# Early Detection of Intrusion in SDN

Md. Shamim Towhid
*Department of Computer Science*
*University of Regina*, Canada
mty754@uregina.ca

Nashid Shahriar
*Department of Computer Science*
*University of Regina*, Canada
nashid.shahriar@uregina.ca

***Abstract*—An intrusion detection system (IDS) is an essential component of any modern network. The purpose of an IDS is to detect intrusion and generate appropriate alarms so that the intrusion can be mitigated. Implementing an IDS in a Software Defined Network (SDN) is easier since an SDN controller has a centralized view of the whole network. Researchers have made many efforts to use machine learning (ML) for developing network-based IDS in SDN. The network-based IDS analyzes different characteristics of incoming network traffic to detect intrusion. Early detection of intrusion is crucial for an IDS because if the intrusion is not detected quickly enough, it can cause severe damage, such as data breaches and service shutdowns. This paper focuses on detecting intrusion in SDN as early as possible using real-time flow-based features. Our aim is to detect intrusion with less amount of packets per flow, which not only facilitates early intrusion detection but also is useful when an intrusion flow has less number of packets. We show that although ML models perform well in offline training on a dataset, their performance decreases ∼25% when fewer packets are used to generate features for the ML model. In all our experiments, a simple Random Forest (RF) algorithm outperforms a complex deep learning model on a publicly available dataset for intrusion detection in SDN.**

***Index Terms*—Real-time intrusion detection, SDN, machine learning, flow-based features**

## I. Introduction

Networks are becoming increasingly complex to support heterogeneous needs of the varieties of applications and enormous volume of traffic. Manual management processes are becoming tedious and error-prone to manage these complex networks. Therefore, modern networks are shifting towards automation and softwarization using technology like Software-Defined Networking (SDN) [1]. SDN is a network architecture that allows centralized control over the network by decoupling the control plane from the data plane. Here, the control plane refers to the activities that define network operations, such as traffic routing, failover recovery, congestion control, etc. On the other hand, the data plane is only responsible for forwarding traffic based on the configurations supplied by the control plane. Since the control plane has a centralized network view, it can make informed decisions for operating and managing the network. Because of this advantage in SDN, it is being deployed widely in different networks such as data-centers, enterprise networks, and wide-area networks.

As SDN deployment in different networks is gaining momentum, so is the attention of attackers. Since the control plane of an SDN is responsible for managing the network, it can become a common target for attackers. An attacker can intrude on a host in an SDN to gain unauthorized access and steal sensitive information with a goal to attack the controller eventually. Attacking the controller of an SDN network, significant harm can be done as the controller is considered the brain of an SDN. Probing, Web attacks, Botnet, and Brute Force attacks are some examples of intrusion in an SDN [2]. An IDS is a common way of defending a network from such attacks. While host-based IDSs exploit system logs and configurations, network-based IDSs analyzes traffic pattern and packet characteristics to detect intrusions. Once the intrusion is detected, there are several ways to mitigate it, such as blocking the malicious traffic or constraining the resources for the suspicious user.

Machine learning (ML) techniques have widely been used to develop network-based IDSs as they facilitate discovering hidden patterns in network data [3] Most existing work focuses on achieving high accuracy using a new ML algorithm or a feature selection method in offline training. However, less attention is given on the early detection of an intrusion or on intrusion carried out using fewer number of packets. Early detection of an intrusion is critical as it can trigger mitigation activities sooner before any harm is done to the network. On the other hand, smart attackers may want to fool an IDS by sending fewer number of packets. To fill this gap in the literature, in this paper, we study ML-based IDSs' performance in terms of early detection and in the presence of fewer packets from the attacker.

This paper aims to develop an IDS for SDN using ML techniques to detect intrusions as early as possible by using real-time packet-level statistics from an SDN. We use only an initial subset of packets from a flow between two hosts to calculate the features because our goal is to detect intrusions sooner with fewer packet samples. Here, a flow is defined as a collection of packets between two hosts with the same source IP address, destination IP address, source port number, destination port number, and protocol. Again, the flow of packets can be bidirectional, meaning that we consider both forward and backward packets between two hosts to be part of the same flow. Our experiments using real-time features from an SDN-based intrusion detection dataset show that RF can outperform complex deep-learning models, such as, Long short-term memory (LSTM) [10] by a large margin (∼9%) when fewer training samples are present in the training set (shown in Table II). We also show that while ML models perform well in offline training on a dataset, their accuracy

decreases when fewer packets are used to calculate the features.

The following section provides our literature review. Section III describes our ML model selection process for early intrusion detection. Our dataset preparation for early intrusion detection, along with all the experiments for early intrusion detection, are discussed in Section IV. Finally, we conclude the paper by discussing how ML models can be leveraged in an SDN environment for intrusion detection, followed by some possible research directions.

## II. Related Work

The research community has made many attempts to solve the intrusion detection problem. Most of the recent works focus on ML techniques to detect intrusion. Researchers use both traditional ML-based and deep-learning algorithms to detect intrusion. Some research works combine deep learning and traditional ML algorithms to get the best from both worlds. In this section, we discuss some of these works.

The authors in [4] address the lack of appropriate datasets for intrusion detection in SDN networks. As a centralized network architecture, SDN traffic has some unique characteristics. The SDN architecture also introduces some new attack vectors in addition to the attack vectors of conventional networks. Since the intrusion detection system's performance highly depends on the dataset's characteristics, an SDN-specific dataset is necessary. InSDN [4] is the most recent publicly available dataset for intrusion detection in an SDN environment.

The authors in [2] use Xgboost [5] to detect intrusion and benign traffic on the InSDN dataset. Xgboost is a decision tree-based boosting algorithm that combines multiple decision trees sequentially to classify the traffic. According to paper [2], Xgboost is the most successful algorithm for intrusion detection in terms of accuracy, although it requires a lot of labeled data and computational resources. According to the survey shown in [2], deep learning methods can not outperform Xgboost considerably. Our experiments also use the InSDN dataset. This dataset and our data preparation steps are described in section IV-A.

The authors in [6] combine Convolutional Neural Network (CNN) with other traditional machine learning algorithms such as RF, K-Nearest Neighbor (KNN), and Support Vector Machine (SVM) for the intrusion detection task in an SDN. A new regularization method is proposed along with the hybrid model to handle the overfitting problem. Therefore, the proposed method works well in binary and multi-class classification. Similar to [2], paper [6] also uses the InSDN dataset for the experiments along with the other two publicly available benchmark datasets collected from traditional/non-SDN networks. In all the datasets, the proposed method outperforms the state-of-the-art approach according to the experimental results shown in [6]. However, our experiments found that the RF algorithm outperforms the proposed method using the same features from the InSDN dataset. Details of this experiment are given in section III.

In [7], the authors formulate the intrusion detection problem as anomaly detection. The idea in anomaly or outlier detection is to train a machine learning model only to identify benign data patterns. Then during the evaluation, we can compare the difference between the learned pattern of benign data and the pattern in the test data. The model classifies the data as an anomaly if the difference between the learned pattern and the test data pattern exceeds a predefined threshold. One-class SVM (OC-SVM) [8] is an algorithm to train a machine-learning model for anomaly detection. OC-SVM cannot operate with a massive and high-dimensional dataset. Therefore, [7] uses an LSTM autoencoder to represent data in low dimensional space. Then the low-dimensional data is used to train the OC-SVM model. The accuracy of anomaly detection algorithms highly depends on the purity of the training data. The model cannot detect anomalies with reasonable accuracy if the training data contains some anomalous samples.

An Artificial Intelligence (AI) system is developed in [9] to propose an intrusion detection system based on only flow-based features in an SDN environment. The AI system periodically gathers statistical information about flows from the SDN OpenFlow [19] switches. Then, it analyzes traffic information by extracting and aggregating a set of pre-defined features. The authors in [9] show that only flow-based features are enough to detect intrusion with high accuracy. In our experiments, we also use only flow-based features. However, instead of collecting features periodically, we collect features by limiting the number of packets, allowing us to detect attacks earlier. Attacks like DoS/DDoS usually send a lot of packets within a concise amount of time. Therefore, collecting features from the packet level should speed up the intrusion detection process.

A deep learning algorithm is proposed in [10] to detect DDoS attacks that use LSTM. LSTM models work well for long sequential input data. The authors in [10] consider the flow of packets as sequential input to the LSTM model. Although the LSTM model shows good performance when a large number of packets are given as input to the model, it falls short when we limit the number of packets per flow in our experiments.

Authors in [11] propose a lightweight CNN model to detect DDoS attacks. In addition to the proposed deep learning model, their focus is on the early detection of intrusion. At first, a time window is defined, and the maximum packet number is fixed within that time window. Then the authors calculate features using the chosen number of packets and feed it to the proposed CNN algorithms. On the other hand, in our experiments, we fix the number of packets per flow instead of fixing the time window, which enables us to detect attacks that involve a small number of packets.

Most earlier works focus on achieving high accuracy using a new ML algorithm or a feature selection method in offline training. In contrast to the state-of-the-art, we aim to detect intrusion as early as possible using real-time features. We use the InSDN dataset to prepare data to facilitate early intrusion detection. We vary the number of packets used from a flow to calculate our features and train ML models to detect intrusion.

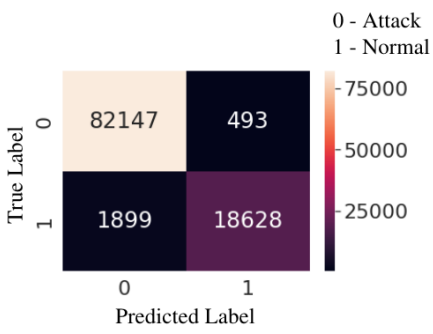


Fig. 1. Confusion Matrix of CNN+RF

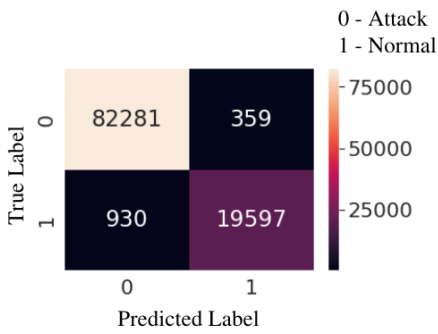


Fig. 2. Confusion Matrix of Random Forest

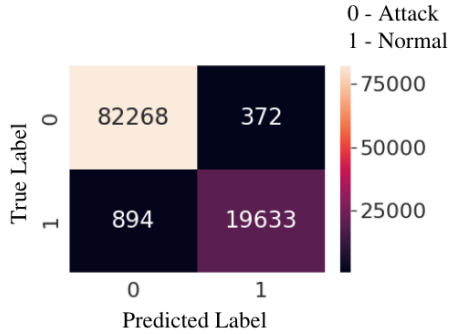


Fig. 3. Confusion Matrix of Xgboost

Our experiments show that the RF algorithm can achieve good accuracy compared to the LSTM-based deep learning approach when features are calculated by limiting the number of packets per flow.

## III. ML Model Selection for Intrusion Detection

Our first goal is to select an appropriate machine learning algorithm that provides a good trade-off between accuracy and computational complexity based on the InSDN dataset. As mentioned in [4], the InSDN dataset is collected using a testbed of four virtual machines. The testbed is prepared using the Mininet [13] and ONOS [15] SDN controller. Data is available both in CSV format and in pcap format. There are seven types of attack traffic in the InSDN dataset. These include web attacks (cross-site scripting, SQL injection), password-guessing attacks, malware (botnet attack), probing (version scan, port scan, service discovery), exploitation, DoS, and DDoS.

We have implemented the hybrid method proposed in [6] and use the same set of 9 features as [6] in our experiment. As mentioned in section II, the authors of the paper [6] combine CNN and RF for training. There are two stages of training according to [6]. At first, CNN layers are trained in a binary classification setup where the last layer of the model is a fully connected layer with two neurons. Next, the final fully connected layer is discarded in the second stage of training. Hence, the output of the CNN model is a 128-dimensional feature vector. The RF algorithm is trained on this feature vector instead of the actual features. The result of this method is shown in Fig. 1. We select the confusion matrix for evaluating the ML model because it gives us a complete view of its performance. For example, we can see from Fig. 1 that 1899 benign traffic is misclassified as attack by the combined model, whereas 493 attack traffic is misclassified as normal. Since we can reproduce the result of [6], our next goal was to improve this result which can help us select a suitable ML model to be used in our early intrusion detection.

We then apply RF, and Xgboost algorithms on the same InSDN dataset inspired by the observation in [2] that mentions decision tree-based models generally perform better for intrusion detection. Results of RF and Xgboost algorithms are shown in Fig. 2 and Fig. 3, respectively. Note that although there is a class imbalance in the InSDN dataset, we do not perform any additional steps to balance the data. This is because, in [6], the authors do not balance the data for binary classification. Therefore, for a fare comparison, we do not balance the classes in the dataset.

We can see that both RF and Xgboost algorithms outperform the proposed method of [6] using the exact dataset and features for training and testing. The CNN+RF method, presented in [6], is more computationally expensive than the standalone RF or Xgboost model. Hence, training and inference time is more extensive in the CNN+RF model. On the other hand, although Xgboost is showing slightly better performance than RF, Xgboost is more computationally expensive than RF. The total training time of RF is 19.58 seconds, whereas Xgboost takes 57.31 seconds to complete the training. Therefore, we select the RF algorithm for early intrusion detection described in the following section.

## IV. Early Detection of Intrusion in SDN

In this section, our focus is shifted toward the early detection of intrusion in SDN. First, we describe the data preparation steps for early intrusion detection, then our experiments and results are described.

### A. Dataset preparation

We prepare multiple datasets for our experiments using the InSDN dataset mentioned earlier. The data is available in pcap (an extension of the file used for network packet capturing) and CSV files in the InSDN dataset. We use the pcap files to prepare our dataset. At first, we divide all the flows present in the InSDN dataset into two sets. One for training and the other to test the ML model. We use a 70:30 split to prepare the training and test set. Since we aim to detect intrusion as early as possible, we prepare datasets with fewer packets per flow. For example, if a flow has an $\mathcal{M}$ number of packets, first, we divide that flow into multiple sub-flows using $k$ ($k < \mathcal{M}$) number of packets in each sub-flow. So, there are $\mathcal{N} = \lfloor \mathcal{M}/k \rfloor$ sub-flows generated from a flow with $\mathcal{M}$ packets. Then, to prepare the dataset, we take the first $n \leq \mathcal{N}$ sub-flows from each flow. If the sub-flow number of a particular flow is less than $n$, then we ignore that flow. To prepare multiple datasets, we try different values for $k$ and $n$. A dataset is represented

as $\mathcal{D}_{k,n}$ throughout the paper. To create sub-flows from the InSDN dataset, we use a python package named Scapy [17]. After creating sub-flows from the actual flow, we calculate the same 9 features mentioned in [6]. These features are protocol, duration of $k$ packets, number of forward packets among the $k$ packets, the total length of the forward packets, mean of the length of forward packets, packet rate, byte rate, mean of inter-arrival time, and standard deviation of inter-arrival time. Table I shows the number of sub-flows per class (attack (a) and benign (b)) in the training and test set for different values of $k$ and $n$. All our prepared datasets and source code for all the experiments are publicly available in [14].

TABLE I
CLASS DISTRIBUTIONS FOR DIFFERENT VALUES OF $k$ AND $n$

| $k$ | $n$ | Train set size | Test set size |
|---|---|---|---|
| 3 | 1 | 978 (a)<br>5953 (b) | 443 (a)<br>2573 (b) |
| | 2 | 1955 (a)<br>11907 (b) | 886 (a)<br>5146 (b) |
| | 3 | 2932 (a)<br>17861 (b) | 1329 (a)<br>7719 (b) |
| | 4 | 3909 (a)<br>23815 (b) | 1772 (a)<br>10292 (b) |
| | 5 | 4886 (a)<br>29769 (b) | 2215 (a)<br>12865 (b) |
| 4 | 1 | 5488 (a)<br>5319 (b) | 2302 (a)<br>2290 (b) |
| | 2 | 10976 (a)<br>10638 (b) | 4604 (a)<br>4580 (b) |
| | 3 | 16464 (a)<br>15957 (b) | 6906 (a)<br>6870 (b) |
| | 4 | 21952 (a)<br>21276 (b) | 9208 (a)<br>9160 (b) |
| | 5 | 27440 (a)<br>26595 (b) | 11510 (a)<br>11450 (b) |
| 5 | 1 | 397 (a)<br>4785 (b) | 160 (a)<br>2042 (b) |
| | 2 | 794 (a)<br>9570 (b) | 320 (a)<br>4084 (b) |
| | 3 | 1191 (a)<br>14355 (b) | 480 (a)<br>6126 (b) |
| | 4 | 1588 (a)<br>19140 (b) | 640 (a)<br>8168 (b) |
| | 5 | 1985 (a)<br>23925 (b) | 800 (a)<br>10210 (b) |
| 6 | 1 | 279 (a)<br>4325 (b) | 138 (a)<br>1842 (b) |
| | 2 | 558 (a)<br>8650 (b) | 276 (a)<br>3684 (b) |
| | 3 | 837 (a)<br>12975 (b) | 414 (a)<br>5526 (b) |
| | 4 | 1116 (a)<br>17300 (b) | 552 (a)<br>7368 (b) |
| | 5 | 1395 (a)<br>21625 (b) | 690 (a)<br>9210 (b) |

We select the minimum value for $k$ as 3 because there are TCP flows in the dataset. We know that the first few packets for a TCP connection are handshake packets. Therefore, detecting an attack or benign traffic using the first one or two packets would be meaningless. ICMP and UDP packets are also present in the dataset.

From Table I, it is clear that the attack flows have fewer packets than benign flows. As a result, when we increase the value of $k$, the number of sub-flows decreases for the attack class. The only exception is when $k = 4$, we see a large number of attack traffic in the dataset compared to $k = 3$. That is because, for every value of $k$, we create the dataset for $n = 5$ first. Then we create datasets for other values of $n$ as a subset of $\mathcal{D}_{k,5}$. The reason for doing this is we want to take only those flows that can be divided into at least 5 sub-flows. When $k = 4$ and $n = 5$, we are getting more flows that can be divided into at least 5 sub-flows. On the bright side, when $k = 4$, we see a balance between attack and benign traffic, whereas, for other cases, there is an imbalance between attack and benign class.

### *B. Evaluation of ML models*

We train two ML models on the datasets shown in Table I. RF is selected because of its better performance in our first experiment. We use the default hyperparameters of the RF model defined in the sklearn [16] library for all our experiments. We implement the LSTM-based deep learning model proposed in [10]. In [10], the authors calculate packet-level features and use them as sequential input. In our case, we use the same model on our flow-level features created for early

TABLE II
EVALUATION METRICS ON THE EARLY DETECTION DATASET

| $k$ | $n$ | Model | Accuracy(%) | Precision(%) | Recall(%) | F1(%) |
|---|---|---|---|---|---|---|
| 3 | 1 | LSTM | 85.31 | 42.66 | 50.00 | 46.04 |
| | | RF | **98.84** | **98.73** | **96.61** | **97.63** |
| | 2 | LSTM | 91.21 | 91.39 | 71.77 | 77.37 |
| | | RF | **98.46** | **97.41** | **96.39** | **96.89** |
| | 3 | LSTM | 92.18 | 92.86 | 74.76 | 80.44 |
| | | RF | **98.34** | **97.20** | **96.13** | **96.65** |
| | 4 | LSTM | 92.41 | 92.28 | 76.02 | 81.43 |
| | | RF | **98.23** | **97.18** | **95.67** | **96.40** |
| | 5 | LSTM | 94.89 | 93.38 | 85.42 | 88.81 |
| | | RF | **98.27** | **96.90** | **96.15** | **96.52** |
| 4 | 1 | LSTM | 90.14 | 90.83 | 90.15 | 90.10 |
| | | RF | **99.24** | **99.24** | **99.24** | **99.24** |
| | 2 | LSTM | 95.32 | 95.37 | 95.32 | 95.32 |
| | | RF | **99.35** | **99.35** | **99.35** | **99.35** |
| | 3 | LSTM | 96.32 | 96.38 | 96.32 | 96.32 |
| | | RF | **99.40** | **99.41** | **99.41** | **99.40** |
| | 4 | LSTM | 97.15 | 97.18 | 97.15 | 97.15 |
| | | RF | **99.43** | **99.43** | **99.43** | **99.43** |
| | 5 | LSTM | 97.65 | 97.66 | 97.65 | 97.65 |
| | | RF | **99.42** | **99.42** | **99.42** | **99.42** |
| 5 | 1 | LSTM | 92.73 | 46.37 | 50.00 | 48.11 |
| | | RF | **98.46** | **97.78** | **90.53** | **93.80** |
| | 2 | LSTM | 92.73 | 46.37 | 50.00 | 48.11 |
| | | RF | **98.57** | **96.66** | **92.46** | **94.44** |
| | 3 | LSTM | 92.73 | 46.37 | 50.00 | 48.11 |
| | | RF | **98.49** | **96.39** | **92.08** | **94.11** |
| | 4 | LSTM | 92.73 | 46.37 | 50.00 | 48.11 |
| | | RF | **98.54** | **96.70** | **92.15** | **94.29** |
| | 5 | LSTM | 92.73 | 46.37 | 50.00 | 48.11 |
| | | RF | **98.52** | **96.84** | **91.89** | **94.20** |
| 6 | 1 | LSTM | 93.03 | 46.52 | 50.00 | 48.19 |
| | | RF | **98.48** | **96.51** | **91.48** | **93.83** |
| | 2 | LSTM | 93.03 | 46.52 | 50.00 | 48.19 |
| | | RF | **98.54** | **97.25** | **91.17** | **93.97** |
| | 3 | LSTM | 93.03 | 46.52 | 50.00 | 48.19 |
| | | RF | **98.74** | **97.391** | **92.73** | **94.88** |
| | 4 | LSTM | 93.03 | 46.52 | 50.00 | 48.19 |
| | | RF | **98.71** | **97.15** | **92.69** | **94.79** |
| | 5 | LSTM | 93.03 | 46.52 | 50.00 | 48.19 |
| | | RF | **98.72** | **96.67** | **93.21** | **94.86** |

intrusion detection. These two selected models' performances are compared in Table II. We use accuracy, precision, recall, and F1 score as evaluation metrics. These are standard metrics to evaluate an ML model for classification. One thing to note here is the LSTM model takes three-dimensional data as input. Therefore, we create input data for LSTM by combining the sub-flows in one dimension. So, the input shape for the LSTM model is $(\mathcal{B} \times n \times f)$. Here, $\mathcal{B}$ is the batch size, and $f$ is the number of features which is 11 in our case. The feature size is 11 because we use the one-hot encoding of the protocol feature. Since three protocols are present in the dataset, we get 11 features after one-hot encoding of the protocol feature.

From Table II, we see that RF outperforms the LSTM model for all values of $k$ and $n$. Furthermore, when there is a low number of training data ($\mathcal{D}_{3,1}, \mathcal{D}_{5,1}, \mathcal{D}_{5,2}$), the LSTM model is overfitting. Therefore, the recall score becomes precisely 50%. That means the LSTM model predicts the same class (attack or benign) for all the data. LSTM performs best when the training size is reasonably high, and there is a balance in the training data, i.e., $\mathcal{D}_{4,5}$. On the other hand, RF is doing better in all cases. From this experiment, we can conclude that RF is practical when there is a small number of training data. Even when there is an imbalance in the data, RF is doing better according to the result in Table II. RF model takes ∼4.3 microseconds on average to classify a data point in the test set, whereas the LSTM model takes ∼62.5 microseconds. Therefore, considering the low inference time, the RF model is preferable to the LSTM model for early intrusion detection.

TABLE III
NEW DATASET SIZE AFTER ADDING DATA FROM OTHER $n$ VALUES

| $k$ | $n$ | Train set size | Test set size |
|---|---|---|---|
| 5 | 1 | 79538 (a)<br>28930 (b) | 800 (a)<br>10210 (b) |
| | 2 | 71920 (a)<br>16328 (b) | 800 (a)<br>10210 (b) |
| | 3 | 3681 (a)<br>19680 (b) | 800 (a)<br>10210 (b) |
| | 4 | 1876 (a)<br>22356 (b) | 800 (a)<br>10210 (b) |
| | 5 | 1985 (a)<br>23925 (b) | 800 (a)<br>10210 (b) |

We can compare RF with LSTM in the previous experiment because we use the same test dataset for RF and LSTM. However, we cannot compare different RF models in Table II, which is essential to understand the effect of limiting the number of sub-flows($n$) in the training set. Therefore, we design another experiment to understand the effect of limiting the number of sub-flows. In other words, by limiting the number of sub-flow, we are limiting the total number of packets from a single flow in the training set. In this experiment, we fix the value of $k$ to 5. From Table I, we see that when $k = 5$, if we vary $n$ from 1 to 5, the total training size increases gradually. Therefore, if we train the RF model on these datasets by fixing the test set, different training set sizes can affect the performance of the RF model. For a fare comparison, we added sub-flows from other values of $n$ to the dataset where the total training size is less than $\mathcal{D}_{5,5}$. The new dataset size is shown in Table III. We use the same test set to evaluate all the RF models in this experiment because the goal is to compare them.

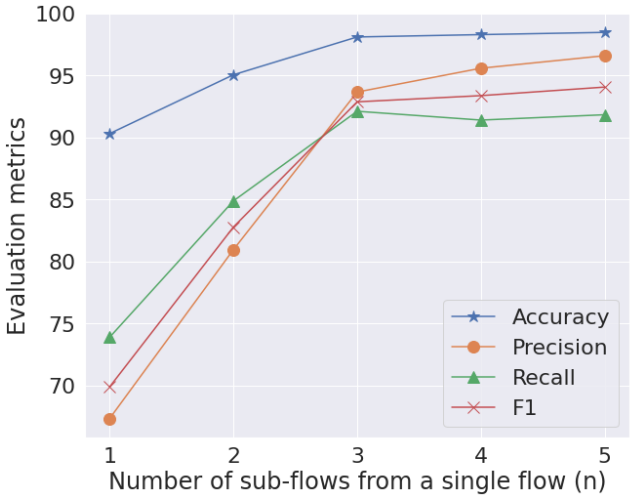


Fig. 4. Evaluation metrics of RF models on new dataset

Fig. 4 shows the evaluation metrics for all the RF models. The same experiment is run five times, and the average for each evaluation metric is shown in Fig. 4. Fig. 4 shows that for lower values of $n$, the performance of the RF model is poor compared to RF models trained with larger $n$ values. When the RF model is trained with fewer total packets per flow ($k = 5, n = 1$), the model's accuracy decreases, although the total training size is larger than other datasets ($k = 5, n = 5$). The precision score has improved the most when the value of $n$ increases. The precision score is an indicator of the false positive rate of the model. A higher precision score indicates that the model has a low false positive rate. When $n = 5$, the RF model achieves the highest precision score. A false positive is crucial when developing an IDS because it can affect a service provider's quality of Service (QoS). From this experiment, we conclude that a complex ML model does not always perform better than traditional ML algorithms, and further research is required to improve the performance of early intrusion detection.

## V. DISCUSSION AND RESEARCH DIRECTION

For real-time intrusion detection, features like flow duration and total forward packet length are not well suited, although many researchers use them in offline training. In this paper, we propose to use real-time features such as packet count rate and average packet size and evaluate the performance of the RF model (best performer in offline settings) for early detection of intrusion in SDN. Our evaluation result shows that the accuracy of the RF model trained on the features of the entire time series of flows decreases as we want earlier detection. To improve this performance, few-shot learning can be used. Few-shot learning is an ML method that makes predictions based on a limited number of training samples. This area of ML,

along with Feature engineering approaches, can be explored to improve the performance of the early detection problem.

The most common way to detect intrusion in SDN is to send a FlowStatsRequest message periodically to the data plane and extract the required features from the reply message. This message exchange between the control and data planes introduces overhead in the network. This overhead may be negligible in a small-scale network. But for an extensive network, it may introduce additional overhead. If we can move our detection to the data plane, this overhead is no longer an issue. Data plane programming using P4 [18] switches can be leveraged for this purpose. Simple algorithms like RF can be implemented on the programmable switch easily compared to the complex deep learning models.

It is interesting to see what happens if entirely unknown attack samples are present in the test set. For example, if we remove all the flows of the Web attack from the training set but the test set contains flows from the Web attack. This is especially applicable to zero-day attacks where attackers inject a novel attack that is unseen by the IDS. The recent development around transfer learning can be explored to detect such zero-day attacks.

Our experiments consider only two classes (Attack and Benign). But in reality, if we want to mitigate different types of attacks on the network differently, we need to identify the type of attack. In that case, we need to consider multi-class classification. Traditional ML algorithms, such as RF, cannot do well on multi-class classification because the samples from different attack class share similar features. Deep learning models can be helpful in multi-class scenarios because these models can learn the complex patterns in the data.

## VI. Conclusion

In this paper, we study real-time intrusion detection in SDN, focusing on detecting intrusion as early as possible using real-time flow-based features. By using real-time features from an SDN-based intrusion detection dataset, we show that although ML models perform well in offline training on a dataset, the accuracy of the ML model decreases for early intrusion detection. We also show that a simple Random Forest algorithm can beat powerful deep-learning models. We conclude that more research is needed to fully understand the spectrum of this problem, as several possible research directions are outlined as part of our future work. We hope this work will ignite the research community to address the problem of early intrusion detection in the SDN environment.

## Acknowledgement

This work was supported in part by funding from the Innovation for Defence Excellence and Security (IDEaS) program from the Department of National Defence (DND).

## References

[1] F. Z. Yousaf, M. Bredel, S. Schaller and F. Schneider, "NFV and SDN—Key Technology Enablers for 5G Networks," in IEEE Journal on Selected Areas in Communications, vol. 35, no. 11, pp. 2468-2478, Nov. 2017, doi: 10.1109/JSAC.2017.2760418.

[2] Quang-Vinh Dang. 2021. Intrusion Detection in Software-Defined Networks. In Future Data and Security Engineering: 8th International Conference, FDSE 2021, Virtual Event, November 24–26, 2021, Proceedings. Springer-Verlag, Berlin, Heidelberg, 356–371.

[3] Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. Journal of Internet Services and Applications, 9(1), 1-99.

[4] M. S. Elsayed, N. -A. Le-Khac and A. D. Jurcut, "InSDN: A Novel SDN Intrusion Dataset," in IEEE Access, vol. 8, pp. 165263-165284, 2020, DOI: 10.1109/ACCESS.2020.3022633.

[5] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). Association for Computing Machinery, New York, NY, USA, 785–794.

[6] Mahmoud Said ElSayed, Nhien-An Le-Khac, Marwan Ali Albahar, Anca Jurcut, A novel hybrid model for intrusion detection systems in SDNs based on CNN and a new regularization technique, Journal of Network and Computer Applications, Volume 191, 2021, 103160, ISSN 1084-8045, https://doi.org/10.1016/j.jnca.2021.103160.

[7] Mahmoud Said Elsayed, Nhien-An Le-Khac, Soumyabrata Dev, and Anca Delia Jurcut. 2020. Network Anomaly Detection Using LSTM Based Autoencoder. In Proceedings of the 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet '20). Association for Computing Machinery, New York, NY, USA, 37–45. https://doi.org/10.1145/3416013.3426457.

[8] Mũnoz-Marí, Jordi, et al. "Semisupervised one-class support vector machines for classification of remote sensing data." IEEE transactions on geoscience and remote sensing 48.8 (2010): 3188-3197.

[9] G. A. Ajaeiya, N. Adalian, I. H. Elhajj, A. Kayssi, and A. Chehab, "Flow-based Intrusion Detection System for SDN," 2017 IEEE Symposium on Computers and Communications (ISCC), 2017, pp. 787-793, DOI: 10.1109/ISCC.2017.8024623.

[10] X. Yuan, C. Li and X. Li, "DeepDefense: Identifying DDoS Attack via Deep Learning," 2017 IEEE International Conference on Smart Computing (SMARTCOMP), Hong Kong, China, 2017, pp. 1-8, doi: 10.1109/SMARTCOMP.2017.7946998.

[11] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martínez-del-Rincón and D. Siracusa, "Lucid: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection," in IEEE Transactions on Network and Service Management, vol. 17, no. 2, pp. 876-889, June 2020, doi: 10.1109/TNSM.2020.2971776.

[12] B. A. Tama, M. Comuzzi and K. Rhee, "TSE-IDS: A Two-Stage Classifier Ensemble for Intelligent Anomaly-Based Intrusion Detection System," in IEEE Access, vol. 7, pp. 94497-94507, 2019, DOI: 10.1109/ACCESS.2019.2928048.

[13] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks, 2010.

[14] Early Intrusion Detection in SDN repository on GitHub. [Online]. Available: https://github.com/shamimtowhid/Early-Intrusion-Detection-in-SDN.

[15] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, W. Snow, and G. Parulkar, "ONOS: Towards an open, distributed SDN os," in Proc. 3rd Workshop Hot Topics Softw. Defined Netw., 2014, pp. 1–6.

[16] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthias Perrot, Édouard Duchesnay. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011. [Online]. Available: https://scikit-learn.org/. [Accessed: 1-Feb-2023].

[17] Philippe Biondi and Arnaud Ebalard. Scapy: a packet manipulation tool for computer networks. [Online]. Available: https://scapy.net/. [Accessed: 1-Feb-2023].

[18] P4 (2022) Open Networking Foundation. Available at: https://opennetworking.org/p4 (Accessed: March 10, 2023).

[19] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69-74, 2008. [Online]. Available: https://www.openflow.org/. [Accessed: 1-Feb-2023].