Flow-Based Intrusion Detection System for SDN

Georgi A. Ajaeiya Nareg Adalian Imad H. Elhajj Ayman Kayssi Ali Chehab
Department of Electrical and Computer Engineering
American University of Beirut
Beirut 1107 2020, Lebanon
{gaa39, nla01, ie05, ayman, chehab}@aub.edu.lb

Abstract— **Software-defined networks (SDN) are vulnerable to most of the attacks that traditional networks are vulnerable to. In addition, SDN has introduced new vulnerabilities through its unique architecture such as those related to the southbound and northbound controller interfaces. In this paper, we introduce a lightweight flow-based Intrusion Detection System (IDS) that periodically gathers statistical information about flows from SDN OpenFlow switches, and analyzes traffic information by extracting and aggregating a set of features. The proposed IDS system proved to be accurate with a high detection rate at 0.98 measured by the F1 score of the classification model and a relatively low false alarm rate.**

Index Terms— Network Security; SDN; NIDS;

I. INTRODUCTION

Software Defined Networking (SDN) has opened a new horizon in the field of networking separating the control and data planes. This separation allows for the centralization of network logic and topology in the Network Operating System (NOS) or the control plane. In turn, such centralization introduces the ability of knowing the network status by querying different statistics from the OpenFlow (OF) switches, and then using these statistics to calculate a set of variables (features) that express traffic flow characteristics across the network. As such, we can classify traffic flows based on the collected features and use the classification model to detect any anomaly in the behavior of the flows.

Classical Network Intrusion Detection Systems (NIDS) rely on monitoring information transmitted from the network hosts, whereas Network Security Monitor systems (NSM) implement traffic analysis on broadcast information of a LAN. In our approach, we try to use the method of NSM to perform intrusion detection without requiring information from the hosts on the network, by taking advantage of the statistics collected by the OF switches. Our aim is to design a network intrusion detection system that is transparent to end hosts (and to intruders).

In this paper, we introduce a method to detect different attacks (such as DoS, HTTP brute force, and SSH brute force) by taking advantage of SDN to measure traffic flow statistics, followed by feature extraction and aggregation to accurately classify traffic flows as being normal or malicious using a supervised machine learning algorithms.

The rest of the paper is organized as follows. In Section II, we review related work. Section III, describes how our proposed system works. In Sections IV and V, the performed experiments and their results are analyzed. Finally, Section VI concludes and discusses future work.

II. RELATED WORK

A. Flow Based IDS

Several research have been proposed regarding statistical intrusion detection and network intrusion detection. The authors of [1] introduced tweaks to refine traffic classification using machine learning techniques, but they used a relatively simple classification method that assumes traffic variable dependency. In [2], the authors used traffic modeling such as volume and connection count in order to detect novel attacks on the network; however, their research was limited to one type of network attacks, namely DoS. The authors of [3] created a traffic classifier from an unsupervised dataset through clustering and then converting the clusters into traffic classes, which introduced a method to use unsupervised datasets in training supervised machine learning models. However, the process had a high overhead for the creation of the clusters.

The authors of [4] suggested using single directional flow-based features to classify traffic, and evaluated the features using Correction Based Filters (CFS). Even though the authors introduced an attack category, they did not elaborate on how they classified traffic in this category. A new statistical technique for protocol flow fingerprinting was introduced in [5] by calculating anomaly scores indicating the statistical difference of an unrecognized traffic from the attributes of a given protocol. However, the introduced method performs better on the client side rather than on the network side, and the performance degrades when the classification is implemented on the network edge.

The authors of [6] introduced a novel system model to discover correlation in traffic flows by arguing that this correlation can increase performance of classification. They introduced what's called Bag of Flows (BoF) contains the correlated traffic flows generated by the same application, which is classified into a predicted class using a neural network classifier. Artificial anomaly generation was used in [7] to create a single model that combines misuse and anomaly detection, it allows the use of supervised learning for both. However, the used technique did not address flow-based classification. In most typical NIDSs the number of features is bounded by the technique used to measure traffic flow statistics, where most methods collect packet-level information. Therefore, a NIDS that uses SDN built-in stats collection capabilities has an advantage of disposing the probes used to collect packet level features around the network. This will increase the performance regarding detection response time.

B. IDS in SDN

Although the separation of data and control planes in SDN allows developers to easily write security applications, the authors of [8] argue that such separation is inefficient in IDS applications because only packet header information can be sent to the controller. Therefore, they developed three modules that are used to capture the entire packets through a secondary interface and accordingly detect if any malicious behavior is observed. However, this method has performance and overhead issues in addition to the need of an additional interface between the two planes. On the other hand, in [9], rather than discarding the middle-boxes used in traditional networks, the authors integrate them within the SDN network to analyze the incoming traffic and hence install flow rules on the switches to prevent malicious flows. However, as the network size increases, more middle-boxes are needed, which is contradictory to the idea of centralizing functions as applications in SDN.

The Atlas framework [10] uses crowdsourcing to obtain flows from mobile agents where the first *N* packets are captured by the Access Point (AP) running the OpenFlow protocol and sent to the controller to be fed to the machine learning trainer. Since OF can send fractions of header information to the controller [11], the *N* packets are either mirrored to the controller or an extension of the switch is applied to store the first *N* packets for each flow then send them to the controller. Once the training is complete, the trained machine learning classifier along with the application-specific policies are installed on the AP, which is fed with the features of flows received for the first time and hence outputs the application type according to which specific flow rules are applied on the switch by the AP or the controller. Although the integration of the classifier in SDN is seamless and scalable, and the C5.0 classifier can handle a large number of flows per second; however, an extension must be applied on the switches to perform the machine learning classification.

The proposed method uses the same logic compared to Atlas framework. However, we pulled the classification technique from the OF switches and performed it as a software layer that runs in parallel with the controller to reduce the classification overhead over the OF switch. Since the method uses built-in statistics collected by the OF switches at 1-second interval, it introduces lower overhead compared to [8], where they used a secondary interface and sent the entire packet through this interface, while the proposed method sends fixed length statistics at fixed intervals. Furthermore, compared to [9], the proposed method does not require additional middle-boxes since we add statistics aggregation and flows classification as additional software that runs in parallel with, and not as part of, the controller. Since the controller and the additional software run on the same machine, the communication time overhead between the two components is negligible.

III. FLOW BASED DETECTION OF ABNORMAL TRAFFIC

Flow-based detection of abnormal traffic, as shown in Fig. 1, is based on extracting a set of predefined features based on 1-second intervals that represent samples of the flow. This step is followed by the aggregation of extracted features, as will be explained in the following subsections. During the 1-second interval, more than one sample of the predefined features can be captured and logged, and most of the aggregated features can be calculated during the lifetime of the flow. This will shorten the time consumed to take a decision when the flow ends.

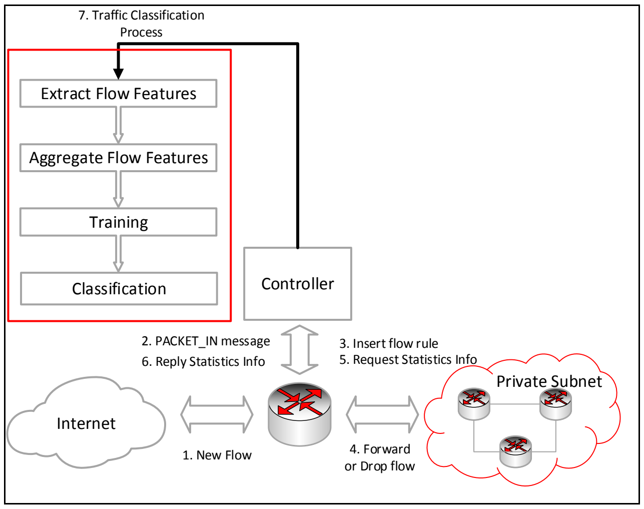

Figure 1 Flow Based detection

By using OF switches as a gateway, the controller will obtain the statistical information from the network edge points. The flow rules that are installed on the gateways for intrusion detection purposes will have the highest priority to guarantee that the incoming flow packets are not processed by general matching flow rules installed by the controller. These general flow rules are installed on the second layer switches, forming the private network, to forward the flows to their correct destination. Since we are using a supervised classification model, we can train the model on multiple types of traffic (normal and abnormal), then use the trained model to classify real network flows. We will elaborate on the training and evaluation phase in the following subsections. The system is scalable since the controller can identify the switches using their DPID and request the statistics from all the OF gateway switches independently then link the collected statistics to each flow by referring to the packet header information.

A. Flow stats Collection

An OF switch such as Open vSwitch (OVS) has flow tables which contain flow entries. Every flow entry consists of idle timeout, priority, match, and instruction fields. When a new flow arrives into an SDN switch, OVS flow table is searched for a match to handle the flow accordingly. If a match exists, the corresponding received packet and byte counters are updated every second, and the flow is handled based on the instruction field of the flow rule. If no match exists, a *PACKET_IN* message is sent to the SDN controller which contains the header information of the first packet of the flow. The obtained header is parsed and the resulting information is used to create a new flow rule which is installed on the OVS to direct the flow. It is then queued in the OVS until the rule is generated and installed. The match field of flows that have ARP, UDP/TCP, or ICMP as their protocol, have the entries represented in Fig. 2.

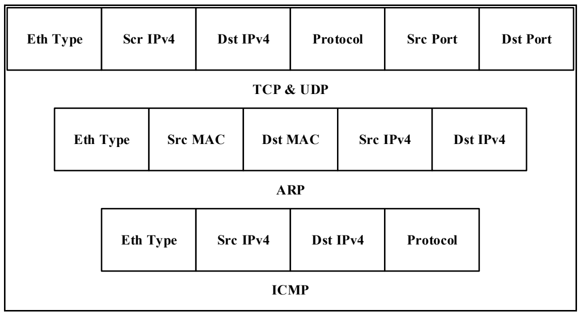

Figure 2 Match Field Entries

The idle timeout of a flow rule represents the maximum number of seconds during which, if no packet is matched, the corresponding rule is removed. Upon the removal of the rule, a *FLOW_REMOVED* message is sent to the controller, which contains flow-related information gathered during the lifetime of the rule such as packet and byte counters. Additionally, every second, a thread running on the controller will request the statistics information of every flow from the OVS switches. Once the request is received, a *FLOW_STATS* reply message is sent, which contains the statistics information of every flow entry in the table. The match field of the *FLOW_REMOVED* or *FLOW_STATS* message is parsed to obtain the fields used to match a flow such as source IP and TCP/UDP port. It logs this information along with the counters to be used for building the classification model.

The proposed technique uses the statistics collection functionality of the OF switches to extract flow features and aggregate them before classification. Since the statistics collection is an existing integral part of the SDN architecture and since the stats are collected only from gateway switches then sent through OF messages (southbound interface) it will not have any performance effects on the traffic flow from the gateway to the target switches. In addition, it won't involve adding any extensions to the existing switches. Furthermore, the classification is performed in parallel using the logged information by an application running with the controller, which does not affect the performance of the SDN Controller by adding any extra functionalities. The number of OF messages will increase over the southbound which can be considered a disadvantage. However, the increase in the number of messages is linear compared to the number of flows over the network. The volume of collected stats will linearly grow as the number of edge switches increases. On the other hand, the stats are sent in Asynchronous OF messages that do not require an acknowledgment. Since SDN control plane can be on a separate dedicated network [11], we can have all OF messages including the IDS messages exchanged over a dedicated network. A separate network will deliver a better performance compared to the control plane managed by the OF switch (in-band controller connection). As for IDS OF messages processing on the controller, it is limited to the hardware specifications of the machine operating the controller. The proposed solution can operate over small and medium networks by implementing these amendments. However, large networks might need a functionality extension in the control plane such as the one proposed in [12] where the authors proposed a hierarchical structure of controllers to handle the overhead produced by adding new features. An IDS can be seen as one of the additional features that runs over the control plane. From a network perspective this is a compromise between performance and security which all network administrators have to deal with. Eventually, using the southbound to send critical information about the flows makes the system transparent since all OVS switches are known to the controller.

B. Feature Extraction and Aggregation

The main advantage of SDN is that OF switches can send statistical information per flow entry to the controller. We can obtain real-time samples at 1-second resolution. Each sample represents 1-second duration of the flow using the set of aggregated features. Additionally, we can get extra information about the flow from reading the *PACKET_IN* messages.

Table 1Extracted Features

Extracted Feature	Definition
Duration (measurements – inter-arrival)	The time in seconds elapsed since the last measurement where packets of the same flow are still being exchanged between peers, and the flow rule on the controller is still active.
Packet count	The number of packets exchanged during the time interval.
Byte count	The number of bytes exchanged during the time interval.
Source IP	The IP of the sending peer
Destination IP	The IP of the receiving peer
Protocol	The used transport layer protocol: TCP, UDP, ICMP, and ARP.
Source port	The port used at the sending peer (extracted from the packet header)
Destination port	The port used at the receiving peer (extracted from the packet header)

Having mentioned that, we introduce the terms used in the feature extraction process:

1) Statistics are measured every 1 second.
2) A flow is considered idle after 2 seconds of inactivity.
3) Multiple samples could be logged for the same flow when not idle. At the same time, samples taken for the pair flow will also be logged, where the pair flow has the same protocol but opposite IP addresses and layer-4 ports.
4) Some features are extracted at the arrival of the first packet of the flow, such as used protocol, source and destination IP, and source and destination port, and are used to install unique flow rules on the OVS switches. The extracted features,

shown in Table 1, describe the flows in general, but can't be used directly for flow classification because of the randomness in logging these features. Therefore, a feature aggregation process should follow the extraction process.

A total of 9 aggregated features are calculated and standardized as shown in Table 2, where the aggregated features are for both same flow and pair flow samples.

Table 2 Aggregated Features

#	Aggregated Feature	Definition
1	Current measurement duration	Number of seconds elapsed since the last measurement
2	Packet count	Number of packets transferred in 1 second.
3	Bytes count	Number of bytes transferred in 1 second.
4	Packet count to duration	Packets to duration ratio.
5	Bytes count to duration	Bytes to duration ratio.
6	Standard deviation of flow duration	The change in flow duration at the current measurement (compared to the rest of the same flow measures).
7	Standard deviation of packets	The change in packet count at the current measurement (compared to the rest of the same flow measures).
8	Standard deviation of bytes	The change in byte count at the current measurement (compared to the rest of the same flow measures).
9	Standard deviation of bytes count to duration	The change in ratio at the current measurement (compared to the rest of the same flow measures).

C. Training and Traffic Classification

The extracted samples from the aggregation phase are used in training a supervised classification model. Therefore, the trained classifier will assign classes for each sample at one second interval. Since each flow may have more than one instance during its lifetime, its behavior will be classified several times. We choose the Bagged Trees [13] classifier to be the system's classification model. We choose 4 other supervised machine learning models to compare against: SVM [14], Decision Trees [15], Random Forest [16], and KNN [17]. By comparing time performance and classification accuracy the proposed classifier will prove to be our best choice. In the training phase the model is built using a training dataset. The training data must represent the ground truth of the flows. Therefore, we have to assure the right class for every training flow. The ground truth of the flows can be known by applying a filter based on domain IPs and names recognized as safe in a controlled environment to extract flows which represent normal traffic. On the other hand, to obtain abnormal flows network attacks can be applied in the same environment and logged. The training samples are produced by processing these known flows and aggregating the feature vectors. After validating the classification performance of the best trained model, it can be used as a part of the system. Updating the model to adapt with normal traffic types will require retraining the model using an updated dataset. This process is done offline to validate the classification accuracy before using the model in a real network. Real-time updates for the classification model are part of our future work. One advantage of such a classification technique is that it can be applied even when the traffic is encrypted because it is independent of the flow packets' payload. We will elaborate on the classification model building part in Section V.

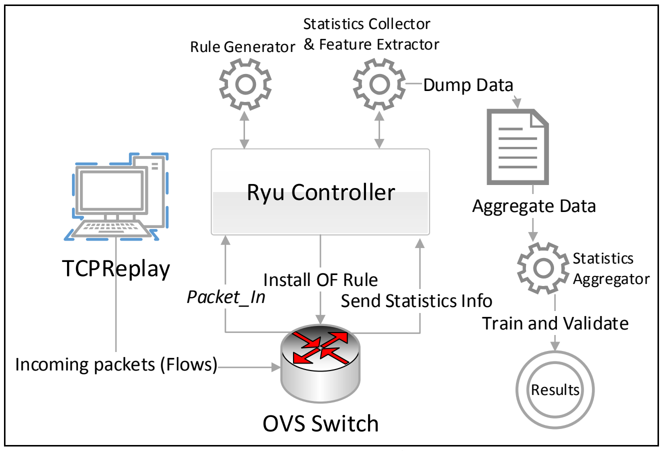

Figure 3 Experimental Setup

IV. Experimental Setup

In the experimental setup shown in Fig. 3, we use *RYU* as the SDN controller and OVS for the switches. We used a single switch to be the network edge for experimental purposes. Traffic data dumps from another experiment [18] are used in our paper, but we had to make sure that we extract the flows that represent normal traffic only. Therefore, we used a whitelist of IP addresses that corresponds to the top 500 domains on Alexa [19] to build a filter using Wireshark. Afterwards, we used the filter to clear the captured traffic in the former experiment and ensured that the output represents normal traffic flows such as browsing, file transfer, instant messaging, and VoIP. On the other hand, we conducted another experiment to collect malicious traffic flows, where we applied multiple kinds of attacks between two hosts on one virtual network connected to the Internet.

Table 3 Traffic Flows

#	Type	No. of Measured Flow Instances
1	TCP DoS	3940
2	HTTP Credential Brute Force	4007
3	Network SYN Scan	5898
4	Port Scan	17296
5	ICMP Flood	4480
6	SSH Brute Force	1033
7	Normal	16624

Types of traffic flows and the size of collected traffic measured by the number of instances for each class can be found in Table 3. Afterwards, we extracted the features by replaying the collected flows using TCP Replay to the SDN network which consisted of a single switch, and ran a Python application in parallel with the controller to log and aggregate the desired features. The aggregation process was performed simultaneously with the controller. Finally, we used the aggregated features' samples to train and validate the classification model.

V. RESULTS AND ANALYSIS

By analyzing the flow instances of the attacks shown in Table 3, and before proceeding to the model training phase we have observed that TCP DoS, Port Scan, SYN Scan and ICMP Flooding have highly correlated features' values and can be grouped under the same attack class, where TCP DoS and ICMP Flooding are straightforward DoS attacks, but examining SYN Scan and Port Scan attacks' traffic revealed a similarity to DoS. In these attacks, large amounts of packets are sent to a single or multiple hosts on the network from multiple virtual sources targeting different ports. Therefore, from a traffic statistics' perspective, these attacks resemble short DoS or packet flooding attacks. Accordingly, our experimental results are based on the following set of classes: **1-DoS**, **2-HTTP credential brute force**, **3-SSH brute force**, and **4-Normal traffic**. First, we used a dataset that contains instances originating from multiple attacks' classes in addition to normal traffic instances. The instances were partitioned into 10 folds each fold represents 10% of the dataset, where 9 folds were used for training and a single fold for testing. The samples were imbalanced in size in each fold. Thus, we made sure that we use an equal number of samples by extracting a smaller dataset that has an equal number of samples for each class. The balanced training dataset has 929 samples for training and 102 for validation randomly picked from each class with respect to the class that has the minimum number of samples. For assuring statistical significance, we created 30 balanced datasets with a size of 4125 samples which consist of randomly picked samples from the larger dataset. Then, we repeated the training and validation and averaged the classification accuracy over the 30 datasets. As mentioned for our choice of supervised classifiers we picked 4 supervised machine learning models to compare against the proposed Bagged Trees classifier.

Table 4 Parameter Values

Parameters/Classifier	SVM	DT	Bagged	RF	KNN
Kernel function	**Poly.**	N/A	N/A	N/A	N/A
Coding	**1 v. 1**	N/A	N/A	N/A	N/A
Poly. order	**3**	N/A	N/A	N/A	N/A
Box constraint	**1**	N/A	N/A	N/A	N/A
Max Splits	N/A	**20**	N/A	N/A	N/A
# Learners	N/A	N/A	**125**	**300**	N/A
Random space size	N/A	N/A	N/A	**7**	N/A
Min. leaf size	N/A	**2**	**1**	**1**	N/A
# Neighbors	N/A	N/A	N/A	N/A	**1**
Distance Metric	N/A	N/A	N/A	N/A	**Corr.**
Distance Weight	N/A	N/A	N/A	N/A	**Inverse**
Break Ties	N/A	N/A	N/A	N/A	**smallest**

A parameter optimization phase is done for each classifier to pick best values for a number of selected parameters. Bayesian Optimization [20] is used to tune the parameter values for each classifier. The generalization function $f(x)$ of the optimization process which needs to be minimized is set to be the FPR. A constraint is set on the minimum averaged F1 score to control the optimization process. According to each parameter type in each classifier a set or a range of values was defined. Hence, the optimization process picks a new value for each parameter at each iteration from the specified range or set. The set of values were bounded to avoid overfitting the training dataset.

We use Matlab 2016b to perform training and evaluation. The Matlab 2016b kit we are using has a parallel processing toolbox enabled. Therefore, we enabled parallel processing on applicable classification models e.g. Bagged Trees, RF, and KNN. The PC specifications which we have Matlab installed on are: Core i7 CPU @ 3.4 GHz and 16 GB RAM running Windows 7 Professional.
Table 4 shows the assigned parameter values for each classifier based on the optimization phase result. We can notice that the picked parameter values are set to avoid overfitting the training dataset. For the SVM classifier the best kernel function is the polynomial function, and the order of the polynomial is 3. KNN classifier distance metric is the correlation measure which is defined as shown in Equation (1).

$$Corr_{knn}(X_n, X_m) = 1 - Corr(X_n, X_m) \quad (1)$$

Where each instance $X = (x_1, x_2, , \dots, x_l)$ is treated as a sequence of values, $Corr(X_n, X_m)$ is the linear correlation between observations. The rest of the parameter values for the rest of the classifiers can be read from the table.

Table 5 ROC Space of 5 Classes

Classifier	Score/Class	1	2	3	4	Avg.
SVM	**TPR**	0.995	0.917	0.822	0.690	0.856
	FPR	0.038	0.105	0.006	0.041	0.047
	F1	0.943	0.821	0.893	0.761	0.854
DT	**TPR**	0.984	0.919	0.851	0.827	0.895
	FPR	0.010	0.052	0.018	0.057	0.034
	F1	0.976	0.885	0.893	0.827	0.895
Bagged	**TPR**	0.996	0.962	0.960	0.924	0.960
	FPR	0.008	0.009	0.016	0.018	0.012
	F1	0.986	0.967	0.955	0.933	0.960
RF	**TPR**	0.994	0.968	0.965	0.925	**0.963**
	FPR	0.009	0.008	0.01	0.01	**0.009**
	F1	0.982	0.971	0.960	0.938	**0.962**
KNN	**TPR**	0.997	0.946	0.939	0.874	0.939
	FPR	0.010	0.020	0.027	0.022	0.019
	F1	0.983	0.942	0.929	0.900	0.938

Table 5 shows the Receiver Operating Characteristic (ROC) space best accuracy points averaged over the 30 balanced datasets for all the classifiers. For each class the True Positive Rate (TPR) and False Positive Rate (FPR) are calculated which refer to the accuracy and recall respectively. We used the *F1 score* to calculate the accuracy as a single measure. The *F1* score is measured as shown in Equation (2).

$$F1 = \frac{2 \times TP}{(2 \times TP + FP + FN)} \quad (2)$$

We can notice that RF has the best F1 score (0.962) among all other classifiers. Additionally, it has an acceptable false alarm rate at 0.9%. Before making a decision to use the Bagged Trees we compared the classifiers performance with respect to time. We measured the training and validation elapsed time for the 30 datasets and averaged the measurements.
Figure 4 shows the ROC curve plot for each class as classified by RF. Each curve represents the prediction TPR vs. FPR for each class when it is set as the positive class. We can notice that RF reports a good prediction accuracy with a low false alarm rate per each class.

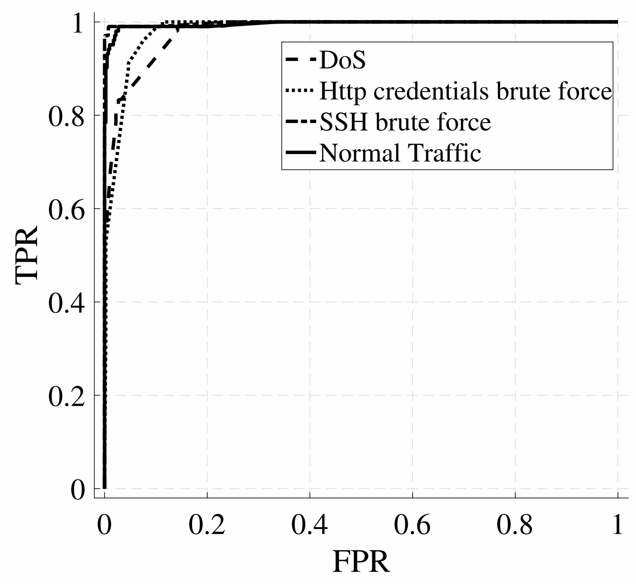

Figure 4 ROC curve (4 classes)

Table 6 shows training and evaluation time results for all classifiers. We can notice that KNN and DT have the lowest training time average at 6 and 14 milliseconds respectively using a training dataset with nearly 4000 samples. On the other hand DT and SVM have the lowest validation time at 1 and 8 milliseconds using a testing dataset with nearly 500 samples.

Table 6 Time Performance

Classifier/Time	Avg. Train (seconds)	Avg. Validation (seconds)
SVM	23.78	0.0079
DT	0.0145	0.001
Bagged	1.3422	0.183
RF	2.234	0.431
KNN	0.060	0.0116

Comparing the top two classifiers with respect to classification and time performance we can notice a compromise. The Bagged Trees classifier is 3% better at classifying the test samples than the KNN classifier. While the KNN classifier is 20 times faster in terms of training the model and 10 times faster in terms of validation. Therefore, using the KNN classifier would deliver a better time performance, while the Bagged Tree classifier have better classification accuracy. Choosing between these two models depends on the criticalness of the classification decision. In security-sensitive networks using a more accurate model with average time performance can have more advantages.

In another experiment, we wanted to test the ability of the best classifier to classify traffic in two classes (normal and abnormal). Therefore, we labeled all traffic instances that originated from all types of attacks to be under a single class. Hence, the dataset contains two type of samples **1-Attacks traffic** and **2-Normal traffic**. We used a bigger dataset as there are no small sized classes. The dataset contains 16500 samples from each class. The Bagged Classifier is trained and validated using 10 folds cross validation with same parameter values obtained from the optimization phase. We can notice the increased accuracy by examining the ROC space shown in Table 7. Where the overall F1 score measured for the binary classification is 0.9834 with an acceptable false alarm rate at 1.6%.

Table 7 ROC Space for 2 Classes

Class	1	2	Avg.
TPR	0.9925	0.9743	0.9834
FPR	0.0257	0.0075	0.016
F1	0.9836	0.9833	**0.9834**

Figure 5 shows the ROC curve plot for the attacks' samples as classified by RF. We notice a high detection accuracy for the attacks samples with a low false alarm rate.

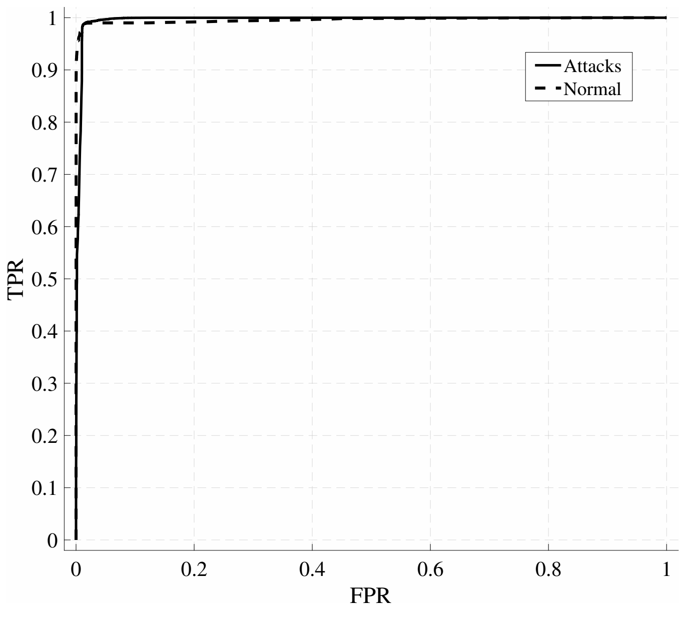

Figure 5 ROC curve (2 classes)

The ROC curve plots shown above demonstrate that the RF classifier is able to classify both normal and attack traffic samples. Classification model training is done within a reasonable time limit. The samples are extracted using the technique mentioned in III.B. In the first experiment, we tested the ability of the RF ensemble to classify samples of different types of attacks along with normal traffic samples. In the second experiment, we validated the RF classification accuracy with two types of classes Normal vs. Attack.

VI. Conclusion

The proposed flow-based IDS for SDN was able to detect malicious traffic with high accuracy. The RF ensemble was able to detect multiple types of attacks and separate these attacks from real traffic. The proposed technique uses the built-in periodically collected flows' statistics from the OF switches to classify the traffic. The system is transparent to the intruders since it runs over the control plane which is an advantage that SDN provides. We were successful in developing a Python application that runs in parallel with the RYU controller to extract flow features from the OF switches, and uses a set of aggregated features to train a supervised classifier based on statistical information of flows only. The supervised classifier has the advantage of classifying encrypted flows since it doesn't require payload information.

As for future work, we plan to make the system adaptive in real time by training it initially with normal traffic and then allowing it to learn malicious traffic, hence allowing us to obtain an anomaly-based intrusion detection rather than a misuse-based one.

Acknowledgments

This research is funded by TELUS Corp., Canada.

References

[1] A. W. Moore and D. Zuev, "Internet Traffic Classification Using Bayesian Analysis Techniques," *Proc. 2005 ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, pp. 50–60, 2005.

[2] J. Caberera, B. Ravichandran, and R. K. Mehra, "Statistical traffic modeling for network intrusion detection," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2000. Proceedings. 8th International Symposium on*, 2000, pp. 466–473.

[3] J. Zhang, Y. Xiang, W. Zhou, and Y. Wang, "Unsupervised traffic classification using flow statistical properties and IP packet payload," *J. Comput. Syst.*, 2013.

[4] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Y. Lee, "Internet traffic classification demystified: myths, caveats, and the best practices," in *Proceedings of the 2008 ACM CoNEXT conference*, 2008, vol. 50, no. 4, pp. 1–12.

[5] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 1, p. 5, 2007.

[6] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, and Y. Guan, "Network traffic classification using correlation information," *Parallel Distrib. Syst. IEEE Trans.*, vol. 24, no. 1, pp. 104–117, 2013.

[7] S. J. Stolfo, P. K. Chan, E. Eskin, M. Miller, and S. Hershkop, "Real time data mining-based intrusion detection," *Proc. DARPA Inf. Surviv. Conf. Expo. II. DISCEX'01*, vol. 1, pp. 89–100, 2001.

[8] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin, and Z. Zhang, "Enabling security functions with SDN: A feasibility study," *Comput. Networks*, vol. 85, pp. 19–35, 2015.

[9] S. Shin, H. Wang, and G. Gu, "A first step toward network security virtualization: From concept to prototype," *IEEE Trans. Inf. Forensics Secur.*, vol. 10, no. 10, pp. 2236–2249, 2015.

[10] Z. A. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt, and G. Noubir, "Application-awareness in SDN," *Proc. ACM SIGCOMM 2013 Conf. SIGCOMM - SIGCOMM '13*, p. 487, 2013.

[11] *OpenFlow Switch Specification*, Version 1. Open Networking Foundation.

[12] J. McCauley, A. Panda, M. Casado, T. Koponen, and S. Shenker, "Extending SDN to large-scale networks," *Open Netw. Summit*, pp. 1–2, 2013.

[13] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

[14] C. Cortes and V. Vapnik, "Support-Vector Networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[15] J. R. Quinlan, "Induction of Decision Trees quinlan.pdf," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.

[16] L. (University of C. Breiman, "Random forest," *Mach. Learn.*, vol. 45, no. 5, pp. 1–35, 1999.

[17] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *Am. Stat.*, vol. 46, no. 3, pp. 175–185, 1992.

[18] "Network forensics training, challenges and contests." [Online]. Available: http://www.netresec.com/?page=PcapFiles. [Accessed: 01-Jan-2015].

[19] "Alexa top 500 global sites." [Online]. Available: http://www.alexa.com/topsites. [Accessed: 01-Jan-2015].

[20] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," *Adv. Neural Inf. Process. Syst. 25*, pp. 1–9, 2012.