# A novel Machine Learning-based Network Intrusion Detection System for Software-Defined Network

1st Duc-Huy Le
*School of Information and Communication Technology*
*Hanoi University of Science and Technology*
Hanoi, Vietnam
huy.ld141938@sis.hust.edu.vn

2nd Hai-Anh Tran
*School of Information and Communication Technology*
*Hanoi University of Science and Technology*
Hanoi, Vietnam
anhth@soict.hust.edu.vn

***Abstract*—Network Intrusion Detection System (NIDS) is an important component in many network systems. The rapid development of the Internet requires NIDS to improve performance in terms of both accuracy and efficiency. In this paper, we propose a flow-based anomaly detection system in applying Machine Learning approach in a SDN network. The paper implements a testbed to achieve an eight-feature dataset as the input for training six Machine Learning models. The obtained experimental results showed that the proposed NIDS is potentially a good security solution for a SDN network.**

***Index Terms*—NIDS, SDN, Machine Learning**

## I. Introduction

Network Intrusion Detection System (NIDS) protects systems by monitoring network activities and detecting malicious attacks. Traditionally, there are two types of NIDS based on strategies to detect attacks. The first one, *signature-based NIDS*, matches the arrived network traffic with a knowledge based of threats, while the other one, *anomaly-based NIDS*, detects attacks according to derivation of the received network traffic with determined normal activities. The most used signature-based technique is rule-based detection, which uses a database of known intrusion patterns, so called detection rules. Any arrived packet is compared to every rule and will be marked as an attack if it matches at least one rule. This strategy has advantages including ease of operation and low error detection rate. However, with the demand of an efficient NIDS for large and fast-growing network systems, rule-based approaches also have some potential drawbacks. Firstly, the rising of cyber threats requires the NIDS be updated frequently. However, since the rules are installed explicitly, hackers can still slightly change the attack behavior to bypass the system. As a consequence, the security system needs sufficient rules to cover all possible circumstances. Hence, the rules database will expand significantly. Secondly, the matching process will consume a lot of system resources (time and memory). With the increasing amount of networking traffic, this second drawback deteriorates the NIDS's ability for real-time operation. To solve these problems, the Machine Learning (ML)-based approaches are considered as a potential solution. ML has witnessed great development as a result of advancements in computer technology such as GPU and TPU. It offers an opportunity to apply ML to a NIDS, thus helps to solve all problems above. However, applying ML to the traditional networks is not feasible because all network's nodes have only local view and control capacity. That problem motivates us to adopt a centralized solution: Software-Defined Network (SDN). The core idea of SDN architecture is to separate the control plane from the data plane of all network nodes. Then, the network control is centralized, making it directly programmable. Therefore, network management can now be performed by a centralized software controller with a global view of the whole system. This technology enables dynamic, flexible, programmatically efficient network configuration. The advantages of SDN has been proven by various scenarios and backbone systems in both academic and industry, making SDN being developed by many hardware vendors, such as HP, Cisco, Dell, and Intel.

In this paper, we propose a ML-based NIDS in a SDN network. In particular, the paper used a Deep Learning (DL) approach in constructing three DL models including a basic NN models and two Convolutional Neural Network (CNN) models. The latter take our self-collected dataset in a SDN network as input for the training process. The paper also compared the accuracy and performance of the proposed CNN models with other traditional ML algorithms. The experimental works show that our proposal yielded a better result in terms of accuracy and running time.

The rest of the paper is structured as follows. In section II, the paper reviews some related works. Section III describes the proposed system. In section IV, the paper analyzes the performed experiments and the obtained results.

## II. Related Works

NIDS in SDN has been extensively developed recently. In [1], by analyzing characteristics of TCP 3-way handshake mechanism and attacks related (TCP Port Scan and TCP SYN Flood), the authors provide four prominent algorithms (threshold random walk with credit base limiting (TRW-CB algorithm), rate limiting, max entropy and NETAD) and implement those techniques in SDN environment with NOX Controller. The common idea of the algorithms is to create a variable related to some characteristics of the attacks and set a threshold for its value. New data with its information exceed the threshold would be considered as an intrusion. Experiments show that the algorithms work well in SOHO (Small Office/Home Office) network with high accuracy. However, this approach can only detect limited types of attack. It is hard to expand and it lacks flexibility because of the fixed threshold.

In a SDN architecture with OpenFlow, developers can easily control packet flows by setting the parameters of a switch's flow through programmable controlling application. In [2], the authors introduce a system which mirrors all the network packets to a separate computer working as a NIDS. Their system uses a well-known rule-based IDS: Snort [3]. For the similar technique of packet handling, the authors in [4] apply two machine learning models (Decision Tree and Support Vector Machine) trained with KDD-99 dataset [5] to classify network traffic and suggest an Intrusion Prevention System (IPS) over the attacks. Although above systems have taken advantages of programmable feature of SDN, but their NIDS is contradictory to the idea of centralizing functions as applications in SDN and still have the scalability problem.

As mentioned above, an OpenFlow switch not only provides the forwarding function over flow rules installed by the controller but also counters to collect some information of each individual flow, known as flow statistics. A SDN controller can also request those statistics through Flow Stats Request message of OpenFlow protocol. In taking advantage of this exclusive feature of SDN, Braga et al. [6] propose a system which applies ML in using Self Organizing Maps (SOM) to classify flows by their information and to detect DDoS attacks. They use a six-feature dataset extracted from KDD-99 dataset to train the model. Their method provided an accuracy of 99% for DDoS attacks. In [7], the authors implement a NIDS for 5G network using SDN architecture. They also use an extracted KDD-99 dataset with four features, the entropies of four parameters in a packet (source IP address, destination IP address, source port and destination port). KDD-99 is an old dataset gathered two decades ago and now it is clearly outdated. Another drawback of this dataset is the far different percentage of classes, which might affect the performance of the ML model.

## III. Proposed ML-based NIDS in SDN

The proposed NIDS is a combination of two approaches: *flow statistic* and *ML*. For *flow statistic-based approach*, the NIDS is not a middle box in traditional network but a service or a program running on the Application Layer of the SDN architecture. The program periodically sends flow statistic requests to OpenFlow switches and uses a trained machine learning model to classify flows, thereby detecting abnormal traffic. This approach allows the NIDS only work on the flows, not on all of network packets, which results in a superior ability of working real-time, especially in large scale networks. On the other hand, *ML-based* approach enables a dynamic, flexible network flow analyzing method. That facilitates the NIDS's expanding and updating process without affecting the system. In our system, we collect our own dataset with eight chosen features to train the models. That self-collected dataset allows us to update newer behavior of attacks and to have more realistic data considering to SDN network.

### A. Overview

As shown in Fig. 1, the proposed NIDS architecture consists of two main modules: **Training Module** and **Realtime Intrusion Detection Module**.

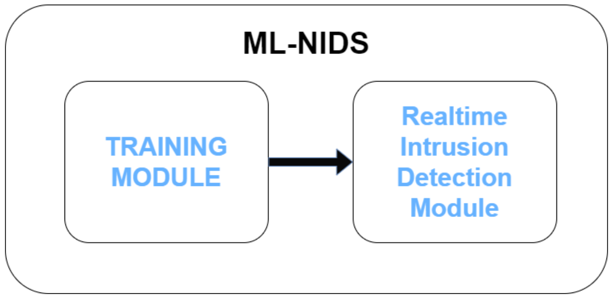

Fig. 1. Proposed NIDS concept

The main objective of the **Training Module** is to develop a trained ML model, which can effectively classify new arrived flow data. This module collects the dataset in our testbed and use it to train the ML models. The second module, called **Realtime Intrusion Detection Module**, uses the trained model in the first module to classify the received data flows in order to detect malicious flows on the fly. The paper based on the obtained result of this second module to evaluate the accuracy and efficiency of each trained model.

### B. Training Module

This module consists of three components: *Forwarding App*, *Data Collector*, and *Machine Learning* Module. The two first components are two services running on the application of our experimental SDN network.

The main task of the *Forwarding App* component is to install flow rules on switches when a PACKET_IN message is delivered to the SDN controller (when no matching rule for an incoming packet). The work flow of *Forwarding App* component is described in Fig. 2.

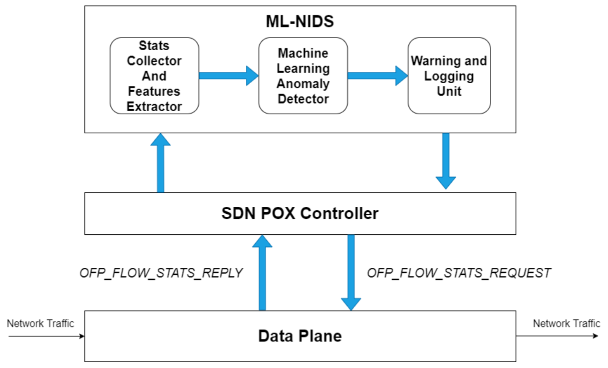

Fig. 2. Forwarding App workflow

For each flow of a switch, the match field is packet's parameters that are used to decide whether that packet belongs to the flow. It consists of many header fields of a packet (from Data Link Layer to Transport Layer). For our general purpose of detecting attacks working on three specific protocols (ICMP, TCP and UDP), the match fields for packet type are described in Fig. 3 in order to gather useful features.

| **ICMP** | | | | |
|---|---|---|---|---|
| Network Protocol | Src. IP address | Dst. IP address | | |
| **TCP/UDP** | | | | |
| Network protocol | Src. IP address | Dst. IP address | Src. Port | Dst. Port |

Fig. 3. Flow entry's match field

The second component, *Data Collector*, periodically sends statistical information request to the switch. Then, it extracts flow's features, labels each flow, and finally stores data into a file. For each flow entry, beside the provided match field , OpenFlow protocol also provides following additional flow information:

- **Duration**: The elapsed time of the flow since being installed in the switch.
- **Packet count**: the number of packets that correctly match with the flow.
- **Byte count**: the number of bytes of all packets that correctly match with the flow.

On the basis of this information, the match field, and the related flows from the same switch, the paper proposes a set of 8 aggregated features in Tab. I. These features are built upon the characteristics of different attacks.

As mentioned above, our dataset is collected in an self-constructed testbed. Each type of flow instances is gathered separately in its own testbed. The final dataset is combined by the individual collected data.

The collected dataset is used in the ML component to train the supervised learning models. In this paper, we propose two CNN models, CNN-1d and CNN-2d , then compare them with a basic Neural Network model and three traditional supervised learning algorithms: Naïve-Bayes, K-Nearest Neighbors and Support Vector Machine. The evaluation result of experimental works is based on the performance and the accuracy.

As shown in Fig. 4, the paper uses a basic ANN model with an input layer, three hidden layers, and an output layer for experiments. The hidden layers contain 10 neurons each. The model's parameters are set up with 64 for batch size and 300 for epoch.

Our experiments showed that there is a connection between some of the features for each output class. That similar point with image processing motivates us to propose two Convolutional Neural Network (CNN) models (Fig. 5). For the first model, called CNN-1d, the paper uses two convolutional layers (with 16 and 32 neurons, respectively) and two pooling layers. In this model, the kernel is a 1-direction matrix with the size of 2. For the second CNN model, called CNN-2d, the paper converts each of our data sample into a 2x4 matrix and applies convolutional layer with kernel size of 2x2. Both CNN models are set up with 64 for batch size and 500 for epoch.

The dataset of these Deep Learning models is divided into 2 subsets: training and validation set, with the proportion of 70% and 30%, respectively. The paper uses the validation set to evaluate the model weight every epoch and uses the validation result to modify model weights in using backpropagation algorithm. The epoch giving the best evaluation loss value is stored and considered as the result of the model's training phase.

| **Ind.** | **Feature** | **Description** |
|---|---|---|
| 1 | protocol | The flow's network protocol (ICMP, TCP, UDP). |
| 2 | Transfer rate | The average number of packets matched the flow in 1 second. |
| 3 | Packet size | The average size of a packet. |
| 4 | Host ratio | The ratio of the number of hosts in the same network that the source host has connection to, to the total hosts ratio. |
| 5 | Host number | The number of hosts in the same network that the source host has connection to. |
| 6 | Different port number | The number of flows having the same network protocol, source and destination IP Address, or the number of (Src Port, Dst Port) pair that two hosts have connection to each other. |
| 7 | Different source port number | The number of flows having the same network protocol, source IP Address, destination IP Address and destination port or the number of ports that the source host uses to connect to the same port in destination host. |
| 8 | Opposite packet ratio | The ratio of the opposite flow (same protocol but opposite IP Address and port)'s packet num to the flow's packet number. |

TABLE I
AGGREGATED FEATURES

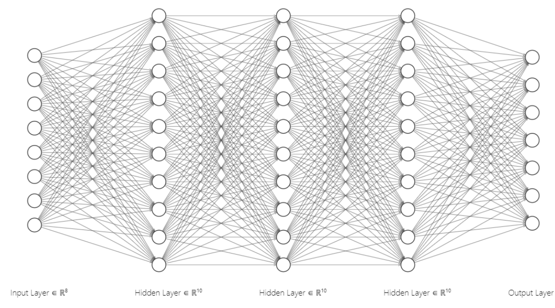

Fig. 4. Basic ANN model

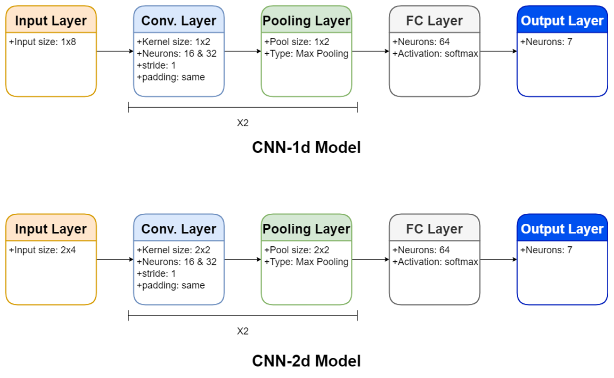

Fig. 5. Proposed CNN models

### *C. Realtime Intrusion Detection Module*

The proposed NIDS architecture is depicted in Fig. 6. In this architecture, the ML-NIDS is an application communicating with switches through SDN Controller. With this method, the NIDS can request statistical information of every switch at anytime needed. Therefore, it can take advantage of this global view for intrusion detection. In our design, the system starts with an initial OFP_FLOW_STATS_REQUEST message sent to switches. Then, OpenFlow switches send back to the controller an OF_FLOW_STATS_REPLY message with all the requested statistics. After, through the controller, our ML-NIDS application gathers that information, extracts needed features for each flow and uses the trained models from Training Module to classify network traffics. Based on the classification result, the application can detect abnormal flows and also the attack types. After those steps, the system finishes the loop by waiting for a fixed amount of time and resend another stats request to start a new detection loop.

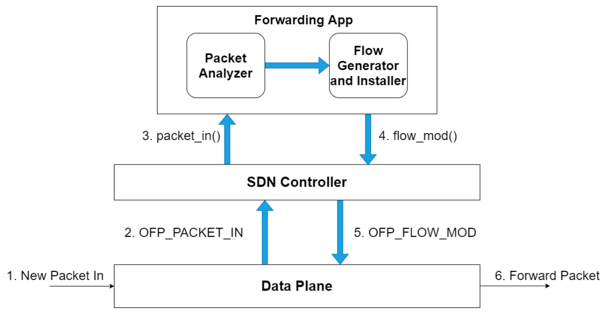

Fig. 6. Proposed NIDS architecture

For this proposed approach, instead of analyzing the whole network traffic (tracking every packets), the system only analyzes flows, which is significantly less than the number of packets. That helps our system to improve the performance of real-time tasks. On the other hand, as our analyzing system at the application layer works concurrently with the controller and Forwarding Apps (or other flow installer service), it does not affect the operation of the network. This approach also provides the ease of NIDS development, error handling, compared to multi middle boxes in traditional network or port mirroring in other systems [2], [4].

## IV. Experiments

### *A. Data collection testbed and results*

In order to collect data used in training machine learning models, we build a network simulation environment. We use the lightweight POX [8] as SDN Controller and use its provided APIs to develop our application. We also simulate a network infrastructure in using mininet [9]. The experiment testbed is described in Fig. 7.

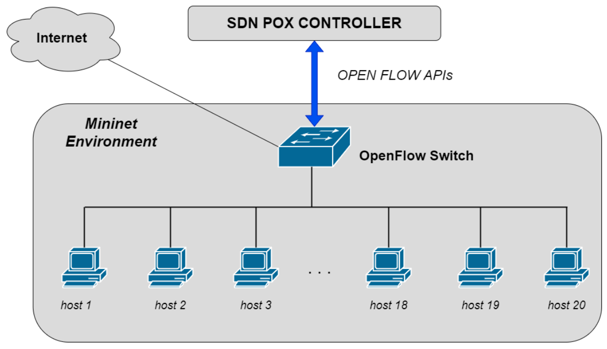

Fig. 7. Our Testbed

The paper aims to detect 6 types of attack, divided into two groups, as presented in Tab. II. Therefore, the dataset consists of seven label classes (normal traffic and six types of attack). The data of each class is collected individually with different scenarios. This method helps us to correctly label the data and cover many attack behaviors.

| Category | Attack type |
|---|---|
| Probe Attack | IP Sweep |
| | Port Scan |
| Denial of Service Attack | ICP Flood |
| | Ping of Death |
| | UDP Flood |
| | TCP SYN Flood |

TABLE II
Attack Types

The paper collects two separate datasets, one for training phase and the other one for validation phase. The details about two datasets is shown in Tab. III.

| Class | Training Set | | Validation Set | |
|---|---|---|---|---|
| | No. of samples | % | No. of samples | % |
| Normal | 3414 | 19.96 | 1242 | 20.99 |
| IP Sweep | 2007 | 11.73 | 647 | 10.93 |
| Port Scan | 2587 | 15.12 | 918 | 15.51 |
| TCP Flood | 3172 | 18.54 | 1045 | 17.76 |
| UDP Flood | 2462 | 14.40 | 821 | 13.87 |
| ICMP Flood | 2224 | 13.00 | 784 | 13.25 |
| Ping of Death | 1240 | 7.25 | 461 | 7.82 |
| Total | 17106 | 100 | 5918 | 100 |

TABLE III
Data collection result

| | | Predicted | |
|---|---|---|---|
| | | X | Not X |
| Actual | X | True Positive | False Negative |
| | Not X | False Positive | True Negative |

TABLE IV
CONFUSION MATRIX OF A CLASS X

Comparing to KDD-99 dataset, our dataset has better classes distribution. The collected data is also more relative to SDN environment because it is obtained in a SDN network.

### B. Evalutaion Metrics

This paper uses four parameters that are widely used to evaluate the performance of a NIDS: True Positive Rate or Recall (TPR), False Positive Rate (FPR), F1 Score (F1) and Accuracy (ACC). To calculate those parameters, a confusion matrix must be considered. A confusion matrix of a class X consists of four values (TP, TN, FP, FN), representing the number of testing samples corresponding to the definition described in Tab. IV. The confusion matrix is calculated for each of the classes in the dataset.

The parameters used to evaluate the accuracy of the NIDS are described as follows.

- Accuracy (ACC)(Eq. 1): The overall percentage of true classification over the dataset.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

- Precision (P)(Eq. 2): The percentage of a true predicted class. We do not directly use this parameter to evaluate the models but need this to calculate the F1-score.

$$P = \frac{TP}{TP + FP} \tag{2}$$

- True Positive Rate (TPR) (Eq. 3): it shows the percentage of the number of a true predicted class versus all samples of that class.

$$TPR = \frac{TP}{TP + FN} \tag{3}$$

- False Positive Rate (FPR) (Eq. 4): it shows the percentage of a predicted class but actually not that class, known as false alarm rate. The lower FPR, the better model is.

$$FPR = \frac{FP}{FP + TN} \tag{4}$$

- F1-score (Eq. 5): the combined measure of accuracy of a machine learning model by considering both Precision (P) and True Positive Rate (TPR) or Recall (R).

$$F_1 = \frac{2}{\frac{1}{P} + \frac{1}{R}} \tag{5}$$

In our method, the execution time of the NIDS for each loop depends on the number of received flows. Hence, to evaluate the efficiency of the NIDS application, the paper uses the parameter *Time per flow* (Tpf) that is the average time for the system to analyze and classify a flow (Eq. 6).

| Classifier | Score (%) | Classes | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Naive-Bayes | TPR | 99.11 | 99.23 | 95.97 | 94.55 | 99.27 | 98.36 | 79.06 |
| | FPR | 2.53 | **0.0** | 0.3 | 0.0 | 0.03 | 1.77 | 0.0 |
| | F1 | 94.98 | 99.61 | 97.13 | 97.2 | 99.51 | 93.75 | 88.31 |
| | ACC | 96.18 | | | | | | |
| K-NN | TPR | 98.15 | 94.15 | 99.34 | 91.97 | 99.64 | 95.84 | 96.1 |
| | FPR | 3.55 | **0.0** | 0.12 | 0.0 | 0.27 | 0.35 | 0.0 |
| | F1 | 92.77 | 96.99 | 99.35 | 95.82 | 98.98 | 96.75 | 98.03 |
| | ACC | 96.54 | | | | | | |
| SVM | TPR | 93.16 | 99.38 | 97.93 | 99.43 | 99.88 | 100 | 98.5 |
| | **FPR** | **0.49** | **0.0** | 0.04 | 0.0 | 0.29 | 1.59 | 0.0 |
| | F1 | 95.54 | 99.69 | 98.85 | 99.71 | 99.04 | 95.08 | 99.25 |
| | ACC | 97.95 | | | | | | |
| Basic ANN | TPR | 99.44 | **99.54** | 99.67 | 99.8 | 97.34 | 99.5 | 98.5 |
| | FPR | 0.68 | **0.0** | **0.01** | 0.0 | 0.02 | 0.2 | 0.0 |
| | F1 | **99.45** | **99.77** | 99.56 | **99.9** | 98.6 | **99.12** | 99.25 |
| | ACC | 99.19 | | | | | | |
| CNN-1d | TPR | 99.52 | 99.38 | 99.56 | 99.81 | 99.15 | 97.35 | 95.5 |
| | FPR | 0.76 | **0.0** | 0.12 | 0.0 | 0.04 | **0.14** | 0.0 |
| | F1 | 98.33 | 99.7 | 99.46 | **99.9** | 99.45 | 98.22 | 99.24 |
| | ACC | 99.14 | | | | | | |
| CNN-2d | TPR | **99.68** | 99.54 | **100** | **100** | 99.03 | 97.48 | 98.05 |
| | FPR | 0.64 | **0.0** | 0.1 | 0.0 | **0.0** | **0.14** | 0.0 |
| | F1 | 98.65 | **99.77** | 99.73 | 99.73 | 99.51 | 98.28 | 99.27 |
| | ACC | **99.29** | | | | | | |

TABLE V
NIDS DETECTION EVALUATION RESULT

$$Tpf = \frac{\text{Total execution time}}{\text{Number of flow}} \tag{6}$$

### C. Experimental results

For each model, the paper calculates three parameters including F1, TPR, FPR for each of the classes in our dataset. ACC is calculated for all classes. For the ease of presenting the result, we set the ID of the classes as follow: 1-normal, 2-IP Sweep, 3-Port Scan, 4-TCP SYN Flood, 5-UDP Flood, 6-ICMP Flood, 7-Ping of Death. The evaluation result measured on validation dataset is presented in Tab. V.

As shown in Tab. V, among six implemented ML models, Naïve-Bayes gives significantly lower result in Ping of Death class compared to other models. This might be the consequence of low distribution of this class in the training dataset.

KNN model gives the worst FPR score for normal flows. That implies that KNN would give the highest false alarm rate when applied in intrusion detection. KNN algorithm predicts new flow according to a few nearest samples in the training set. This indicates that there are some normal flow data in the training set that have similar information with attack traffic's flow. Also, in the KNN model, the TPR of class TCP SYN Flood is noticeably lower than other model's. This can be explained that, in our experiment, we try some Distributed Denial of Service (DDoS) technique for TCP SYN Flood testbed. We use multiple hosts attacking to a single host with low rate. Therefore, some TCP SYN Flood flows have similar characteristic to normal traffic. Hence, the KNN's map has an area that normal TCP data and TCP SYN Flood data are mixed, and this causes our model to misclassify some TCP SYN Flood samples into normal TCP flow, therefore does not deliver satifactory results.

| **Model** | Naive-Bayes | KNN | SVM | Basic-NN | CNN-1d | CNN-2d |
|---|---|---|---|---|---|---|
| **TpF** | 0.45ms | 0.47ms | 0.38ms | 0.55ms | 0.67ms | 0.57ms |

TABLE VI
EXECUTION TIME EVALUATION

SVM algorithm uses hyper plane to separate each class's area in an 8-dimension space. For each attacking class, the group of samples is usually convergent at separate area. However, with normal traffic flow, the samples are spread at a larger area, which makes SVM's hyper plane for normal class are not very accurate. As a result, normal class's TPR value is the lowest in six models.

We also notice that our three Deep Learning models (Basic ANN, CNN-1d, CNN-2d) provide significantly overall better results than traditional ML models. All of them give $ACC > 99\%$, while CNN-2d model gives the best result with overall accuracy $\sim 99.29\%$. However, the result difference between them is negligible. Therefore, the two CNN models, which take longer time to train and predict due to complex architecture, are not efficient as we thought.

In our NIDS system, the paper also records elapsed time of each detection loop and the number of flows sent from controller to calculate *Tpf* values for each ML model. In our experiment, we use a computer with CPU Intel Core i7 4770HQ and 16GB RAM to run the SDN controller and our NIDS application. The obtained result is represented in Tab. VI.

Tab. VI shows that Deep Learning models take longer time (about 25-40%) compared to traditional ML models. However, for example, for a switch with 2000 flows (it is considered as normal in a large network), our NIDS takes only more than 1 second to analyze and return the result.

For conclusion, the Basic ANN model has the best performance in our detection module with high classification accuracy and an acceptable execution time.

## V. CONCLUSION

In this paper, we have proposed and developed a ML-based NIDS running in a SDN network. Three proposed Deep Learning models have been proposed and compared to traditional ML algorithms. The experimental result shows that the proposed approach with a high accuracy has promising potential for further development compared to traditional approaches.

Concerning the future works, we will try to implement our system in a real SDN testbed with physical SDN devices (Openflow switch) and reevaluate the performance, and collect more realistic data to improve the system in a SDN network.

## ACKNOWLEDGMENT

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.02-2019.314

## REFERENCES

[1] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *International workshop on recent advances in intrusion detection*. Springer, 2011, pp. 161–180.

[2] P. Manso, J. Moura, and C. Serrão, "Sdn-based intrusion detection system for early detection and mitigation of ddos attacks," *Information*, vol. 10, no. 3, p. 106, 2019.

[3] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.

[4] A. Le, P. Dinh, H. Le, and N. C. Tran, "Flexible network-based intrusion detection and prevention system on software-defined networks," in *2015 International Conference on Advanced Computing and Applications (ACOMP)*. IEEE, 2015, pp. 106–111.

[5] S. Hettich and S. D. Bay, *KDD Cup 99 Data*, 1999. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/

[6] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *IEEE Local Computer Network Conference*. IEEE, 2010, pp. 408–415.

[7] J. Li, Z. Zhao, and R. Li, "A machine learning based intrusion detection system for software defined 5g network," *arXiv preprint arXiv:1708.04571*, 2017.

[8] S. Kaur, J. Singh, and N. S. Ghumman, "Network programmability using pox controller," in *International Conference on Communication, Computing & Systems (ICCCN'2014)*, 2014, pp. 134–138.

[9] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*. IEEE, 2014, pp. 1–6.