compressed table, we transport the rules in to the data plane switch to perform in-network classification and report the results in Table IV.

Attack	Results based on sci-kit learn with two features			Results based on range tables with two features		
	src_bytes, pkts	src_bytes, src_prt	src_bytes, proto	src_bytes, pkts	src_bytes, src_prt	src_bytes, proto
UDPLag	0.978	0.975	0.981	0.976	0.977	0.981
UDP	0.996	0.995	0.999	0.996	0.995	0.999
Syn	0.938	0.967	0.938	0.938	0.967	0.938
Portmap	0.996	0.996	0.996	0.996	0.996	0.996
NetBIOS	0.998	0.999	0.998	0.998	0.999	0.998
MSSQL	0.999	0.996	0.999	0.999	0.792	0.999

TABLE III: Accuracy values for the Decision Tree

Attack	Results based on sci-kit learn with two features			Results based on range tables with two features		
	src_bytes pkts	src_bytes, src_prt	src_bytes, proto	src_bytes, pkts	src_bytes, src_prt	src_bytes, proto
UDPLag	0.966	0.841	0.972	0.972	0.836	0.974
UDP	0.993	0.988	0.998	0.992	0.988	0.998
Syn	0.866	0.818	0.847	0.866	0.817	0.846
Portmap	0.995	0.993	0.994	0.819	0.992	0.994
NetBIOS	0.997	0.997	0.998	0.995	0.997	0.998
MSSQL	0.998	0.992	0.999	0.997	0.992	0.999

TABLE IV: Accuracy values for the Logistic Regression

In Table III and IV, we observe the accuracy of the classification from the sci-kit learn based mathematical model as well as the accuracy from classifying the packets using the compressed range tables for both decision tree and logistic regression models respectively (e.g., compare columns 2, 3, 4 with column 5, 6, 7, respectively, in Table III and IV).

We see that the intelligence is ported to the data plane without any loss of information in both decision tree and logistic regression in almost all of the attacks considered, despite performing compression of the individual entries to ranges. This demonstrates that for the attack types considered, there is no loss of information between the actual mathematical model and the proposed solution that uses range tables to accommodate the data plane limitations. We see that our models with fewer features is able to classify packets based on range based match-action table and the classification accuracy is on par with the offline run. We evaluated the performance of the translated models with other metrics like precision, recall, F1 score and saw that the values follow the sci-kit learn based models similar to accuracy. The values are not reported due the page limit. In future work, we will further explore the reasons for the differences in the accuracy as seen in certain attacks and translate other ML models on to the data plane devices.

VII. Conclusions

In this work, we proposed a scalable approach to prevent the controller from becoming a bottleneck during flooding based intrusion attacks. We utilized an ML based approach where we translated the decision boundary of linear ML classifiers to the data plane devices in the form of range based match-action table rules to perform in-network classification of packets. These rules are added dynamically such that there is no network downtime. Our evaluations show that switches with the match-action rules from ML models can perform packet classification with very high accuracy.

This work is partially supported by NSF (DGE-1820640).

References

[1] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, New York, NY, USA, 2019.

[2] B. M. Xavier, R. S. Guimarães, G. Comarela, and M. Martinello, "Programmable switches for in-networking classification," in *INFOCOM*, 2021.

[3] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." in *4th ICISSP*, Portugal, 2018.

[4] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *ICCST*, Chennai, India, 2019.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] R. Ben Basat, X. Chen, G. Einziger, and O. Rottenstreich, "Designing heavy-hitter detection algorithms for programmable switches," *IEEE/ACM Transactions on Networking*, vol. 28, pp. 1172–1185, 2020.

[7] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*, Santa Clara, CA, 2017.

[8] D. Jankowski and M. Amanowicz, "On efficiency of selected machine learning algorithms for intrusion detection in software defined networks," *IJET*, vol. 62, pp. 247–252, 2016.

[9] H. Polat, O. Polat, and A. Cetin, "Detecting ddos attacks in software-defined networks through feature selection methods and machine learning models," *Sustainability*, vol. 12, 2020.

[10] T. Swamy, A. Rucker, M. Shahbaz, I. Gaur, and K. Olukotun, "Taurus: a data plane architecture for per-packet ml," in *Proceedings of the 27th ACM ASPLOS*, 2022.

[11] P. Bosshart, D. Daly, G. Gibb, M. Izzard *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM CCR*, vol. 44, p. 87–95, 2014.

[12] F. Hauser, M. Häberle, D. Merling, and other, "A survey on data plane programming with p4: Fundamentals, advances, and applied research," *arXiv preprint arXiv:2101.10632*, 2021.

[13] "Behavioral model (bmv2)." [Online]. Available: https://github.com/p4lang/behavioral-model

[14] "simple_switch_grpc." [Online]. Available: https://github.com/p4lang/behavioral-model/tree/main/targets/simple_switch_grpc

[15] "P4runtime specification." [Online]. Available: https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html

[16] "grpc," 2021. [Online]. Available: https://grpc.io/

[17] "Google protocol buffers," 2021. [Online]. Available: https://developers.google.com/protocol-buffers/

[18] "Mininet official website." [Online]. Available: http://mininet.org/