Network Traffic Feature Behaviour Augmentation Fusion based Attacks Classification for Intrusion Detection System in SDN Framework

Duong Van Dan[1], Tran Nam Khanh[1], Ta Minh Thanh[1]
[1] Le Quy Don Technical University
Correspondence: Ta Minh Thanh, thanhtm@mta.edu.vn
Communication: received 02 Dec 2023, revised 22 Jan 2024, accepted 13 Mar 2024
Digital Object Identifier: 10.32913/mic-ict-research.v2024.n2.1247

Abstract: The implementation of Software-Defined Network (SDN) across multiple data centers aims to simplify the control and management of networks. However, the increasing popularity of SDN has also attracted the attention of attackers. To tackle this problem, it is essential to have an intrusion detection system (IDS) in place, which plays a crucial role in cybersecurity by addressing external threats. The advantage of SDN's centralized nature is that it facilitates the training of an IDS based on machine learning. However, there is a scarcity of research specifically focused on intrusion detection in SDN. Existing literature often treats SDN intrusion detection as similar to traditional computer systems and relies on intrusion datasets generated for those systems. We explore the issue of intrusion detection in SDN using the most recent public dataset (InSDN). However, InSDN is an imbalanced data set. In this paper, we have recommended a method to balance the data as well as a method to find the best features to improve the quality of IDS using Machine Learning. In addition, we also suggest a method of classifying SDN network traffic and normal network traffic. At the same time, we also evaluate the efficiency of the SDN system with the load balancing system and without the load balancing system.

Keywords: *Software-Defined Network, Intrusion Detection System, Machine Learning, Data Augmentation, Load Balancing, Network Flow Feature, SDN Detection, Feature Fusion.*

I. Introduction

1. Overview

The demand for network usage is continually growing in complexity, driven by advancements in both hardware and software technologies. Operating and managing the network manually shows certain disadvantages when errors occur, and flexibility is not high. Therefore, modern networks are moving towards automation as well as softwareization using advanced technologies such as SDN. SDN is a network architecture that allows centralized control over the network by separating the control plane from the data plane. There, the control plane will do things like traffic routing, congestion control, and so on.

Instead, the data plane just forwards the network traffic based on the configurations provided by the control plane. The control plane has the ability to control and view the network centrally. Therefore, it can make smart decisions to manage and operate the network securely. The advantage of SDN is indisputable, and it is increasingly widely used in data centers and enterprises. It is estimated that by 2027, the SDN market could reach 52 billion USD [1].

With its numerous advantages, SDN is gaining popularity and is increasing widely in network management. However, along with its increasing adoption, SDN has also drawn the attention of attackers. In SDN, the control plane holds centralized control and management over the network. If the control plane is compromised by an attacker, they can potentially gain control over the entire network, inadvertently turning SDN into a target for malicious attacks. This highlights the importance of implementing robust security measures to protect the control plane and mitigate the risk of such attacks in SDN environments.

Infiltrating the SDN server grants attackers unauthorized access, allowing them to acquire crucial system information. Their primary objective is to target the SDN controller, recognized as the brain of the network. Attackers may execute various malicious activities, including Denial-of-Service (DoS) attacks, Distributed Denial-of-Service (DDoS) attacks, Brute Force attacks, and other methods to compromise the controller's functionality and disrupt network operations. Protecting the SDN controller from such attacks is crucial to maintaining the security and

integrity of the entire SDN infrastructure.

It is extremely important to detect such attacks quickly so that prevention methods can be put in place. Normally, we would be able to think of IDS [2] as a common way to detect such attacks. More importantly, current IDS mostly use rules to detect attacks that have certain weaknesses, while IDS rely on network traffic analysis and packet characteristics to detect intrusions. Supposedly, such network traffic does not fall under the rules that the network administrator sets for a traditional IDS, but with an IDS using packet analysis, they can still be detected. This improves the quality of detecting malicious packets and attacks on network systems, helping network administrators quickly prevent attacks.

The utilization of Machine Learning (ML) for analyzing network packets and constructing IDS has become a well-established approach. ML has demonstrated notable advantages in terms of accuracy compared to traditional IDS methods. However, prior research has often overlooked the detection of specific attack details. Detecting these details can enable administrators to take targeted measures to prevent attacks effectively.

Furthermore, a prevalent issue in the ML-based IDS domain is the usage of imbalanced datasets for training ML algorithms. These datasets often exhibit a significant disparity in the frequency of different labels, which can result in learning algorithms becoming biased toward the more prevalent label during training. Consequently, when these algorithms are used to predict network flows, they may produce errors due to their skewed learning experience. Addressing this data imbalance challenge is crucial for developing reliable and effective ML-based IDS systems. Techniques such as data balancing methods, including over-sampling and under-sampling, can help mitigate this issue and improve the performance and accuracy of the ML algorithms for intrusion detection.

In addition, it is also important to detect what is SDN traffic and what is traditional network traffic. This helps attackers determine what their targets are and how they attack. However, there has not been much research on how to detect traditional network traffic and SDN network traffic. The use of modern technologies such as ML, DL is almost nonexistent. Therefore, we also focus on proposing a method that can classify whether the traffic belongs to the SDN network traffic or not.

2. Our contributions

In this paper, we focus on developing an IDS for SDN to detect and predict attacks and what types of attacks they are. In addition, we also develop a method for detecting SDN networks.

Our contributions can be explained as follows:

- Create a balanced dataset: The collected dataset obtained from real-world scenarios often exhibits a significant data imbalance, with the majority of data records representing instances when no attack is present. This imbalance can greatly skew the learning models used for IDS in SDN. The success of an IDS system in SDN heavily relies on the availability of a well-balanced dataset for training the models. Consequently, our first contribution in this paper is focused on the creation of a balanced dataset specifically tailored for SDN networks.

- Identifying a set of optimal features: Our second contribution is to evaluate the features from the balanced dataset in order to efficiently feed to our proposed algorithm. In general, using too many features leads to long training times and large file sizes that make it difficult to implement in practice. Therefore, we propose a method to find the feature set with the best classification ability while the model still ensures accuracy.

- Discover the best method: The third contribution is to conduct experiments and to find the most suitable method as well as an algorithm for detecting and also classifying data of attacks. Our suggested algorithms can improve IDS performance.

- Find a suitable load balancing algorithm: Lastly, we evaluate the performance of web servers in SDN without load balancing and with load balancing.

3. Roadmap

The rest of this paper is arranged as follows. Section II discusses the related work and theory analysis. In section III, the proposed method algorithm is described in detail. The experimental results and comparisons are shown in section IV. Finally, a short conclusion is drawn in section V.

II. Related Work

Intrusion detection is a difficult problem [3], which has been attempted by a lot of previous researchers to solve particularly in SDN. Most recent research works focused on applying ML [4–6] to build IDS for SDN. Deep Learning (DL) algorithms are also used in some research but not much. In this section, we discuss some of the previous studies concerned IDS for SDN.

In the article [7], the authors have built an IDS system using ML algorithms on the available data set. The results show that using ML to build IDS gives relatively good results, reducing dependence on leading experts and replacing the need for a large amount of data to train models. However, the method proposed by the authors has many

limitations, not overcoming the inherent weaknesses of the data set as well as improving the efficiency of IDS attack detection.

In more detail, IDS using ML in SDN has made great contributions to generating a relatively large and rich dataset of attacks in SDN [8]. This is the foundation for subsequent researchers to develop highly transparent ML-using IDSs for SDN systems. The authors also evaluated the generated dataset with basic ML algorithms and achieved very good results with specific detection of each type of attack. However, the authors have not solved the imbalance problem and are using a model to detect attacks in binary form, which has not been done at the multi-class level. In order to detect and defense the specific network attack such as flooding DDoS attack for SDN, Song Wang *et al*. [9] had employed lightweight supervised learning techniques. However, their experimental dataset is not belong to real SDN. Tuan *et al*. [10] proposed to compute entropy and logarithmic values to detect TCP-SYN and ICMP flood attacks in SDN, respectively. KNNs (K-Nearest Neighbor) are recruited to find the K nearest Euclidean distance points from the current entropy or logarithmic point to determine if the network is subject to a DDoS attack. This model is evaluated with CAIDA 2007 dataset and self-generated traffic using Bonesi, accuracy is over 99% when $K = 9$.

In another paper[11], the authors propose a method to detect early attacks in SDN networks. Their method took a set of 9 features and then transform them through CNN to obtain a 128-dimensional feature vector. Then let that feature vector go through algorithms like XGBoost and Random Forest (RF) for training. The solution method is relatively good and practical, and the training time is relatively low. However, they can only do it at the level of detecting attacks and not being attacked, not solving the problem of what type of attack. There are also some studies [12] on applying ML and Deep Learning (DL) to IDS problems of SDN. They summarized in detail the methods that have been applied and the technologies applied are relatively new. However, the authors use data sets that are not large enough, and the data imbalance is not resolved.

In the article "Network intrusion detection in software defined networking with self-organized constraint-based intelligent learning framework" [13], the authors have also outlined a number of ways to use Machine Learning to detect network intrusions, but the problem they focus on solving is to reduce the error when the model predicts as well as to avoid problems such as over-fitting. Recently, the application of AI technology to systems is gradually gaining popularity. There are many research works related to network intrusion detection using AI. A group of authors conducted a synthesis of methods to apply AI network intrusion detection in SDN [14]. However, the article has not yet provided solutions for the weaknesses of the existing works.

In the paper "Survey on SDN based network intrusion detection system using machine learning approaches" [15], the authors have synthesized methods of network intrusion detection using ML. The article mentioned tools that can develop NIDS (Network Based Intrusion Detection System) in a real environment. Conversely, the methods they came up with have not yet solved the problem of data imbalance. In another way, applying deep learning for network intrusion detection is also a very positive and potential direction. There are many groups of authors who have applied different deep learning methods to realize that. In the paper "A Novel Approach To Network Intrusion Detection System Using Deep Learning For Sdn: Futuristic Approach" [16], the authors used 12 features out of 41 features in the NSL-KDD dataset[1] by using the feature selection method. It is also considered to choose the effective feature of real network traffic. The results of the paper are relatively good and capable of applying it to practical implementation. However, the paper uses a general data set for the IDS system and it has not been specialized for SDN application.

In the paper "Early Detection of Intrusion in SDN" [17], Towhid *et. al.* focused on detecting network intrusions as early as possible. Using real-time stream-based traffic network features, they built a method for early detection of network intrusions. They used the InSDN dataset [8], then employed the 9 features outlined in the paper "A novel hybrid model for intrusion detection systems in SDNs based on CNN and a new regularization technique" [18]. However, their method also reduced the effectiveness of intrusion detection, not solving the problem of data imbalance. They also have not specifically detected each type of network attack.

According to previous studies, we found that there are several problems that are not still solved for SDN environment. Those are data imbalance and type of network attack detection for SDN. Our method proposes a method to balance the data and detect the details of each type of attack for IDS of SDN applications.

III. Our Proposed Method

In traditional networks, IDSs are commonly positioned behind firewalls. These IDSs typically receive data from the SPAN port of the switch, enabling them to analyze network data and make predictions. In the case of software-defined networking (SDN), all network traffic flows through the

[1] https://www.unb.ca/cic/datasets/nsl.html

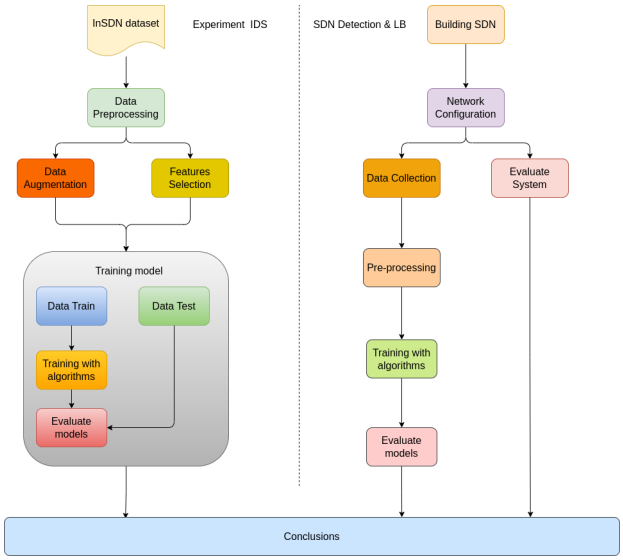

Figure 1. Pipeline for experiments of this paper

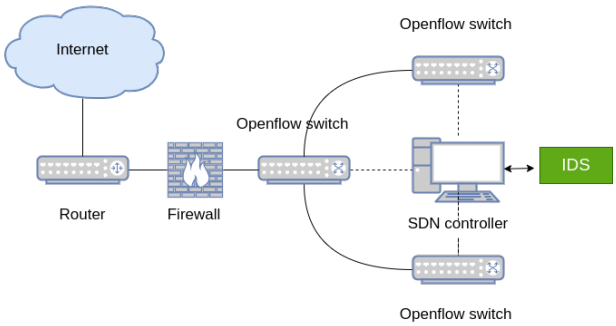

Figure 2. IDS location in SDN

SDN controller. Therefore, it is recommended to place the IDS behind the firewall and retrieve network traffic from the SDN controller. Our proposed IDS setup is illustrated in Fig. 2. Following the training of the model, the Machine Learning (ML) algorithm identified the model with the most effective classification capabilities. This model is then utilized for network intrusion detection in SDN. The network traffic originates from the client and passes through the SDN controller before reaching the server. During this process, the network traffic is directed through the IDS by the SDN controller. If the IDS detects any abnormal network traffic behavior and classifies it as intrusive, an alarm is triggered.

In this section, we present the issues that have been resolved in the article. Firstly, data imbalance is a big problem. Therefore, we have studied and proposed different methods to solve the data imbalance problem. Secondly, we propose a method to find the set of data imbalances features with the best classifier. This can improve the network intrusion detection classification ability for IDS. In addition, we also focus on reducing the number of features used in order to reduce the training time and weight of the model after training. This is the basis for being able to deploy IDS on actual devices. We perform this test on the dataset InSDN [7]. Thirdly, we propose an algorithm to classify traditional network traffic and that of SDN. To the best of our knowledge, this is the first method to classify the network traffic of traditional network (physical-based network) and that of SDN (software-based network). After distinguishing the type of network traffic from physical-based network or from software-based network (SDN), we can build different attacks and defense solutions for the SDN network. Finally, we evaluate the SDN system with and without load balancing for real applications.

In the second experiment, we build an SDN network with Mininet[2] and Ryu controller[3]. Conduct network traffic data collection of traditional networks and SDN networks. After collecting the data from the traditional network and the SDN network, we build a classifier of these two networks. Finally, based on the built-in network model, we evaluate the SDN system with load balancing and without load balancing in order to evaluate the its performance in real applications.

1. Our Approach

Data imbalance poses a significant challenge in Machine Learning (ML) [19] as it can adversely affect the learning process of algorithms. When the data is imbalanced, the algorithm tends to assign more weight to the labels that appear more frequently. To address this issue, various techniques such as down-sampling and over-sampling can be employed. However, down-sampling the data significantly reduces its size, then leads potentially resulting in the loss of important training data for the trained model.

In our paper, we present a novel approach for addressing data imbalance through the use of over-sampling, specifically employing the Synthetic Minority Oversampling Technique (SMOTE) [20]. The SMOTE technique, introduced by Nitesh Chawla *et al*. [21], operates by identifying instances that are close to each other in the feature space. It establishes a connection between these instances and generates synthetic samples along that connection, effectively increasing the representation of the minority class in the dataset [22].

Once we successfully addressed the issue of data imbalance, our focus shifted toward developing a method to identify features that serve as effective classifiers for our IDS. By identifying such features, we were able to

[2]http://mininet.org/
[3]https://ryu-sdn.org/

reduce the number of features required to construct the IDS. This reduction not only decreased the training time but also reduced the overall weight of the model. The resulting streamlined IDS was more efficient and effective in detecting network intrusions.

In this paper, we recognized the significance of distinguishing between SDN network traffic and traditional network traffic. To address this, we collected relevant data and constructed models to classify network traffic into these two categories. By developing these models, we aimed to accurately differentiate between SDN-specific traffic and traffic associated with traditional networks. This classification process provided valuable insights and contributed to a better understanding of network traffic patterns in the context of SDN.

Our project involved the design and implementation of a basic Software-Defined Networking (SDN) infrastructure using Mininet and Ryu controllers. The purpose of this setup was to assess the performance of the SDN system under two conditions: with load balancing and without load balancing. By comparing the outcomes of these two scenarios, we aimed to evaluate the impact of load balancing on the overall performance of the SDN system.

2. Data Pre-processing

In our data preprocessing phase, we performed cleaning operations on the existing dataset. Specifically, we removed any records that contained missing values represented as "NaN" fields. Subsequently, we addressed the data imbalance problem by applying techniques such as over-sampling and down-sampling. However, during our experimentation, we observed that down-sampling led to the loss of crucial data points, and therefore, we decided not to proceed with this method.

After addressing the data imbalance, we proceeded to split the dataset into three sets: training, testing, and validation. The split was performed at a ratio of 80% for training, 10% for testing, and 10% for validation. This division allowed us to have separate sets for training the model, evaluating its performance, and validating its generalization capabilities.

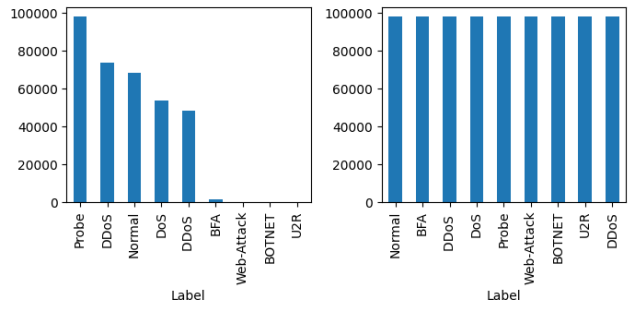

Figure 3. Before and After Using Data Augmentation

3. Data Augmentation

In our study, we utilize the Synthetic Minority Oversampling Technique (SMOTE) [22] to enhance the existing dataset. By employing SMOTE algorithm, our objective is twofold: to augment the data and to ensure a balanced representation across different IoT devices. To achieve this purpose, we leverage the largest number of device samples present in the dataset and apply SMOTE to generate an equivalent number of samples for the remaining devices [23]. This approach allows us to address the data imbalance issue and create a more representative and balanced dataset for further analysis and modeling.

Our approach follows a two-step process. First, we utilize the Imblearn [24] library, specifically leveraging the oversampling technique, to augment the dataset. This step involves generating synthetic samples to balance the class distribution in the dataset. Next, we apply this data augmentation method to our specific dataset, resulting in outcomes comparable to those shown in Fig.3. This approach allows us to effectively address the data imbalance and create a more balanced dataset for further analysis and modeling.

4. Features Selection

In fact, the collected IoT data includes many different features in which there are many features of less importance and features with low classification ability. Using all the features can cause interference with the model during the learning process, increase model weight after training, and training time.

In order to select the efficient features, we propose an algorithm, which allows us to find features with high classification ability for many ML algorithms. The input of this algorithm is all the features of the data set, the output is the features with the best classification ability.

In this problem, we use the function available in the libraries as feature "_importances_". This function is created by the library based on the mathematical formula of the authors Trevor Hastie, Robert Tibshirani, and Jerome Friedman [25]. They define the calculation of feature importance using the following equation:

$$I_l^2(T) = \sum_{t=1}^{J-1} i_t^2 \prod [v(t) = l], \tag{1}$$

where T is the whole decision tree, l feature in question, J number of internal nodes in the decision tree, i squared the reduction in the metric used for splitting, $\prod$ is indicator

Algorithm 1 Find the best features

Input: List of all features
Output: List of best features

```
Initialize Algorithms: models = {RandomForest, Xgboost, LightGBM}
Initialize important feature of models: iF = [ ]
Initialize list of best features: bestFeature = [ ]
for each in models do
    Model training with all features
    Append to iF
end for
for i = 0 to 50 do
    for j = 0 to 50 do
        for k = 0 to 50 do
            if iF['RF'][i]=iF['Xgb'][j]=iF['LGBM'][k] then
                Append feature to bestFeature
            end if
        end for
    end for
end for
Return bestFeature
```

function, $v(t)$ is a feature used in splitting of the node t used in splitting of the node. This math formula will sum up all the decreases in the metric for all the features across the tree (Random Forest, Xgboost, and LightGBM algorithms are all based on Decision Tree algorithm).

We train algorithms like RandomForest, XGBoost, and LightGBM with all features. We then superimpose the significant features of each algorithm. Within the 50 most important features of each algorithm, if it is also the important feature of the rest of the algorithms, we add the most important feature.

Once we have found those important features, we'll retrain them with basic Machine Learning algorithms to evaluate accuracy of IoT devices classification.

Based on Formula (1), we utilize the "_importances_" function in the libraries to conduct experiments as described earlier. As a result, we got 12 features, as shown in Table I.

5. Data Collection and Pre-processing

To be able to distinguish traditional network traffic and SDN network traffic. We collect network traffic data of traditional networks and SDN networks. We use Wireshark to capture packets exchanged in the experimental network.

To do this, we built an experimental system. We use docker to create a traditional network. This network consists of 10 clients and 3 servers. Similarly, in SDN we use Mininet and Ryu controller to build an equivalent network of 10 clients and 3 servers.

Then, we collect the data in turn of the networks. We let the devices in the traditional network communicate with each other and used Wireshark[4] to capture the packets it exchanged and label them as "0". Next, we let the devices in the SDN communicate with each other and also use Wireshark to capture those packets and label them as "1".

In order to be able to use this data to classify traditional networks and SDN networks. We need to clean the data first. The collected data will include a lot of "NaN" data fields. We need to get rid of them. For example, the problem we are solving is based on IPv4 data fields, so the information in the network related to IPv6 will be corrupted. We need to remove the records with IPv6 data fields away.

In general, most of our experiments are built on a relatively small network model. The allocation of IP addresses to devices in network models is relatively limited. When the number of IP addresses allocated in the network is relatively small, the IP address data collection that exists in the training dataset is also relatively small. This causes the model during training to be biased. It will default when it sees IP address A as the traditional network. Otherwise, it will default when it sees IP address B as SDN. To solve this problem, we can do it in two ways. We can build a large enough network with a large enough number of source and destination IP addresses to avoid this problem. The above is only suitable for large systems, we do not recommend it for small experimental systems. Another simpler way, we can remove the source IP and destination IP address features to train the model. Based on our preprocess, the bias of small dataset can be avoided.

IV. Results and Comparison

1. Experimental environment

Our experimental environment is on a PC with the following configuration: Intel i7-10700f CPU, 32 GB RAM, and 16GB VRAM RTX A4000 VGA card. We choose Python language and jupyter notebook platform to perform the experiments. When we train the model, we use Python version 3.6.13, sci-kit learn 0.24.2, TensorFlow 2.6.2, Numpy 1.19.2, pandas 1.1.5, and some other tools to analyze and train the model in the experiment.

In this paper, we use some popular Machine learning algorithms for testing such as Random Forest (RF), XGBoost (XGB), LightGBM (LGBM), and some others. They require the use of relatively different libraries. RF belongs

[4]https://www.wireshark.org/

to sci-kit learn library, XGB belongs to XGBoost library[5] and LGBM belongs to Lightgbm library[6].

2. Dataset preparation

In this section, we provide a description of the dataset utilized for training our Machine Learning algorithms. We employed the InSDN dataset [8], which was created in 2020. This dataset is notably extensive and encompasses a wide range of SDN network data. It incorporates data related to diverse types of attacks, including DoS, DDoS, Web-attacks, and numerous other attack types. The dataset contains a total of 343889 distinct records of network traffic in SDN. It comprises 84 columns, with 83 columns representing various network features, and 1 column indicating the label. Among the different types of attacks, Probe attacks make up the majority with 98129 records, while U2R attackers have the smallest representation, comprising only 17 records.

It is evident that the dataset is significantly imbalanced, with a substantial difference in the number of records between different labels. To address this issue, we employ the SMOTE method with over-sampling. By utilizing SMOTE, we successfully generate synthetic samples and achieve a balanced dataset, as depicted in Fig. 3. That is the first training dataset in our experiment.

In the second training dataset, we use the selected features. To select these features, we used all the features of the dataset to train with machine learning algorithms such as Random Forest, XGBoost, and LightGBM. After training, we obtain the order of important features for each algorithm. Within the 50 most important features of each algorithm, we superimpose each other to obtain the most important features. This method has been described in the section III.4.

We then obtain a list of features that fall within the 50 most important features of each algorithm. This list includes 12 features which is shown in Table I.

3. Result of ML application in intrusion detection

In this section, we do experiment with two training datasets. These two data sets are the result of our two different experimental directions. Both datasets we created for the experiment are based on the original data set, InSDN. For the first experiment, we used an incremental dataset. In the second experiment, we use the dataset after the important features have been extracted based on our proposed method.

[5]https://xgboost.readthedocs.io/
[6]https://lightgbm.readthedocs.io/

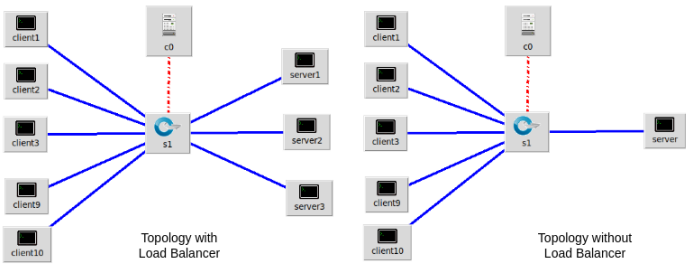

Figure 4. Topology of SDN

TABLE I
LIST OF IMPORTANT FEATURES

Ft/Algo	XGBoost	LightGBM	RandomForest
Bwd Pkts/s	**0.179**	425	0.039
Dst Port	0.073	987	0.052
Bwd Header Len	0.067	110	0.015
Timestamp	0.046	**1753**	**0.093**
Src IP	0.034	1470	0.048
Init Bwd Win Byts	0.027	351	0.061
Flow ID	0.014	1715	0.038
Dst IP	0.006	564	0.031
Src Port	0.005	892	0.035
Flow IAT Min	0.001	394	0.025
Fwd Pkts/s	0.0006	302	0.027
Flow Duration	0.0002	822	0.029

We tested in turn with algorithms such as Decision Tree, Random Forest, XGBoost, Logistic Regression, and LightGBM. First, we do experiment with the original dataset. Then, we implement experiment with the balanced and enriched dataset. Finally, we evaluate with the dataset after having extracted the important features.

Table II presents a comparison of the results before and after applying the data balancing method. The initial dataset was unbalanced, leading to low values for evaluation metrics such as F1-score. Furthermore, there were noticeable differences between the performance indexes. However, after applying the data balancing method, significant improvements are observed in our experimental results. The F1-score shows substantial enhancement. This indicates that balancing the data has positively impacted the performance of the model, resulting in improved accuracy and effectiveness in the classification task.

The algorithm for the highest accuracy is Random Forest with accuracy up to 99.95%. However, the XGBoost algorithm has the largest Recall and F1-scores at 99.90% and 99.86%, respectively. This index reflects the learning level of the model with the unbalanced dataset. With the current stats showing, the XGBoost algorithm with augmented dataset (XGBoost + Aug) is superior compared with another ML methods. The details of experimental results are shown in Table II.

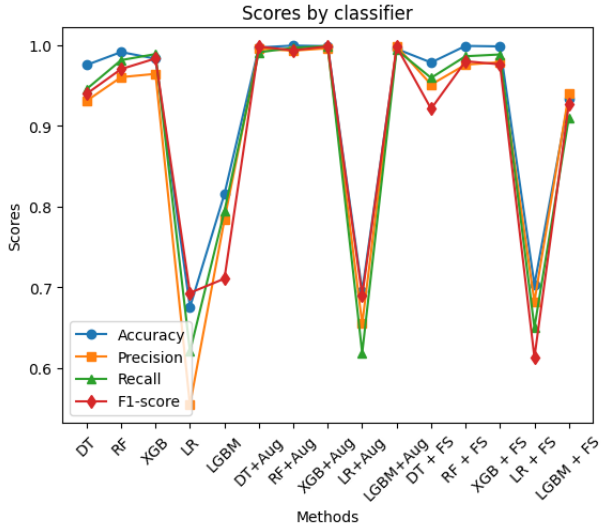

Figure 5. Scores of The Methods

TABLE II
RESULTS USING THE METHODS

Method	Accuracy	Precision	Recall	F1-score
Decision Tree	0.9755	0.9311	0.9457	0.9398
Random Forest	0.9915	0.9604	0.9815	0.9704
XGBoost	0.9833	0.9645	0.9890	0.9835
Logistic Regression	0.6755	0.5545	0.6208	0.6932
LightGBM	0.8156	0.7839	0.7943	0.7108
Decision Tree + Aug	0.9972	0.9955	0.9912	0.9983
Random Forest + Aug	**0.9995**	0.9932	0.9967	0.9936
XGBoost + Aug	0.9988	0.9962	**0.9990**	**0.9986**
Logistic Regression + Aug	0.6955	0.6549	0.6188	0.6898
LightGBM + Aug	0.9954	**0.9987**	0.9943	0.9981
Decision Tree + FS	0.9785	0.9512	0.9595	0.9212
Random Forest + FS	0.9991	0.9759	0.9863	0.9801
XGBoost + FS	0.9985	0.9789	0.9885	0.9763
Logistic Regression + FS	0.7032	0.6821	0.6501	0.6132
LightGBM + FS	0.9338	0.9402	0.9101	0.9271
InSDN[8] + Random Forest	-	0.9942	0.9969	0.9955
NIDS-DL[16] + CNN	0.9863	0.9845	0.9898	0.9872

As shown in Table II, the accuracy of algorithms such as Random Forest, and CNN on datasets like InSDN[8] and NIDS-DL[16] is relatively high. However, algorithms such as Random Forest, XGBoost, and LightGBM improved performance when combined with augmented datasets.

In addition, our Features Selection (FS) method gives less accurate results than the data enrichment method. However, after applying the FS method, the model training time is significantly reduced and the model weight is also reduced relatively. It implies that our FS method is effective for feature selection from many less efficient features.

TABLE III
TRAINING TIME AND FILE SIZE WITH FS

Method	Accuracy	Training time	File size
Random Forest	99.15	1m16s	4.9mb
Xgboost	98.33	5m59s	821.3kb
LightGBM	81.56	35.5s	2.1mb
Random Forest + FS	**99.91**	48.8s	2.3mb
Xgboost + FS	99.85	1m14s	**815.1kb**
LightGBM + FS	93.38	**15.7s**	2.0mb

TABLE IV
RESULTS USING ML FOR SDN DETECTION

Method	Accuracy	Precision	Recall	F1-score
Decision Tree	0.9935	0.9201	0.9655	0.9638
Random Forest	0.9951	0.9356	**0.9832**	0.9614
XGBoost	**0.9973**	**0.9745**	0.9830	**0.9795**
Logistic Regression	0.9765	0.9641	0.9608	0.9432
LightGBM	0.9906	0.9539	0.9747	0.9632

Table III shows that after applying FS method, the training time and file size of the model after training are greatly reduced. There is a slight increase in the accuracy of the model compared to using the original data set. The file size of the Random forest algorithm is halved.

4. SDN Detection

In this section, we explain the results of traditional network and SDN detection based on network traffic. With the dataset collected in both types of networks, we proceed to process, label and train the model.

The dataset we collected includes 91489 rows and 35 columns. After preprocessing the data, we removed 2 columns, so the training data remained 33 columns. The two columns of data that we leave out are ip.src and ip.dst. As we explained above, because our dataset is small, it is necessary to remove the source and destination IP addresses to avoid the bias of experimental results.

With the collected data set as described above, we train the model to detect SDN. The algorithms we tested cover the basics of Machine Learning. Our experiment works on algorithms like Decision Tree, Random Forest, and XGBoost. With our dataset, when testing with the above basic algorithms, the results are significantly improved.

The dataset we collected has not yet resolved the imbalance problem, so the results still have a difference between F1-score and Accuracy. Table IV shows that when using ML, we can be completely possible to detect SDN. The above results are also very positive and have potential for real development.

TABLE V
SYSTEM RESPONSE TIME

Method	max	avg	min
Non-LB	23.8ms	13.2ms	0.5ms
LB-RR	**13.8ms**	**7.2ms**	**0.4 ms**
LB-WRR	15.3ms	7.5ms	0.51ms

5. Evaluation of SDN system with load balancing

In this section, we have successfully constructed a basic SDN network using the Mininet network emulator and the Ryu controller. Building a simple SDN network is straightforward with these tools. For our setup, we opted for the simplest network topology, which is a star topology. The network architecture includes 10 clients, 3 web servers, 1 switch, and 1 controller. This configuration allows for the evaluation and testing of SDN functionalities and applications within a controlled environment.

Setup three web servers to open port 80 providing HTTP services. With the web server installed as Apache. Program the Ryu controller to test all 3 cases: Non-Load Balancing, Load Balancing with Round Robin, and Load Balancing with Weight Round Robin. With Weight Round Robin we set the weights for servers 1,2,3 to be 1,3,5, respectively.

In order to evaluate the response time of the system, we conducted experiments using three different network configurations. The network topology was set up as depicted in Fig. 4, utilizing Mininet and the Ryu controller. For the load balancing experiment, we programmed the Ryu controller to manage the traffic distribution among the servers. In the Round Robin algorithm case, equal weights of 1 were assigned to all servers. In the Weight Round Robin scenario, the weights were set as 1, 3, and 5 for servers 1, 2, and 3 respectively. Through these experiments, we measured and analyzed the response time of the system under different load balancing strategies.

We can see the response time is extremely small. The response time of the system with load balancing is comparatively much better than the system without load balancing. In particular, the Round Robin algorithm proved to be superior when there were significantly better coefficients. The fastest response time was 0.4 ms and the longest was 23.8 ms.

V. Conclusions

SDN is a relatively new and highly applicable field. However, it also has many potential security problems when centralizing control in the controller. The study of network intrusion detection methods is extremely necessary for the system.

In this paper, we have proposed a method to build IDS based on ML using network traffic. Our results are relatively good and can be deployed into real systems. We also propose methods to solve the problem of data imbalance. In addition, we also found the features with the best classification. The detection of SDN traffic has also been demonstrated by us. Finally, we also evaluate the performance of the SDN network with the load balancing system and without the load balancing system.

In the future, we will continue to develop our research. We will study and apply Deep Learning models to the problem. Test the actual implementation of the model on a given company.

References

[1] J. A. Sava, "Software-defined networking (sdn) market size worldwide from 2021 to 2027," Available at https://www.statista.com/statistics/468636/global-sdn-market-size/ (2023/25/5).

[2] A. Chawla, B. Lee, S. Fallon, and P. Jacob, "Host based intrusion detection system with combined cnn/rnn model," in *ECML PKDD 2018 Workshops*, C. Alzate, A. Monreale, H. Assem, A. Bifet, T. S. Buda, B. Caglayan, B. Drury, E. García-Martín, R. Gavaldà, I. Koprinska, S. Kramer, N. Lavesson, M. Madden, I. Molloy, M.-I. Nicolae, and M. Sinn, Eds. Cham: Springer International Publishing, 2019, pp. 149–158.

[3] Y. Al-Nashif, A. A. Kumar, S. Hariri, Y. Luo, F. Szidarovsky, and G. Qu, "Multi-level intrusion detection system (ml-ids)," in *2008 International Conference on Autonomic Computing*, 2008, pp. 131–140.

[4] S. S. Gopalan, D. Ravikumar, D. Linekar, A. Raza, and M. Hasib, "Balancing approaches towards ml for ids: A survey for the cse-cic ids dataset," in *2020 International Conference on Communications, Signal Processing, and their Applications (ICCSPA)*, 2021, pp. 1–6.

[5] A. S. Dina and D. Manivannan, "Intrusion detection based on machine learning techniques in computer networks," *Internet of Things*, vol. 16, p. 100462, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660521001037

[6] P. Parkar and A. Bilimoria, "A survey on cyber security ids using ml methods," in *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, 2021, pp. 352–360.

[7] M. Almgren and E. Jonsson, "Using active learning in intrusion detection," in *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.*, 2004, pp. 88–98.

[8] M. S. Elsayed, N.-A. Le-Khac, and A. D. Jurcut, "Insdn: A novel sdn intrusion dataset," *IEEE Access*, vol. 8, pp. 165 263–165 284, 2020.

[9] S. Wang, J. F. Balarezo, K. G. Chavez, A. Al-Hourani, S. Kandeepan, M. R. Asghar, and G. Russello, "Detecting flooding ddos attacks in software defined networks using supervised learning techniques," *Engineering Science and Technology, an International Journal*, vol. 35, p. 101176, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2215098622000842

[10] N. N. Tuan, P. H. Hung, N. D. Nghia, N. V. Tho, T. V. Phan, and N. H. Thanh, "A ddos attack mitigation scheme in isp networks using machine learning based on sdn," *Electronics*, vol. 9, no. 3, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/3/413

[11] N. S. Shamim Towhid, "Early detection of intrusion in sdn," Available at https://uregina.ca/~nss373/papers/Early-detection-SDN.pdf (2023/25/5).
[12] M. R. Ahmed, salekul Islam, S. Shatabda, A. K. M. M. Islam, and M. T. I. Robin, "Intrusion Detection System in Software-Defined Networks Using Machine Learning and Deep Learning Techniques –A Comprehensive Survey," 12 2021. [Online]. Available: https://www.techrxiv.org/articles/preprint/Intrusion_Detection_System_in_Software-Defined_Networks_Using_Machine_Learning_and_Deep_Learning_Techniques_A_Comprehensive_Survey/17153213
[13] A. Bhardwaj, R. Tyagi, N. Sharma, A. Khare, M. S. Punia, and V. K. Garg, "Network intrusion detection in software defined networking with self-organized constraint-based intelligent learning framework," *Measurement: Sensors*, vol. 24, p. 100580, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2665917422002148
[14] S. Dahiya, V. Siwach, and H. Sehrawat, "Review of ai techniques in development of network intrusion detection system in sdn framework," in *2021 International Conference on Computational Performance Evaluation (ComPE)*, 2021, pp. 168–174.
[15] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad, "Survey on sdn based network intrusion detection system using machine learning approaches," *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 493–501, Mar 2019. [Online]. Available: https://doi.org/10.1007/s12083-017-0630-0
[16] M. R. Hadi and A. S. Mohammed, "A novel approach to network intrusion detection system using deep learning for SDN: Futuristic approach," in *Machine Learning &amp Applications*. Academy and Industry Research Collaboration Center (AIRCC), jun 2022. [Online]. Available: https://doi.org/10.5121%2Fcsit.2022.121106
[17] M. S. Towhid and N. Shahriar, "Early detection of intrusion in sdn," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023, pp. 1–6.
[18] M. S. ElSayed, N.-A. Le-Khac, M. A. Albahar, and A. 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*, vol. 191, p. 103160, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804521001739
[19] S. Ray, "A quick review of machine learning algorithms," in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019, pp. 35–39.
[20] P. Jeatrakul, K. W. Wong, and C. C. Fung, "Classification of imbalanced data by combining the complementary neural network and smote algorithm," in *Neural Information Processing. Models and Applications*, K. W. Wong, B. S. U. Mendis, and A. Bouzerdoum, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 152–159.
[21] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *J. Artif. Intell. Res. (JAIR)*, vol. 16, pp. 321–357, 06 2002.
[22] G. Douzas and F. Bacao, "Geometric smote a geometrically enhanced drop-in replacement for smote," *Information Sciences*, vol. 501, pp. 118–135, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025519305353
[23] Z. Wang, J. Hu, G. Min, Z. Zhao, and J. Wang, "Data-augmentation-based cellular traffic prediction in edge-computing-enabled smart city," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 6, pp. 4179–4187, 2021.
[24] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365
[25] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference and prediction*, 2nd ed. Springer, 2009. [Online]. Available: http://www-stat.stanford.edu/~tibs/ElemStatLearn/

Van Dan Duong graduated with a degree in Information Technology from Le Quy Don Technical University in Vietnam in 2023. His research interests lie in the areas of deep learning, steganography, and computer vision.
Email: duongvandan2806@gmail.com

Nam Khanh Tran obtained an impressive IT engineering degree from Le Quy Don Technical University in Vietnam in 2020. He has since been contributing to the academic community as a lecturer at the same university, specializing in the research fields of cybersecurity, technology network, and image processing.
Email: khanhtn107195@lqdtu.edu.vn

Minh Thanh Ta is currently an Associate Professor and Vice Dean of Institute of Information and Communication Technology in Le Quy Don Technical University, Vietnam. He is also a Postdoctoral Fellow of the Department of Mathematical and Computing Sciences at Tokyo Institute of Technology. He received his B.S. and M.S in Computer Science from National Defense Academy, Japan, in 2005 and 2008 and his Ph.D. from Tokyo Institute of Technology, Japan, in 2015, respectively. He is a member of IPSJ Japan and IEEE. His research interests lie in the area of watermarking, network security, and computer vision.
Email: thanhtm@lqdtu.edu.vn