International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. #, No.# (Mon-20..)

Exploring Honeypot as a Deception and Trigger Mechanism for Real-Time Attack Detection in Software-Defined Networking

Harman Yousif Khalid [1], Najla Badie Aldabagh [2]

[1]Department of Computer Science, College of Science, University of Duhok, Duhok, Iraq
[2]Department of Computer Science, College of Computer Science and Mathematics, University of Mosul, Mosul, Iraq
E-mail address: Harman.khalid@uod.ac, Najlabadie@uomosul.edu.iq

Received ## Mon. 20##, Revised ## Mon. 20##, Accepted ## Mon. 20##, Published ## Mon. 20##

Abstract: Cyberattacks are becoming more frequent and sophisticated, making their detection harder. Probe attacks in Software Defined Networking (SDN) not given much attention by the research community, which represents the starting phase for other attacks. The attacker scans the network to get the necessary details about hosts and services running in network to launch successful attacks exploiting vulnerabilities in the system. The issue with probe attacks is that they occur passively and the target system is not aware of them. On one hand, additional mechanism is required to check the network traffic continuously by embedding switches with independent agents, which is against the OpenFlow standard. On the other hand, using statistics provided by OpenFlow switches to the controller, which overloads the controller with the extra task of continuously checking traffic statistics. In this work, a lightweight detection mechanism proposed that detects probe attacks in real-time using machine learning. Honeypot integrated into the detection mechanism to detect passive probe attacks by luring attackers through proving fake services and serving as a trigger mechanism that activates the detection mechanism when necessary. The experimental results show that the proposed mechanism successfully detects probe attacks in real-time achieving accuracy (94.73%) with the minimum CPU load.

Keywords: *Intrusion Detection System (IDS), Software Defined Networking (SDN), Probe, Reconnaissance, Honeypot, Machine learning (ML).*

1. Introduction

Probe attacks are often considered a preliminary step in various cyberattacks, particularly in the context of network reconnaissance. Network reconnaissance is the first phase of the cyber kill chain, which involves gathering information about a target system or network to identify vulnerabilities and potential entry points for further exploitation [1]. Probe attacks involve port scanning, ping sweeps, network mapping, etc. to discover active hosts, open ports, services, the type of operating system of the target host, etc. Since such information could be useful during attack planning to exploit vulnerabilities associated with a specific OS or understand how different systems are connected. After obtaining the necessary information, attackers may perform vulnerability scanning to find specific weaknesses or vulnerabilities in the target systems and identify potential entry points and weaknesses in the target's network. After the initial reconnaissance phase is complete, attackers may proceed with more targeted and specific attacks based on the information gathered, such as denial of service (DoS), man-in-the-middle (MitM), brute force, etc.

The issue with probe attacks is that they operate in passive mode and are very hard to detect. Several methods are used by the research community to detect probe attacks, such as deploying Snort Intrusion Detection System (IDS), which continuously monitors the network traffic to detect malicious traffic. However, Snort is not effective to deal with today's attacks because they are signature-based, and any simple deviation in the attack signature could easily be bypassed by the attacker [2][3]. Moreover, embedding an OpenFlow switch with additional duties is against the OpenFlow protocol since it clearly states that the switch must be as simple as possible and perform only packet forwarding.

SDN's features, such as having full control and view over the whole network as well as programmability, open the door for the deployment of various security applications through the existing open API. Some

E-mail:author's email

researchers exploit the power of SDN controllers to deploy artificial intelligence (AI)-based IDS to detect various types of attacks. AI-based IDS methods require features extraction from networks to check current flows, whether they are malicious or normal. The OpenFlow protocol provides a method to gather various statistics details from switches through OpenFlow statistics message requests and replies. The controller initiates the request message to all switches, and the switches provide the required details for the controller. AI-based IDS take advantage of those details for feature extraction to detect malicious flow. However, periodic inquiries about those features and corresponding replies lead to huge problems for the controller.

This work is extended from a survey [4], where a deep investigation was conducted on the works performed by other researchers regarding the deployment of AI-based IDS in SDN environments; some open issues and challenges were highlighted. The survey shows that, due to the difficulty of traffic feature extractions and mapping them to the features in the dataset they used, almost all previous studies on deploying IDS neglected real-time attack detection in SDN. They have primarily concentrated on obtaining high accuracy through the utilization of novel machine learning (ML) and deep learning algorithms (DL), or by employing feature selection methods during offline training. There was no clear implementation of real-time detection work, and the accuracy of the literature did not reflect reality, as it did not run in real-time. In addition, the survey showed that most of the current work only paid attention to DDoS attacks, neglecting other attacks since the nature of the centralized controller of SDN makes them an attractive target for DDoS attacks. Moreover, majority of works deployed IDS, checks the network for malicious traffic on a fixed time interval. A longer time interval may make it more difficult to identify the attack in the early stages and may even give the attacker more time to cause more serious damage to the network [2]. While if interval is very short, it has many consequences, such as increasing the controller's CPU load, especially in large-scale networks [5].

In this work, we aim to solve the mentioned open issues, and we propose a lightweight machine learning mechanism to detect passive attacks such as probe attacks in an SDN environment using features provided by OpenFlow switches. The mechanism comprises of a triggering module based on honeypots and detection and mitigation modules. The following is a summary of this work's primary contributions:

- Early detection and prevention of attacks since we consider probe attacks, which regarded as the initial stage of nearly all other kinds of attacks, including DDoS, botnet, MitM, , etc. In other words, we break the first phase of the attack kill chain before gaining control of the target system.
- To identify passive attacks, we proposed a lightweight mechanism by exploiting the honeypot's capability, which acts as a trap by luring attackers by providing fake services.
- We used the honeypot as an alerting mechanism to minimize the CPU overhead of the SDN controller in a large-scale network by triggering the detection module when needed instead of continuously checking the traffic. Moreover, the honeypot contributes to filtering the flow of traffic by providing additional useful details about the attacker.
- We conducted a simulation scenario to verify the proposed mechanism in real-time, considering attack detection in its early stage as well as attack mitigation.

2. Related Work

The novel SDN architecture allows the research community to take advantage of its features of programmability, flexibility, and ease of deployment. Implementing a security application, which resides in the controller, exploits the power of ML and DL to detect malicious traffic in a network. This section briefly introduces some recent and popular approaches that have been suggested for attack detection in an SDN environment using AI capabilities.

The Grey Wolf Optimization (GWO) algorithm for feature selection was implemented in [6] to improve the performance of IDS to detect probe attacks more accurately. They discussed the benefit of feature selection to the overall detection model. They highlighted that feature selection is essential to minimize the computation time, which will make the classifier have high accuracy with optimal features selected as well as decrease the dataset size for testing and training. Moreover, for real-time detection, it is easier to extract fewer features, thus decreasing the detection time. They showed that by selecting a subset of 8 features from the InSDN dataset using the Light Gradient Boosting Machine (LightGBM) classifier, accuracy increased to 99.8%, while using all features was 77.3%. However, their topology was the same as that of the creator of the dataset, and they did not perform real-time detection.

In another direction, some studies have developed hybrid IDS that merge flow-based IDS with signature-based IDS to provide a more robust detection mechanism. The author in [7] implemented two approaches for detecting DDoS attacks in SDN. First, they use signature-based Snort IDS alongside with SDN to analyze the network for checking malicious traffic. Second, implementing a machine learning Support Vector Machine (SVM) model trained with the NLS_KDD dataset to detect unknown attacks. The motive behind using both methods is that the unknown attacks are detectable by machine learning, and their signatures will be stored in the Snort database to make Snort able to detect them next time they occur again. The drawback of this method is that they implemented Snort as an independent hardware module connected to a switch, which requires additional resources and whose performance degrades when the network is larger.

In the same context, the author in [8] used Snort IDS, which connects to an Open vSwitch and monitors the network through port mirroring. They also provide flow-based IDS in controller to overcome the shortcoming of Snort being unable to detect novel attacks. The OpenFlow statistics message was used to extract features for machine learning over a certain time interval. They selected seven features that can be easily obtained by SDN nature. However, using a combination of both Snort and machine learning leads to issues in large networks and creates a load.

The detection of DDoS attacks in SDN environments has been covered in [9]. The author proposed two lines of security. First, they used Snort to detect known attacks in signature-based databases. The second defense line was using ML and DL to detect anomaly-based attacks. They used SVM and Deep Neural Network (DNN) models, which trained on the NLS_KDD dataset, and the accuracy was 74.3% and 92.3%, respectively. Snort used in this work for detecting known attacks in signature databases and as a data collector for ML and DL models. However, they periodically monitor the network for anomalies, which makes their detection mechanism active even when there is no traffic overload on the controller.

In [10], a comparative analysis of different feature selection algorithms for detecting DDoS attacks based on various machine learning models is presented. The experiment was conducted using feature selection algorithms such as Information Gain (IG), Correlation Coefficient, Chi-square, Forward Feature Selection (FFS), Backward Feature Selection (BFS) and Recursive Feature Elimination (RFE). Machine learning classifiers such as SVM, Decision Tree (DT), Random Forest (RF), Naïve Bayes (NB), and K-Nearest Neighbor (KNN) used for binary classification. The optimal model was RF with an accuracy of 99.97% using a feature subset of 28 selected by the RFE. They mentioned that the detection model used OpenFlow statistics messages to get 41 features of NSL-KDD. However, it is very difficult to get those numbers in real time, as highlighted before.

In [11], an attack detection and mitigation module was proposed using a hybrid model of Convolutional Neural Network (CNN) and Extreme Learning Machine (CNN-ELM) to classify DDoS attacks in an SDN environment. Their model found DDoS attacks by using information taken from the SDN environment. This information came from both packet-in messages sent to the controller and statistics messages sent to the controller by OpenFlow switches. The extracted features from the OpenFlow switch were mapped to a subset of 12 features in the InSDN dataset. In addition, they constructed four additional features, such as average speed flow, average duration, average packet size, and ratio asymmetric flow. Through the experiments conducted, they showed that using a subset of 12 features not only increased accuracy but also reduced test time. However, this methodology creates overhead in the controller since every packet-in message should be checked, which will not be effective during a DDoS attack. Moreover, there was no clear description of how features would be extracted from packet-in, and their methodology was not verified. Moreover, the manually created four features were not verified either.

Similarly, the authors in [12] proposed the Deep Convolutional Neural Network (DCNN) to detect DDoS in SDN. They suggest similar detection and mitigation mechanisms to previous work. Except that they used only the features provided by the flow table through OpenFlow statistics messages, and those messages periodically sent to the controller for anomaly detection. They mentioned that they only used 12 features of InSDN mapped to extracted information from the OpenFlow switch. They argue that the existing system suffers from using a large number of features for machine learning or deep learning and needs more functions to extract them, which creates network congestion and latency. While using a small and limited number of features does not provide reliable attack detection. However, in practical implementation, they used 78 features for training, not only 12. It is difficult to map the basic features provided by OpenFlow switches to the 78 features of the InSDN dataset. Moreover, their methodology, which requires every packet-in message to be checked by the controller as well as periodically requesting statistics from the switch, creates overhead in the controller. Similar to previous work, there was no clear description of how features would be extracted from packet-in, and they were not verified.

A work in [13] proposed a lightweight supervised learning model to detect DDoS attacks against SDN controllers using only one feature of fluctuation of flows, which is the count of packet-in messages to the controller in a fixed time slice and for many consecutive times to avoid the behavior of a normal burst. They created their own dataset for the proposed system, but for testing and training their model, they used the InSDN dataset. The idea behind using only one feature is that it will be easier to obtain as well as consume less time and resources for training and real time prediction. They used multiple machine learning models with seven selected features of InSDN, which were flow-ID, protocol, timestamp, flow-pkt/s, bwd-pkt/s, pkt-len-mean, init-bwd, and win-byts. The conducted experiments show Binary Tree (BT) and KNN were the best in terms of accuracy, while in terms of both accuracy and training time, CPU utilization, and decision time, KNN was the optimal. They tested the proposed work using their own dataset and obtained an accuracy of 99.4% with BT using one feature. The author argues that using many features will lead to either higher performance or overfitting for some models. However, they did not mention the methodology of feature selection, and some of the selected features were irrelevant, such as flow-ID and timestamps, which can affect the learning process during model training. Moreover, continuous checking of the count of packet-in created load on the controller as well as the methodology of time slice create delay in decision time. Finally, the method of considering only one attribute for training the intrusion system is not a promising solution.

In summary, the majority of the existing work in the literature targets DDoS attack detection only, neglecting other attacks. The focus of the previous works was on the analyzing of the proposed models to achieve high accuracy or implementing some studies on feature selection algorithms, and there was no clear description of real-time detection. Moreover, some works monitor packet-in messages as well as statistics message to construct their features, but none of them considers the overload that created on controller when periodically checks the network traffic or extract traffic features. In this work, we consider probe attack, which considered as first step of other attacks and extract limited number of features through statistics provided by OpenFlow switches to ensure the fast and lightweight detection mechanism. We consider the tradeoff between high accuracy and low controller overhead by using triggering mechanism for checking malicious traffic when needed instead of periodically manner. We implemented our own dataset, which contains features dimensions that easily obtained from OpenFlow switches and evaluated the efficiency of dataset using many common supervised learning algorithms such as DT, RF, Adaptive Boosting (AdaBoost), NB, XGBoost and KNN.

3. The Proposed Detection Mechanism

In this section, we describe the methodological stages followed to build the proposed probe detection mechanism. The SDN features make SDN operation easier and offer a number of benefits [14]. This motivates the deployment of light, effective, and attack detection in real-time. Our proposed SDN defense system distributed on two sides, as shown in Figure 1. On the controller side, where the detection and mitigation modules reside, as well as on the host side, where honeypot resides as a trap to lure attackers and notify the controller when possible malicious traffic is detected. Honeypot is a deception mechanism used to lure attackers by providing fake services. Any contact with the honeypot considered as a possible attack. This contact will trigger the detection module in the controller to start checking the current flows in the network instead of continuously checking the network periodically. In addition, the honeypot programmed to send some useful details to the controller with a triggered message about the attacker traffic to filter out the possible malicious flows and reduce the load on the CPU.

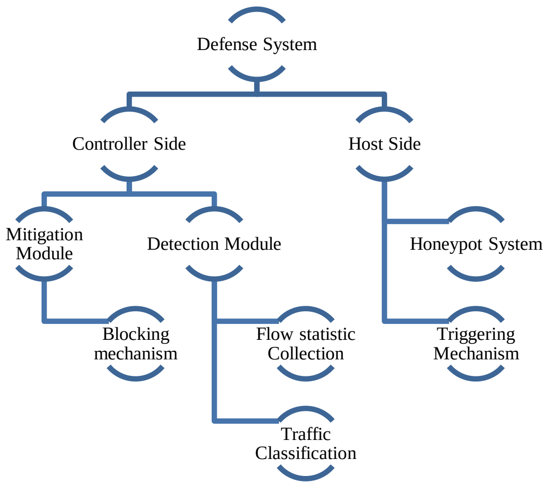

Figure 1. Defense system modules diagram.

Machine learning techniques have recently dominated IDS research because they produce more accurate predictions than other techniques [15]. Machine learning models can overcome the drawbacks of other methods by classifying abnormal traffic as an anomaly with self-learning capabilities [16]. ML has been deployed in a wide area, from medical analysis and image processing to data mining. The concept of machine learning is to make machines learn automatically from the given training data without human intervention [17]. However, machine learning required input values to feed to create prediction output based on those inputs. These inputs should be extracted from the live attributes of traffic flow to accurately detect malicious traffic. SDN controllers' beneficious features, such as power and storage provided, open programmability, global visibility and control, and statistics features provided by the OpenFlow switch, make them suitable locations for implementing machine learning intrusion detection and mitigation applications. Due to the availability of those features, in proposed work, machine learning models are implemented in the controller for classification.

As shown in Figure 2, the detection and mitigation applications are inactive in the controller, waiting to be triggered to check traffic flow when necessary. Honeypot is actively providing some fake services to lure attackers along with real services on the network. The defense mechanism can be described in pseudocode in Figure 3. When the attacker first initiates a probe attack to check for active hosts and services in the network, the honeypot is configured to register all events on a dedicated log file for every connection attempt to the services they provide. In general, the log file used for analyzing the attacker's movements and steps during the attack.

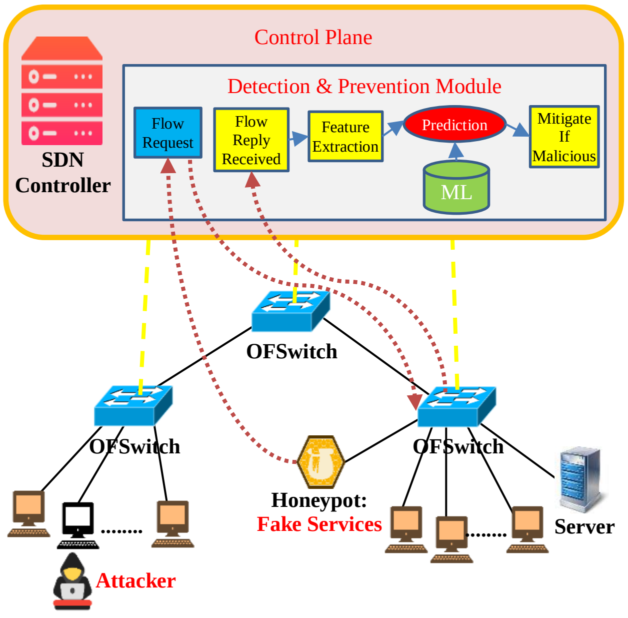

Figure 2. The framework of the proposed mechanism.

```
1: If  packet_in message, then
2:          If Ethertype = ‘0x88FF’
3:          Craft OFP stats request to switch(ID)
4: if OFP stats reply message
5:          Filter according to (IP, port, protocol)
6:          Extract features from filtered flow
7:          Result= ML_Model (Extracted features)
8:          If Result ==0 then
9:                    Take no action
10:         Else: Result ==1 then
11:                   Block the attacker port
```

Figure 3. Defense System Pseudocode

However, in this work, we utilize the honeypot to be exploited as a triggering mechanism to alert for possible attacks. Once the log file registers an event, a special Python script monitors the log file and extracts the IP of the attacker, the protocol of flow, and the port used by the attacker. Later, the mechanism constructs a special packet-in message to be sent by the OpenFlow switch to the controller with a special ethertype field code ‘0x88FF’ to be recognized by the controller as a trigger message. In the next step, once the controller receives the trigger message, it will forward it to the detection and mitigation application. The application creates a flow statistics request message to be forwarded to the switch, which is connected directly to the host where honeypot exists through the switch ID provided by the packet-in trigger message. After receiving the reply messages from the switch, which contain flow entry statistics, they are filtered by other details provided by Honeypot, such as IP, port, and protocol. Then the necessary features extracted from the filtered flows and fed to the ML module for testing. Once the malicious flow detected, the attacker domain located and controller initiated packet-out message to block the attacker's source port.

A. *Dataset Preparation*

Using a high-quality dataset for training AI-based IDS has a great impact on the accuracy of the predictions made by the model. Comparing to other domains, such as computer vision, there are limited datasets for intrusion detection in general and for probe attacks specifically due to privacy and legal issues [18] [2]. In the past few decades, academics have used a number of well-known public datasets, including the KDD Cup, NSL KDD, UNSW-NB15, and CICIDS2017 databases. Since SDN is a new paradigm, comparing to traditional networks, there are less research studies that address the intrusion detection problem in SDN. According to the authors of [15][19][20], most published research work deals with the intrusion detection in SDN similar to that of a conventional network.

Scholars employed classical network datasets, like KDD Cup and NSL-KDD, as training datasets for anomaly detection in SDN contexts. Training the SDN-based IDSs using old datasets can cause significant issues, as they only detect attacks, which have similar behavior in both SDN and traditional networks. Because intrusion attack tactics are always evolving, they are getting more complex and difficult to spot [19]. However, these datasets are either unreliable and outdated since they were released a long time ago [15] [21], or they suffer from compatibility issues since they were collected during the traditional networks and the SDN architecture are different. Therefore, they are not efficient to use for real-time detection [21].

The InSDN dataset [22] published as an SDN specific dataset in 2020 for the purpose of assessing and training IDS within SDN environments. The work in [4] conducted regarding this dataset reveals that applying this dataset for IDS deployment in SDN research work has mostly focused on achieving high accuracy by using new ML and DL algorithms or by using feature selection techniques in offline training. There was no clear implementation of real-time detection work, and the accuracy of the literature did not reflect reality, as it did not run in real-time. The reason behind that is the difficulty in obtaining the required features from the traffic flow to map to the features of the dataset.

In this work, we focus on a binary classification for detecting probe attacks and do not delve further into classifying the various types of attacks. In this work, we created our own dataset specific for training and evaluating the proposed IDS for detecting probe attacks in the SDN context. The dataset contains 282128 instances in total, with sample sizes of 126685 (45%) for the normal class and 155443 (55%) for the probe class, respectively. . The probe traffic collected by initiating a live host scan, port scan, version scan, service scan, OS detection, etc. For normal traffic, we replicated the proper random inter-departure time and packet size by creating traffic at the packet level as well as multiple sessions of parallel, sequential, and bidirectional streams. Numerous widely used application services, including HTTPS, HTTP, FTP, SFTP, Telnet, DNS, VoIP, and others, represented in the normal traffic. All hosts connected to the internet and ordinary daily tasks executed by executing several applications, such as browsing different websites, watching YouTube videos, opening and sending emails, to mimic real Internet traffic.

B. Feature extraction

Deploying AI-based IDS requires features from real-time traffic to feed the classifier model for decision making in real-time. The first task of the IDS is to collect traffic statistics and related information about flows. Many studies have implemented ML and DL models in SDN, but the source of data to provide features to perform real-time detection is not a straightforward process. Either they had to use methods used in traditional networks by using dedicated tools in the forwarding plane acting as sensors like Snort and sFlow agent, which degrade performance, or using features provided by SDN. Nowadays, networks are faster due to the requirements of applications and advances in technology. The intrusion detection mechanisms need to consider their performance in terms of detection time. The demand for lightweight IDS is a hot topic in academia nowadays. The trade-off between model accuracy and execution must be addressed carefully. Choosing the best features is essential to improving SDN-based IDS efficiency.

Having a large dataset with a huge number of features will delay training, testing, and model detection. In addition, using a high-dimensional dataset may not necessarily lead to higher accuracy due to overfitting and redundant features [23]. Reducing the number of features or selecting relevant features in general will lead to a decrease in model training, testing, and detection time, as well as model complexity, since real-time feature extraction is not an easy process [2] [23]. Moreover, feature selection reducing high dimensionality reduces the likelihood of overfitting issues in the model [2]. In order to obtain excellent model performance utilizing ML and DL, many researchers have turned their attention to the removal of noisy, duplicate, and useless features [2] [24] [25].

In the SDN framework, there is some point of feature extraction that can be beneficial for feeding the model such as:

1. Controller packet-in messages: Upon receiving the initial packet of a particular flow, the OpenFlow switch will examine the packet header and determine whether any flow rules exist in one of its flow tables that correspond with the flow in question [26]. There is an action connected to every flow rule. The traffic flow will be redirected to the appropriate destination if a match is identified. When there is no match and a miss table appears, the switch uses the packet-in message to convey the packet to the controller, who processes it and determines what to do with it. Some researchers used packet-in behavior to construct some useful features that successfully detect DDoS attacks, as in [11] [12] [13] [27].

2. OpenFlow statistical messages: Basic information about the flows could be obtained by parsing the OFP_stats_reply messages of an SDN network [21]. These messages were sent to the controller in response to OFP_stats_request by the controller. Usually, this happens periodically, depending on the specific event triggered. This process does not require additional effort because it is provided by the controller according to the OpenFlow specification.

3. Using independent agents: using other methods used for capturing network traffic, like in [27], which combined an OpenFlow switch and sFlow for effective anomaly detection in an SDN environment [28]. In addition, others in [9] have used Snort as a data collector.

In our mechanism, we used the OpenFlow statistical messages method to obtain the required features since they are easily obtained in an SDN environment and do not require an additional agent extended to the data plane switch following the OpenFlow protocol specification. The features extracted are shown in Table 1.

C. Preprocessing stage

An essential step in preparing the input data for the model's training in order to create an accurate detection system is data preprocessing. The generated dataset does not contain socket information such as source IP, destination IP, flow ID, etc. to avoid the overfitting problem [2], where such data can be changed from network to network. In addition, since we aimed to record the features related to the behavior of attacker flow, we excluded source and destination ports because they are often constant values representing network identifier attribute which may lead to overfitting. As the dataset does not contain a large number of samples, we deployed ML mechanisms instead of DL, since the latter proved to be effective with large datasets [18] [19]. As a first step in preprocessing as depicted in Figure 4, the dataset is checked to see if it contains empty or null values to prevent any significant effect on the model efficiency. Moreover, duplicated samples were removed from the dataset. Normalization, which involves transforming data to a common scale, is an essential preprocessing step in machine learning. We used Min-Max scaling to make values between “0” and “1”, to ensure that all features have the same scale, making it easier to compare their importance and contributions to the model. We divided the dataset where 80% is used for training and the other 20% is kept aside for the model test to see how well we can predict the data.

Table 1: Features used in proposed mechanism.

Feature	Description
flow_duration (nsec)	The duration of flow remaining in the switch flow table in nanoseconds
ip_proto	The flow protocol
srcport	Source port
dstport	Destination port
byte_count	Total flow bytes
packet_count	Total flow packets
byte_s	Number of flow bytes per second
pkt/s	Number of flow packets per second

We performed cross validation with k = 5 to make sure there is no bias in performance estimation to prevent an imbalanced occurrence where the minority class is significantly underrepresented compared to the majority class. The fundamental principle of cross-validation is to divide the dataset into several folds, or subsets, and train and test the model repeatedly on various combinations of these folds.

D. Practical implementation

The proposed work is implemented using Python programming language. Mininet version 2.3.1, beta 4, network simulator designated for SDN education and research was utilized to implement and evaluate the suggested technique. . Mininet was installed on a virtual machine, Ubuntu 20.04, with 12 GB of RAM and 4 CPU cores. The Ryu controller is used as the SDN controller in testbed for managing compliant switches. It operated with Layer four (L4) learning capabilities to forward flow traffic based on matching of MAC, IP, protocol, and service’s port. Scikit-learn, an open-source library used in our work to implement machine learning. On the host side, to deceive the attacker, we used Honeyed as a low interaction honeypot. During the experiment, we generated probe traffic using the NMAP tool and for normal traffic generation iperf and D-ITG tools had been used. Moreover, we performed some real tasks such as browsing the internet, watching YouTube, using emails, downloading and uploading files, and SSHing different mininet hosts to generate realistic traffic.

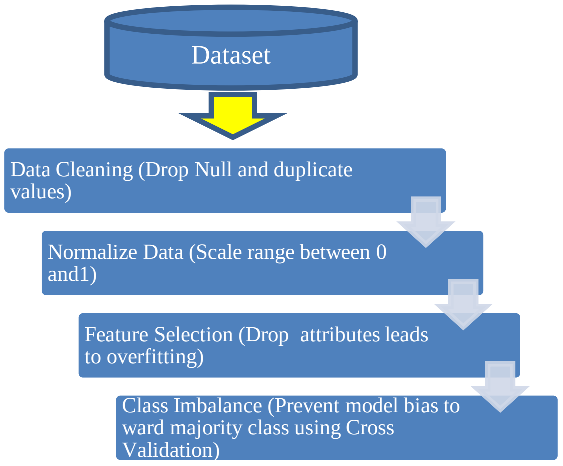
Dataset
Data Cleaning (Drop Null and duplicate values)
Normalize Data (Scale range between 0 and1)
Feature Selection (Drop attributes leads to overfitting)
Class Imbalance (Prevent model bias to ward majority class using Cross Validation)

Figure 4. Data Preprocessing Steps

4. Findings and Discussion

The performance of the suggested mechanism is evaluated using the evaluation metrics covered in depth in this section, which is then followed by a detailed presentation of the findings and outcomes. For evaluation, seven popular ML models adopted to determine the best model, including DT, RF, NB, AdaBoost, LightGBM, and KNN classifiers. By employing this diverse set of models, we intended to select the one that best fits the characteristics of our data and the requirements of the mechanism. The equations in Table 2 show the metrics used to evaluate the proposed mechanism in terms of accuracy, recall, precision, and F-measure, respectively.

Table 2: Evaluation Metrics Description

Accuracy	Precision	Recall	F1 Score
Percentage of True classified flow in relation to all classified flows	Percentage of malicious flows identified correctly in relation to all flows classified as attacks	Percentage of correctly classified malicious flows versus all malicious flows presented in the dataset	The harmonic mean between Precision and Recall
$\frac{TP+TN}{TP+TN+FP+FN}$	$\frac{TP}{TP+FP}$	$\frac{TP}{TP+FN}$	$\frac{2 * Precison * Recall}{Precision + Recall}$

Where TP, FN, TN, and FP stand for true positives, false negatives, and false positives, respectively. As depicted in [4], the majority of scholar articles used other metrics such as F1-score, precision, and recall because the average model's performance tends to favor the majority classes' performance, it is impossible to fairly assess classification performance under the imbalanced dataset using only the accuracy rate. A confusion matrix was utilized for each classifier in order to precisely assess the suggested mechanism and compute these performance indicators; the outcomes are displayed in Table 3. Among the ML models used, their results were very good except for NB since they are commonly used for text classification tasks. The XGBoost model achieves the best result in terms of accuracy 94.73% as well as precision, recall, and f1-score.

Table 3: Precision, Recall and F1 Score of Models Evaluation

Classifier	Accuracy	Precision	Recall	F1-score
DT	0.9390	0.9449	0.9378	0.9397
RF	0.9400	0.9457	0.9392	0.9410
NB	0.8449	0.8830	0.8356	0.8374
AdaBoost	0.9196	0.9260	0.9203	0.9218
XGBoost	0.9473	0.9544	0.9458	0.9480
LightGBM	0.9459	0.9532	0.9445	0.9467
KNN	0.9128	0.9170	0.9126	0.9139

Moreover, the author in [15] highlights the importance of the Area Under Curve (AUC) metric for imbalanced dataset classification. The AUC of the XGBoost classifier obtained good results of about 0.98%, as seen in Figure 5. The confusion matrix of the XGBoost classifier in Figure 6, shows that our detection mechanism successfully recognizes probe attacks, but the ratio of normal traffic recognized as probe attacks (false positive) needs more consideration.

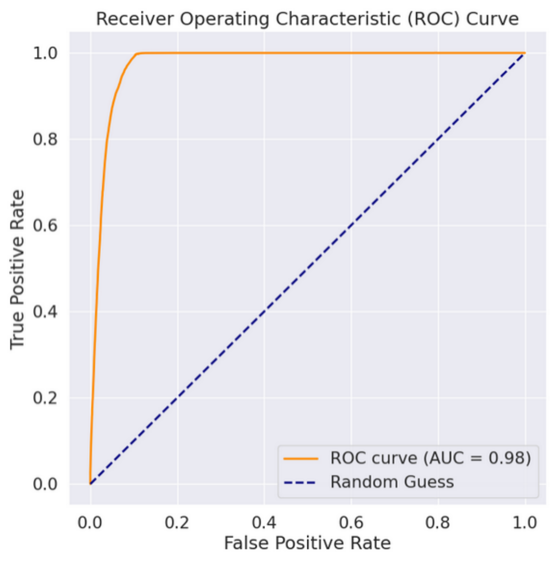

Figure 5: XGBoost AUC ROC Curve.

We conducted another experiment with a larger topology of tree with depth 4 and fanout 2, as shown in Figure 7, to support the hypothesis we previously highlighted in this article, which states that continuous traffic checking through statistics requests and reply messages by detection applications will lead to an increase in controller workload. This topology aims to increase the number of switches and hosts in the network, which will increase the number of flow rules in each switch. In this stage, normal traffic originated from each host to another, creating huge traffic.

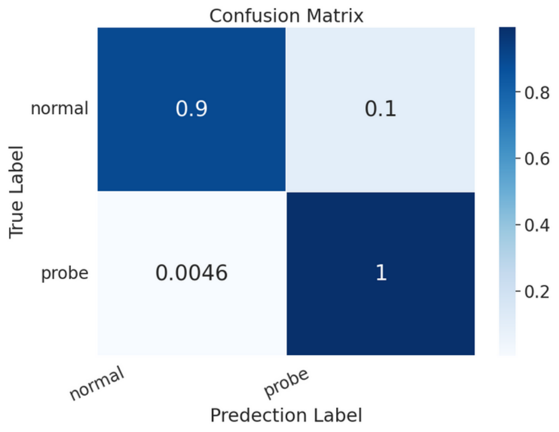

Figure 6: XGBoost Confusion Matrix

Four separate scenarios were employed in order to assess the controller CPU load, as seen in Figure 8. Several traffic checking intervals, such as 1, 5, and 10 seconds, were chosen for the first three scenarios of statistics request message sending. In the final scenario, honeypot triggering solutions are employed in place of periodic checking when necessary. Only normal traffic transmitted for all scenarios, and when controllers check the traffic for each second, controller overhead was quite high 33%. Controller overhead significantly decreased when the interval extended to every 10 seconds, but attackers in this case will have plenty of time to compromise the network. However, since the detecting mechanism is in sleep mode and would activate anytime there is an alarm, our idea of applying a triggering mechanism successfully lowered CPU demand to 2.1%.

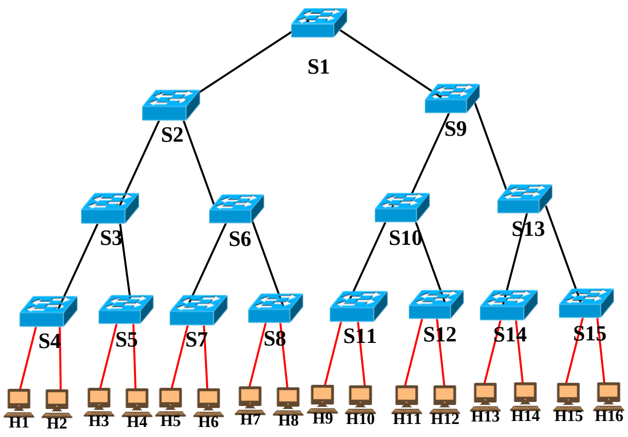

Figure 7: Tree Topology (Depth 4, Fanout 2)

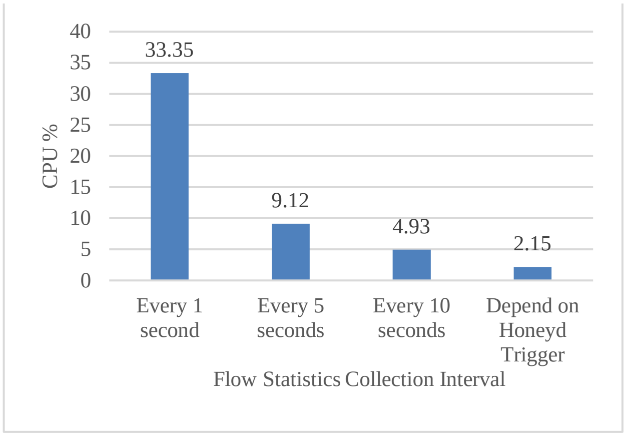

Figure 8: Ryu Controller CPU Utilization

5. Conclusion

Due to the growing threat of cyberattacks in SDN environments and the lack of built-in security mechanisms, this model finds it difficult to replace traditional network architecture. One important attack that is neglected by the research community is the probe attack. Probe attacks are considered the first phase of other serious attacks such as DoS, botnets, MitM, Brute force, etc. to exploit vulnerabilities in the network. The issue with this attack is that it is hard to detect since it works passively without being noticed. Machine learning and deep learning are widely used in current works for detection techniques because of their ability to detect novel attacks, contrary to classical methods of using independent agents in networks, which depend on fixed thresholds or attack signatures. In this work, we propose a novel online lightweight detection mechanism empowered by machine learning models. Our approach leverages the dynamic nature of SDN to embed honeypots within the network infrastructure. These honeypots in the network host serve a dual purpose: they lure potential attackers by presenting fake services, and they act as triggers for our detection mechanism in the SDN controller when any traffic contacts them. Machine learning models need a good dataset to provide accurate detection in real-time. Therefore, we collected a new dataset specific to probe attacks in SDN environments. Contrary to other datasets, which are, either outdated, incompatible with SDN, or whose features cannot be easily obtained, our dataset features were collected through OpenFlow statistics messages that can be easily collected in an SDN environment. Through extensive experimentation, we have demonstrated the efficacy of our proposed system. Notably, our detection mechanism achieves an exceptional level of accuracy, while imposing minimal stress on the SDN controller's CPU. These results underscore the practical viability and effectiveness of our approach in strengthening SDN security against evolving cyber threats.

References

[1] A. Alshamrani, “Reconnaissance Attack in SDN based Environments,” Institute of Electrical and Electronics Engineers (IEEE), Oct. 2020, pp. 1–5. doi: 10.1109/ict49546.2020.9239510.

[2] M. S. El Sayed, N. A. Le-Khac, M. A. Azer, and A. D. Jurcut, “A Flow-Based Anomaly Detection Approach With Feature Selection Method Against DDoS Attacks in SDNs,” *IEEE Trans Cogn Commun Netw*, vol. 8, no. 4, pp. 1862–1880, Dec. 2022, doi: 10.1109/TCCN.2022.3186331.

[3] A. S. Alshra’A, A. Farhat, and J. Seitz, “Deep Learning Algorithms for Detecting Denial of Service Attacks in Software-Defined Networks,” in *Procedia Computer Science*, Elsevier B.V., 2021, pp. 254–263. doi: 10.1016/j.procs.2021.07.032.

[4] H. Yousif, I. Khalid, N. Badie, and I. Aldabagh, “A Survey on the Latest Intrusion Detection Datasets for Software Defined Networking Environments,” *Technology & Applied Science Research*, vol. 14, no. 2, pp. 13190–13200, 2024, doi: 10.48084/etasr.6756.

[5] M. S. Towhid and N. Shahriar, “Early Detection of Intrusion in SDN,” in *Proceedings of IEEE/IFIP Network Operations and Management Symposium 2023, NOMS 2023*, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/NOMS56928.2023.10154272.

[6] A. Almazyad, L. Halman, and A. Alsaeed, “Probe Attack Detection Using an Improved Intrusion Detection System,” *Computers, Materials and Continua*, vol. 74, no. 3, pp. 479–4784, 2023, doi: 10.32604/cmc.2023.033382.

[7] A. Althobiti, R. Almohayawi, and O. Bamsag, “Machine learning approach to secure software defined network: Machine learning and artificial intelligence,” in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Nov. 2020. doi: 10.1145/3440749.3442597.

[8] A. Abubakar and B. Pranggono, “Machine learning based intrusion detection system for software defined networks,” in *2017 Seventh International Conference on Emerging Security Technologies (EST)*, 2017, pp. 138–143. doi: 10.1109/EST.2017.8090413.

[9] K. B. V., N. D. G., and P. S. Hiremath, “Detection of DDoS Attacks in Software Defined Networks,” in *2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS)*, 2018, pp. 265–270. doi: 10.1109/CSITSS.2018.8768551.

[10] M. W. Nadeem, H. G. Goh, V. Ponnusamy, and Y. Aun, “Ddos detection in sdn usingmachine learning techniques,” *Computers, Materials and Continua*, vol. 71, no. 1, pp. 771–789, 2022, doi: 10.32604/cmc.2022.021669.

[11] J. Wang and L. Wang, “SDN-Defend: A Lightweight Online Attack Detection and Mitigation System for DDoS Attacks in SDN,” *Sensors*, vol. 22, no. 21, Nov. 2022, doi: 10.3390/s22218287.

[12] V. Hnamte and J. Hussain, “An efficient DDoS attack detection mechanism in SDN environment,” *International Journal of Information Technology (Singapore)*, vol. 15, no. 5, pp. 2623–2636, Jun. 2023, doi: 10.1007/s41870-023-01332-5.

[13] S. Wang *et al.*, “Detecting flooding DDoS attacks in software defined networks using supervised learning techniques,” *Engineering Science and Technology, an International Journal*, vol. 35, Nov. 2022, doi: 10.1016/j.jestch.2022.101176.

[14] H. Y. Ibrahim, P. M. Ismael, A. A. Albabawat, and A. B. Al-Khalil, “A Secure Mechanism to Prevent ARP Spoofing and ARP Broadcasting in SDN,” in *2020 International Conference on Computer Science and Software Engineering (CSASE)*, 2020, pp. 13–19. doi: 10.1109/CSASE48920.2020.9142092.

[15] Q.-V. Dang, “Intrusion Detection in Software-Defined Networks,” in *Future Data and Security Engineering*, J. and C. T. M. and T. M. Dang Tran Khanh and Küng, Ed., Cham: Springer International Publishing, 2021, pp. 356–371.

[16] A. Mzibri, R. Benaini, and M. Ben Mamoun, “Case Study on the Performance of ML-Based Network Intrusion Detection Systems in SDN,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Science and Business Media Deutschland GmbH, 2023, pp. 90–95. doi: 10.1007/978-3-031-37765-5_7.

[17] N. Mazhar, R. Salleh, M. A. Hossain, and M. Zeeshan, “SDN based Intrusion Detection and Prevention Systems using Manufacturer Usage Description: A Survey,” 2020. [Online]. Available: www.ijacsa.thesai.org

[18] M. Said Elsayed, N. A. Le-Khac, S. Dev, and A. D. Jurcut, “Network Anomaly Detection Using LSTM Based Autoencoder,” in *Q2SWinet 2020 - Proceedings of the 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, Association for Computing Machinery, Inc, Nov. 2020, pp. 37–45. doi: 10.1145/3416013.3426457.

[19] 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, Oct. 2021, doi: 10.1016/j.jnca.2021.103160.

[20] H. M. Chuang, F. Liu, and C. H. Tsai, “Early Detection of Abnormal Attacks in Software-Defined Networking Using Machine Learning Approaches,” *Symmetry (Basel)*, vol. 14, no. 6, Jun. 2022, doi: 10.3390/sym14061178.

[21] E. M. Zeleke, H. M. Melaku, and F. G. Mengistu, “Efficient Intrusion Detection System for SDN Orchestrated Internet of Things,” *Journal of Computer Networks and Communications*, vol. 2021, 2021, doi: 10.1155/2021/5593214.

[22] M. S. Elsayed, N. A. Le-Khac, and A. D. Jurcut, “InSDN: A novel SDN intrusion dataset,” *IEEE Access*, vol. 8, pp. 165263–165284, 2020, doi: 10.1109/ACCESS.2020.3022633.

[23] N. Abbas, Y. Nasser, M. Shehab, and S. Sharafeddine, “Attack-Specific Feature Selection for Anomaly Detection in Software-Defined Networks,” in *2021 3rd IEEE Middle East and North Africa COMMunications Conference, MENACOMM 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 142–146. doi: 10.1109/MENACOMM50742.2021.9678279.

[24] Kurniabudi, D. Stiawan, Darmawijoyo, M. Y. Bin Idris, A. M. Bamhdi, and R. Budiarto, “CICIDS-2017 Dataset Feature Analysis With Information Gain for Anomaly Detection,” *IEEE Access*, vol. 8, pp. 132911–132921, 2020, doi: 10.1109/ACCESS.2020.3009843.

[25] D. Kreutz, F. M. V Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-Defined Networking: A Comprehensive Survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015, doi: 10.1109/JPROC.2014.2371999.

[26] H. KHALID, P. ISMAEL, and A. AL-KHALIL, “EFFICIENT MECHANISM FOR SECURING SOFTWARE DEFINED NETWORK AGAINST ARP SPOOFING ATTACK,” *The Journal of the University of Duhok*, vol. 22, no. 1, pp. 124–131, Nov. 2019, doi: 10.26682/sjuod.2019.22.1.14.

[27] T. A. Tang, D. McLernon, L. Mhamdi, S. A. R. Zaidi, and M. Ghogho, “Intrusion detection in sdn-based networks: Deep recurrent neural network approach,” in *Advanced Sciences and Technologies for Security Applications*, Springer, 2019, pp. 175–195. doi: 10.1007/978-3-030-13057-2_8.

[28] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, “Deep learning approach for Network Intrusion Detection in Software Defined Networking,” in *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, 2016, pp. 258–263. doi: 10.1109/WINCOM.2016.7777224.