Machine Learning Aided Traffic Tolerance to Improve Resilience for Software Defined Networks

Siddharth Gangadhar*, and James P.G. Sterbenz*†‡
siddharth@ittc.ku.edu, jpgs@{ittc.ku.edu|comp.{lancs.ac.uk|polyu.edu.hk}}

*Information and Telecommunication Technology Center
Electrical Engineering and Computer Science
The University of Kansas, Lawrence, KS 66045, USA
www.ittc.ku.edu/resilinets

†School of Comp. and Comm. (SCC) and InfoLab21
Lancaster University, LA1 4WA, UK
‡Computing, The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong

***Abstract*—Software Defined Networks (SDNs) have gained prominence recently due to their flexible management and superior configuration functionality of the underlying network. SDNs, with OpenFlow as their primary implementation, allow for the use of a centralised controller to drive the decision making for all the supported devices in the network and manage traffic through routing table changes for incoming flows. In conventional networks, machine learning has been shown to detect malicious intrusion, and classify attacks such as DoS, user to root, and probe attacks. In this work, we extend the use of machine learning to improve traffic tolerance for SDNs. To achieve this, we extend the functionality of the controller to include a resilience framework, ReSDN, that incorporates machine learning to be able to distinguish DoS attacks, focussing on a neptune attack for our experiments. Our model is trained using the MIT KDD 1999 dataset. The system is developed as a module on top of the POX controller platform and evaluated using the Mininet simulator.**

***Index Terms*—Resilience, survivability, traffic tolerance; Future Internet; SDN, OpenFlow; Machine learning; Network security, DoS attack, SYN flood**

I. Introduction

Software defined networks (SDNs) have risen in prominence owing to their flexible network management and monitoring abilities. This has primarily been driven by the use of a logically central controller that manages a set of "dumb" switches and modifies flow table entries on demand. The controller, using a custom protocol such as OpenFlow [1], interacts with the switches to monitor statistics, and upon the detection of an event of interest, pushes forwarding table changes to the underlying SDN infrastructure through a Secure Sockets Layer (SSL) encrypted channel. The approach of SDNs, aided by the presence of the centralised controller, has enabled applications that aim to improve network actions such as routing, network management, and security.

Resilience of a network is defined as *the ability of the network to maintain operational state in the face of challenges or faults to normal operation* [2]. The heightened dependance on cyber-physical systems, incorporated in the Internet, has contributed to the increase in vulnerabilities resulting for example in denial of service (DoS) attacks resulting in monetary loss. In more extreme cases, crackers have gained access to sensitive government information resulting in privacy breaches and possibly loss of life. A resilient system must be able to address such challenges using mechanisms that minimise the level of damage incurred due to the challenge.

The ability to tolerate traffic anomalies is a major aspect of providing resilience to a network. Traffic tolerance is a subdomain of challenge tolerance [2], depicted in Figure 1. Traffic tolerance is defined as *the ability to tolerate an unpredictable traffic load on the network* and encompasses challenges such as SYN flooding [3] or smurf attacks [4]. Such attacks can potentially operate in low-rate scenarios and focus on exhausting the resources of the victim and drive the service offline. Any resilient network must have the ability to handle such DoS attacks that aim at bringing down infrastructure.

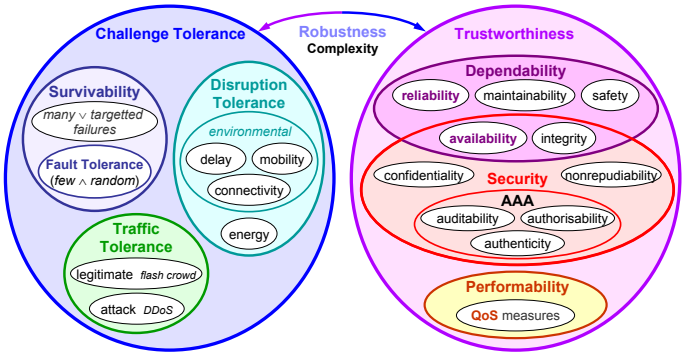

Fig. 1. ResiliNets Disciplines

Machine learning has been extensively used in the area of network anomaly detection, primarily in a classifier role distinguishing legitimate traffic from malicious, for conventional networks [5], [6]. Algorithms such as decision trees (DT) [7], naive Bayes (NB) [8], and support vector machines (SVM) [9] have been employed on traffic datasets, trained using historical information, and deployed for real-time traffic classification. These algorithms, when trained using historical data, have the ability to accurately predict and classify real-time network

978-1-5386-0671-1/17/$31.00 ©2017 IEEE

traffic such as DoS and non-malicious traffic [10], [11].

Our work aims at extending the functionality of the controller in an SDN environment to implement a real-time system that better achieves traffic tolerance using machine learning algorithms. The proposed system would allow the controller to decide what packets are attack-based and drop the respective flow accordingly. The main advantage of the usage of the machine-learning alorithms in an SDN is its ability to have a network-wide drop rule for the attack as opposed to a local rule addition. To report the performance of the real-time system, we compare well known machine learning algorithms, DT, SVM, and NB. The different algorithms are chosen to cover three different types of machine learning algorithms: a probabilistic classifier (NB), a support vector based classifier (SVM) and finally a tree-based classifier (DT). The prediction accuracy is verified both theoretically in addition to the real-time system and reported.

The paper is organised as follows: we summarise relevant literature to our work in Section II. Section III provides a detaied overview of our system, covering system design and a detailed account of its various components. We follow this with an evaluation of our model provided in Section IV, highlighting the accuracy of our individual classification models, followed by the evaluation of our real-time system. Finally, we conclude summarising our findings in Section V.

II. Background and Related Work

Traffic classification using machine learning has been extensively researched for conventional networks. Numerous works have been surveyed [5] covering various learning approaches such as supervised learning, unsupervised learning or clustering, and hybrid. As an example of unsupervised learning, flow clustering using expectation maximisation was used to cluster flows to generate different traffic labels [12]. Attributes such as packet size statistics, byte counts, and interarrival statistics are used as features for this approach. Alternately, in the domain of supervised learning, multiple approaches such as Bayesian neural networks [13] and genetic algorithms [14] based classification techniques have been researched. Bayesian neural networks are seen as an enhancement to traditional neural networks with the integration of the Bayes theorem used to assign weights to the neural network during the training phase [13]. A semi-supervised learning approach was proposed [15] using k-means clustering to aid in forming the unsupervised clusters, and labeled data available in the clusters is used to map the unlabeled samples in the same cluster.

There has been recent work done in the domain of resilience for SDN networks. A resilience framework has been proposed for SDNs to improve network management [16]. This work highlights the use of management patterns, a set of rules or resilience mechanisms that are used in the event of a challenge. A detailed survey of the state of the art of resilience mechanisms for SDNs [17], summarises research in the various domains of resilience, including traffic tolerance. Scalability of the controller and resilience of the underlying SDN network are important, comparing HyperFlow, Onix, and FlowVisor aiming to solve performance and scalability by modifying the controllers. Techniques include path-based protection and controller replication strategies to support resilience [18].

Traffic management through the use of OpenFlow applications have been widely proposed. Traffic can be better engineered through the use of fully-polynomial time-approximation schemes for incrementally deployed SDNs, especially in a multi-controller environment [19]. A load balancer solution enabled by flow admission control that depends on the number of physical requested blocks of data [20] is reported to have a 237% resource gain for per flow allocation. Plug-n-Serve, a load balancer for unstructured SDNs [21], relies on the LOBUS algorithm that aims at minimising the average response time to decide to which particular server the request gets routed. Existing traffic anomaly algorithms have been ported from coventional networks to SDN home networks [22]. A set of 4 anomaly detection algorithms are ported as part of the controller and the performance of each of these investigated in various DDoS attack scenarios.

III. System Design

In this Section, we provide a detailed overview of our model. We outline the design of the system, describe the specifics of the challenge profiler component, provide details on the learning engine, and introduce the resilience engine component.

A. System Architecture

The main goal of our system is to increase resilience of the Internet in the face of challenges to normal operation, focussing on traffic-based anomalies in the network. We aim to accompalish this for SDN networks by extending the controller functionalities, taking advantage of the central nature of the controller. In order to be able to negate DoS attacks, our system must be able to learn the unique characteristics of the attack that help distinguish it from non-malicious traffic.

Thus, our system is implemented as an application module sitting on top of the controller and consists of three main components: the challenge profiler (CP), learning engine (LE), and resilience engine (RE), depicted in Figure 2. The entire module is built as part of the application layer and is implemented as a POX [23] controller module.

B. Challenge Profiler

The main function of the CP is to profile challenges. The CP, via its connection to the underlying SDN infrastructure, gets notified of all new packets arriving in the network via `packet-in` events. Using this information, it then takes appropriate action when a network anomaly occurs through its consultation with the LE (learning engine). The CP consists of three main components: a flow aggregator (FA), feature extractor (FE), and flow inspector (FI).

1) Flow Aggregator: The FA is used to hold flow level information for every packet. A flow is defined as a five tuple ⟨source IP address, destination IP address, source port, destination port, protocol⟩. Every time a packet arrives at the

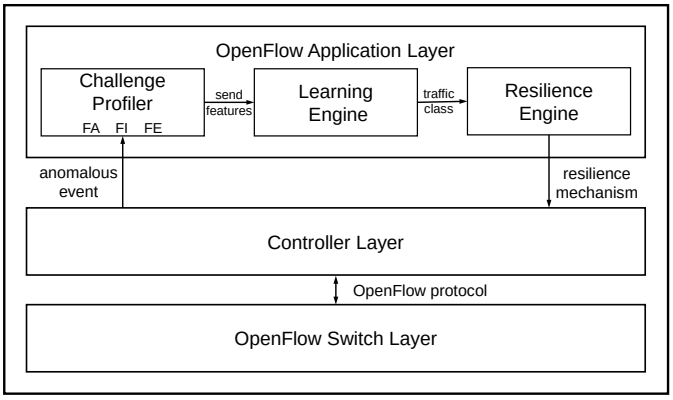

Fig. 2. System architecture

controller, these fields are checked to determine whether the packet is from a new or an existing flow. Once checked, the flow features pertaining to every flow are saved as a dictionary with the key being the flow information (the 5-tuple flow parameters) and the value being the features of the flow.

2) Feature Extractor: The FE extracts features from every packet and provides to the FA. The different features such as checksum (IP checksum is correct or wrong), source bytes (number of data bytes sent by source), and destination bytes (number of data bytes received by destination) are all made available to the FA, which in turn updates its feature values upon the receipt of multiple packets pertaining to the same flow.

3) Flow Inspector: The FI passes the flow information obtained from the FA for classification to the LE. Once the flow has been classified, good traffic is allowed through the network whereas bad traffic is sent to the RE (resilience engine) for appropriate action. Thus the FI forms the interconnection between the LE and the RE.

C. Learning Engine

The main objective of the LE is the classification of network traffic using prediction algorithms. We use a special class of prediction algorithms called supervised learning that uses historical information along with a ground truth of labels to train the classification model. The algorithm that needs to be used undergoes training and testing phases; if performing well, the model is saved for use with the real-time system classification.

Typically, the classification model is trained using cross validation [24], a technique that splits the training data into multiple complementary subsets. Training is carried out on a subset of the training data, the *training set*, while validation is done against a different *validation set*, with accuracy of the selected model as the primary metric. Independent testing can also occur with a dataset that is different to the training dataset. Once an acceptable accuracy is obtained, the model is selected for use in the real-time system. The model is saved for this purpose in order to avoid delays in classification. The psuedocode for this procedure and the real-time system is provided in 1.

Algorithm 1 ML Based Real-time Classification System

procedure MACHINE LEARNING
    for given dataset
Training phase:
    preprocess data
    split training data$(80, 20)$
Testing phase:
    test algorithm$(20\% of data)$
    if *accuracy* $>$ *threshold* **then**
        save model for real-time use
procedure REAL-TIME SYSTEM
for every packet:
    evaluate the pertaining flow
    deduce the features of the packet
for each flow:
    classify flow according to saved model
    if *flow is a DoS attack* **then**
        drop flow
    else
        allow flow

D. Resilience Engine

The RE's main function is to decide and install resilience mechanisms depending on the traffic flow type decided by the LE. Legitimate traffic is allowed to flow through the network, whereas DoS classified traffic is blocked through the push of the `dropFlow` resilience mechanism. The `dropFlow` mechanism utilises the 5-tuple flow identifiers to match the particular flow to be dropped.

IV. EVALUATION

In this section, we present insight into the dataset that was used, theoretical results showing the performance of the various algorithms, and evaluation of our model. Firstly, we provide an overview of the dataset used, then provide information about the metrics used for our evaluation, and finally detail performance of various algorithms using the dataset. We further present the evaluation our real-time system, using Mininet [25] as the simulation platform and POX [23] as the controller.

A. DoS dataset characteristics

The dataset DoS classification is originally part of the MIT Intrusion Detection Evaluation Dataset [4]. This data has been processed and available as part of UCI Machine Learning repository [26]. The dataset contains of a total of 494021 flows as its training data. A total of 41 features exist for this dataset, such as `protocol_type` representing the type of the protocol, `src_bytes` representing the data bytes sent by the particular sender, and `num_failed_logins` that shows the number of failed logins into the system. The features are

classified into three different classes: features of individual TCP connections, content features, and traffic features.

The final column of the dataset consists of 22 attack labels, clustered into three attack types: `u2r` depicting user-to-root, `probe` for probe attacks, `r2l` for remote-to-local attacks, and finally `dos` for denial of service attacks. User-to-root attacks invariably start with the cracker already having access to user permissions on a particular system and attempt to get root access to the same system. Probe attacks aim at scanning a set of computers in a network using probes, discovering the network topology, and looking for vulnerabilities that can be exploited. On the other hand, remote-to-local attackers try to gain access to a system that can be accessed remotely. Finally, a DoS attack is one in which the cracker tries to exhaust the resources owned by a system thus, disabling it from handling any further requests, making it slow and eventually unresponsive [27]. In addition to the attack labels, there is also a distinguisher label for non-malicious traffic called normal. Thus, a total of 23 labels exist for our dataset.

As part of dataset preprocessing, we cluster all labels into three main classes: *normal* representing non malicious traffic, *DoS*, and *other* including all other attack types. In addition to the above post processing, all non-numeric data is converted into numeric values. In order to visualise the dataset, principal component analysis (PCA) [28] is performed allowing for the projection of the high dimensional dataset to lower dimensions, specifically two dimensions for our case. For this purpose, the first two principal components, ordered from high to low variance, are considered and plotted for all three attack labels. Figure 3 shows a random subset of the data samples for each attack label. We observe a clear difference between normal traffic and other attacks; DoS attacks are in the middle.

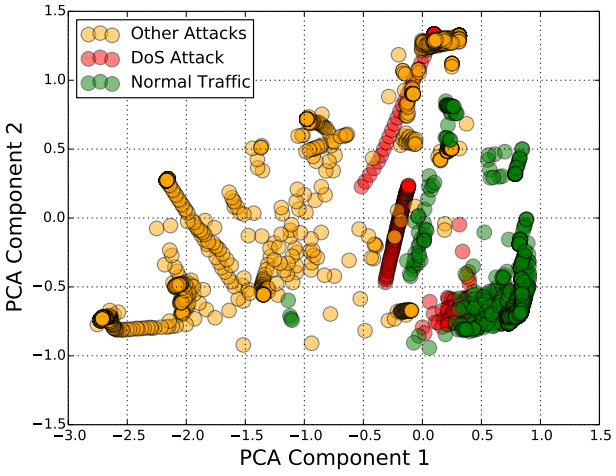

Fig. 3. PCA component visualization of dataset

Additionally, to simplify classification, we further group the other attacks and normal traffic together and label them non-DoS traffic. This allows us to use a binary SVM for classification purposes.

B. Performance Metrics

We consider three performance metrics for our comparison throughout this paper: *accuracy*, *precision* and *recall*. To remain consistent, we use the notation below:

- **TP** – True positives are DoS attacks that have been correctly identified as a DoS attack.
- **FN** – False Negatives are DoS attacks that have falsely classified as legitimate traffic.
- **FP** – False positives are legitimate traffic falsely classified as DoS attacks.
- **TN** – True negatives are legitimate traffic being correctly classified as legitimate traffic.

Accuracy (A) is defined as the ratio of accurate results returned by a classifier:

$$A = \frac{\text{TP+TN}}{\text{TP+FN+FP+TN}}$$

In some cases, the accuracy metric is not precise in reflecting true classifier performance. For instance, when true positives are incorrectly classified as true negatives, the above equation would still hold and hide the misclassification. Furthermore, accuracy can be manipulated and be made to improve by always predicting the existence of one class and ignoring the other. For example, assuming the notation {TP, FN, FP, TN}, a score of {9700,150,50,100} has an accuracy of 0.98 whereas a score of {9850,0,150,0} has a higher value of 0.985. This is called the *accuracy paradox* [29]. In order to better represent the actual performance, we make use of two more metrics: *precision* and *recall*.

Precision (P) is defined as the percentage of retreived results that are relevant. For example, when an intrusion detection system returns 40 that are labeled as DoS attacks but only 20 are actual DoS attacks, the precision is 0.5. In other words, precision provides an insight into how useful the classification results are.

$$P = \frac{\text{TP}}{\text{TP+FP}}$$

Recall (R) is defined as the percentage of relevant instances that have been retreived. Using the simlar intrusion detection system example, if the IDS fails to detect 60 attacks while accurately detecting 20 DoS attacks, the recall value for this scenario is $R = 0.25$. As opposed to precision, recall provides information about how complete the results are.

$$R = \frac{\text{TP}}{\text{TP+FN}}$$

C. Algorithm Comparison

As part of our first experiment, we compare the performance of the various algorithms of interest: SVM, DT, and NB. The comparison is done over the entire dataset, consisting of 494021 rows and 41 features used as training, and a separate

TABLE I
TCP/IP HEADER FEATURES USED

Feature Name	Description
duration	duration of TCP connections
protocol	protocol used for connection
service	service used for connection
src_byteCount	bytes transferred in one connection
dst_byteCount	bytes received in one connection
land	1 if src = dest IP/port, else 0
wrong_checksum	# of wrong checksums for connection
urgent	# packets with Urgent flag set

testing dataset `corrected` consisting of 311029 rows and 41 features used for testing. The dataset is split into 80% for training and 20% for testing. Once the model is trained, the testing dataset is used to test the performance of the system using the metrics accuracy, precision and recall.

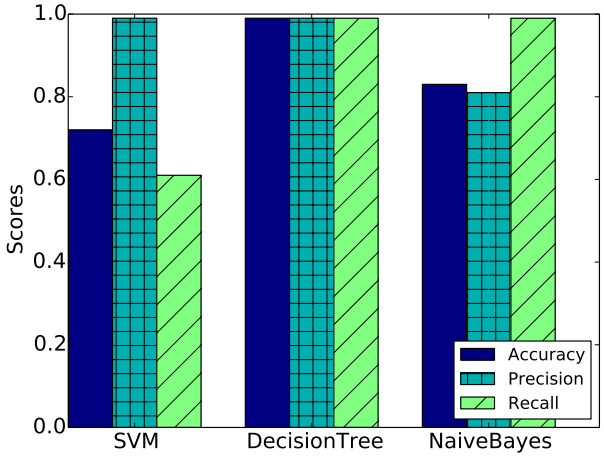

Fig. 4. Performance metrics comparison over 41 features for dataset analysis

The performance of all three algorithms are presented in Figure 4. DT performs best with an accuracy value of 0.996, and precision and recall greater than 0.99 showing the robustness of the accuracy metric. NB follows with an accuracy of 0.83 and precision and recall greater than 0.80. SVM, armed with the radial basis function, performs the worst in this scenario with an accuracy of 0.72 and a high precision of 0.99 and a low recall of 0.61.

1) Effects of Varying Features: As part of our second experiment, we reduce the number of features and check its performance effects for each algorithm. A reduced number of features in turn helps in the implementation of the real-time system by reducing the scope of the implementation process. The reduced feature space is selected if the performance of the particular algorithm is good. To achieve this, we concentrate on the features derived from the TCP/IP header of the packet, summarised in Table I.

From Figure 5, we notice that similar to the previous case: DT scores the best in terms of accuracy at 0.994 with precision

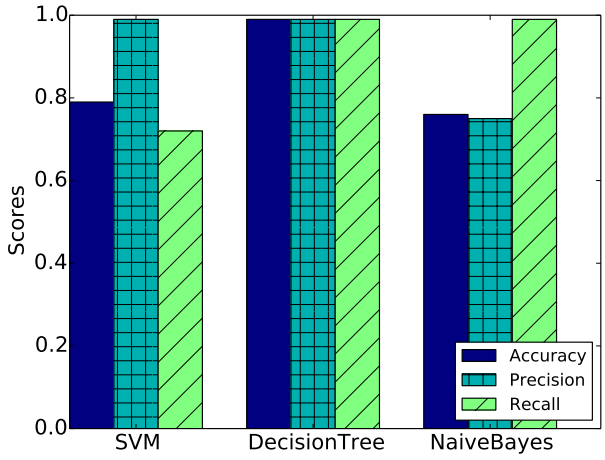

Fig. 5. Performance metrics comparison over 8 features for dataset analysis

TABLE II
ACCURACY VS. SAMPLE SIZE

	10000	50000	100000	200000
SVM	0.52	0.53	0.54	0.55
DT	0.85	0.85	0.86	0.86
NB	0.80	0.80	0.82	0.77

and recall values > 0.99, showing the proficiency of the accuracy metric. SVM scores second for this scenario with an accuracy of 0.79 with precision of 0.996, and recall of 0.72. NB scores worst with an accuracy of 0.76, precision of 0.75, and high recall of 0.996.

2) Effects of Varying Sample Sizes: As part of our third experiment, we investigate the effects of varying the sample size of the algorithm accuracy. Ideally, the accuracy of the algorithm should not change much with any sample size showing the stability of the algorithm. For this experiment, we reduce the training sample size from the initial 494021 samples to vary between the range [10000, 200000]. For each of the sample sizes, we first train using the reduced number of samples and test the trained model using the `corrected` testing samples. The results are tabulated in Table II.

We notice from the table that all three algorithms provide reasonable stability with respect to varying sample sizes. SVM performs the worst throughout in the range of [0.52, 0.55] whereas DT performs best consistently in the range of [0.85, 0.86] over the varying sample sizes. NB performs second best with the highest deviation ranging from [0.77, 0.82].

D. System Evaluation

In addition to the dataset evaluation of different learning algorithms for the purpose of DoS attacks, we have implemented a real-time system that aims to act as an intrusion prevention system and classify DoS attacks. This system is implemented in the controller of an SDN. Similar to the setup in our dataset based results in Section IV-C, we compare all three algorithms:

DT, SVM, and NB. As mentioned in Section III, the system possesses a CP (challenge profiler) that detects challenges, an LE (learning engine) that is responsible for the classification of attacks, and a RE (resilience engine) that houses multiple resilience mechanisms.

We use a SYN-flooding attack to evaluate our system, labeled as "neptune" in the dataset considered [27]. This type of attack affects the TCP/IP stack by taking advantage of the 3-way handshake that occurs during connection initialisation. Multiple TCP connections are initialised through IP-spoofed packets to the victim host and are left half-open after sending the initial SYN packet. The half-open states eventually fill up the TCP data structure that holds connection state and renders the victim unresponsive [27].

For testing, we make use of 46 different services in two stages. The first stage deals with how the system responds to a DoS SYN-flooding attack. For this scenario, we simulate a SYN-flood attack against every one of the services and determine the ones that are accurately classified as a DoS attack (**TP**) and ones that are not classified (**FN**). For the second scenario, we simulate legitimate traffic for all services and determine ones that have been accurately classified as legitimate traffic (**TN**) and ones that are wrongly classified as a DoS attack (**FP**). To simulate legitimate traffic, we make use of the Distributed Internet Traffic Generator (D-ITG) [30] tool. For testing, we assume all services operate on their well-known port numbers.

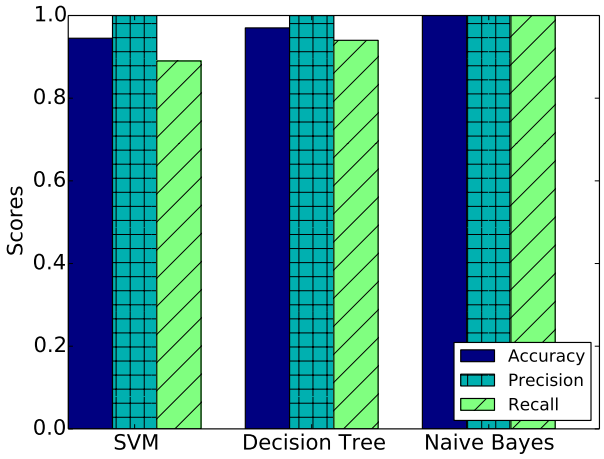

Fig. 6. Performance metrics comparison for real-time system

We first evaluate the system using the DT algorithm. The stage-one testing scenario yields a true positive rating of correctly classifying DoS attack to be 0.94; 3 out of the 46 tested services were misclassified to be legitimate traffic. The misclassified services are observed to be `HTTP`, `IRC` and `X11`. The second scenario testing of legitimate traffic yields a true negative value of 0.96 with 0.04 of legitimate traffic being incorrectly classified to be DoS traffic and thus blocked. Upon further investigation, it is seen that all of this incorrectly classified traffic has a duration feature value greater than zero. Thus, a policy is put in place to check for the duration feature to be zero and allow for misclassified traffic that has a duration greater than zero. This policy is correct as SYN flooding occurs only at time zero. This claim is also supported by the fact that all neptune attacks in the dataset occur with the duration feature being zero. Thus these values are plotted as accuracy, precision and recall in Figure 6. SVM, on the other hand, misclassifies 5 out of the 46 services to achieve a 0.896 true positive rate, while classifying all legitimate traffic services to achieve a 1 true negative rate. The services that are misclassified in the first scenario are `HTTP`, `tftp_u`, `X11`, `ntp_u`, and `IRC`. Finally, NB performs extremely well with classifying all services accurately for DoS attacks to achieve a 1 true positive rate. For the second scenario, initially NB incorrectly blocked all other class of traffic affecting the default signaling port of 9000. Blocking port 9000 meant that no DITG native signaling went through thus bringing all communications to a halt. To negate this, a policy is put in place to allow for other type of traffic. Once allowed, NB performs well, accurately classifying 1 of legitimate traffic.

V. Conclusions

In this paper, we detect and predict DoS attacks, specifically SYN flood, through the use of machine learning. To aid our analysis, we make use of a real-world dataset and use the machine-learning algorithms DT, SVM, and NB to help classify traffic. In addition to a detailed dataset analysis of the performance of the various algorithms, we use the reduced feature space discovered through this analysis to implement a real-time system that is implemented in the controller and classifies packets. The performance of the system is reported with DT consistently performing best throughout our dataset analysis, while NB performs best during our real-time system analysis. As part of future work, we plan to implement various other DoS attacks and evaluate the system under these challenges.

Acknowledgments

We would like to acknowledge the members of the ResiliNets research group for their advice, useful discussions, and suggestions that helped us with this paper. We would also like to thank Murphy McCauley from the POX group for his timely responsiveness and guidance with questions we had regarding the controller, and the Mininet project with help regarding the Mininet simulator.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008.

[2] J. P. G. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer Networks*, vol. 54, no. 8, pp. 1245–1265, 2010.

[3] W. Eddy, "TCP SYN Flooding Attacks and Common Mitigations." RFC 4987 (Informational), Aug. 2007.

[4] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 darpa off-line intrusion detection evaluation," *Computer networks*, vol. 34, no. 4, pp. 579–595, 2000.
[5] T. T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification using Machine Learning," *Communications Surveys & Tutorials, IEEE*, vol. 10, no. 4, pp. 56–76, 2008.
[6] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.
[7] J. R. Quinlan, "Induction of Decision Trees," *Mach. Learn.*, vol. 1, pp. 81–106, Mar. 1986.
[8] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian Network Classifiers," *Mach. Learn.*, vol. 29, pp. 131–163, Nov. 1997.
[9] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
[10] M. Sabhnani and G. Serpen, "Application of machine learning algorithms to kdd intrusion detection dataset within misuse detection context.," in *MLMTA*, pp. 209–215, 2003.
[11] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pp. 1–6, IEEE, 2009.
[12] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow Clustering using Machine Learning Techniques," in *Passive and Active Network Measurement*, pp. 205–214, Springer, 2004.
[13] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian Neural Networks for Internet Traffic Classification," *Neural Networks, IEEE Transactions on*, vol. 18, no. 1, pp. 223–239, 2007.
[14] J. Park, H.-R. Tyan, and C.-C. J. Kuo, "Ga-based Internet Traffic Classification Technique for QoS Provisioning," in *Intelligent Information Hiding and Multimedia Signal Processing, 2006. IIH-MSP'06. International Conference on*, pp. 251–254, IEEE, 2006.
[15] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Semi-supervised Network Traffic Classification," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, pp. 369–370, ACM, 2007.
[16] P. Smith, A. Schaeffer-Filho, D. Hutchison, and A. Mauthe, "Management Patterns: SDN-enabled Network Resilience Management," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–9, IEEE, 2014.
[17] A. S. da Silva, P. Smith, A. Mauthe, and A. Schaeffer-Filho, "Resilience Support in Software-Defined Networking: A Survey," *Computer Networks*, vol. 92, pp. 189–207, 2015.
[18] B. J. van Asten, N. L. van Adrichem, and F. A. Kuipers, "Scalability and Resilience of Software-Defined Networking: An Overview," *arXiv preprint arXiv:1408.6760*, 2014.
[19] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic Engineering in Software Defined Networks," in *INFOCOM, 2013 Proceedings IEEE*, pp. 2211–2219, April 2013.
[20] S. Namal, I. Ahmad, A. Gurtov, and M. Ylianttila, "SDN based Inter-technology Load Balancing Leveraged by Flow Admission Control," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pp. 1–5, IEEE, 2013.
[21] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-Serve: Load-balancing Web Traffic Using OpenFlow," *ACM Sigcomm Demo*, vol. 4, no. 5, p. 6, 2009.
[22] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting Traffic Anomaly Detection using Software Defined Networking," in *International Workshop on Recent Advances in Intrusion Detection*, pp. 161–180, Springer, 2011.
[23] "The POX Controller." https://github.com/noxrepo/pox, July 2010.
[24] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, vol. 1. Springer series in statistics New York, 2001.
[25] "An Instant Virtual Network on your Laptop (or other PC)." http://www.mininet.org, July 2010.
[26] M. Lichman, "UCI machine learning repository." http://archive.ics.uci.edu/ml, 2013.
[27] M. I. T. Lincoln Laboratory, "DARPA Intrusion Detection Evaluation." https://www.ll.mit.edu/ideval/docs/attackDB.html, 2016.
[28] H. Abdi and L. J. Williams, "Principal Component Analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.
[29] "Why accuracy alone is a bad measure for classification tasks, and what we can do about it." https://tryolabs.com/blog/2013/03/25/why-accuracy-alone-bad-measure-classification-tasks-and-what-we-can-do-about-it/, 2017.
[30] A. Botta, A. Dainotti, and A. Pescapè, "A tool for the Generation of Realistic Network Workload for Emerging Networking Scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, 2012.