Bringing to Light: Adversarial Poisoning Detection for ML-Based IDS in Software-Defined Networks

Tapadhir Das, *Member, IEEE*, Raj Mani Shukla, Suman Rath, *Graduate Student Member, IEEE*, and Shamik Sengupta, *Senior Member, IEEE*

***Abstract*—Machine learning (ML)-based network intrusion detection systems (NIDS) have become a prospective approach to efficiently protect network communications. However, ML models can be exploited by adversarial poisonings, like Random Label Manipulation (RLM), which can compromise multi-controller software-defined network (MSDN) operations. In this paper, we develop the Trans-controller Adversarial Perturbation Detection (TAPD) framework for NIDS for MSDNs. The detection framework takes advantage of the MSDN architecture and focuses on periodic transference of ML-based NIDS models across the SDN controllers in the topology, and validates the models using local datasets to calculate error rates. We demonstrate the efficacy of this framework in detecting RLM attacks in an MSDN setup. Results indicate efficient detection performance by the TAPD framework in determining the presence of RLM attacks and the localization of the compromised controllers. We find that the framework works well even when there is a significant number of compromised agents. However, the performance begins to deteriorate when more than 40% of the SDN controllers have become compromised.**

***Index Terms*—Adversarial attacks, adversarial detection, data poisoning, network intrusion detection, software-defined networks, machine learning.**

I. Introduction

To ENSURE more efficient capabilities, software-defined networks (SDN) have become an essential part of modern computer communications. SDNs decouple the control and data planes through a logically centralized SDN controller that regulates and supervises the entire network of switches. The SDN controller injects flow tables within the network switches, therefore, helping to administer network operations [1]. Additionally, this integrated architecture makes tasks like data mining and anomaly detection for improved operations and security possible [2]. SDNs have become prevalent in various industrial settings and the global SDN market is expected to reach $35.6 billion by 2026 [3]. To keep up with networking demands, multi-controller SDNs (MSDNs) have emerged as an extension of traditional SDNs. This architecture helps alleviate the flow processing pressure faced by single controllers as it helps provide a greater capacity to deal with the increasing number of flow requests within the SDN and ensures that switches appropriately route new incoming packets in case a controller fails in the infrastructure [4].

Received 13 August 2023; revised 6 December 2023; accepted 15 December 2024. Date of publication 18 December 2024; date of current version 24 February 2025. This work was supported by the National Science Foundation under Grant 2019164. Recommended for acceptance by Dr. Dusit Niyato. *(Corresponding author: Tapadhir Das.)*

Tapadhir Das is with the Department of Computer Science, University of the Pacific, Stockton, CA 95211 USA (e-mail: tdas@pacific.edu).

Raj Mani Shukla is with the School of Computing and Information Science, Anglia Ruskin University, CB1 1PT Cambridge, U.K. (e-mail: raj.shukla@aru.ac.uk).

Suman Rath and Shamik Sengupta are with the Department of Computer Science and Engineering, University of Nevada, Reno, NV 89557 USA (e-mail: sumanr@unr.edu; ssengupta@unr.edu).

Digital Object Identifier 10.1109/TNSE.2024.3519515

As the usage of SDNs continues to increase, the security of these networking infrastructures becomes crucial. Network intrusions can adversely affect all layers of the SDN architecture and can deteriorate the network's availability, authority, confidentiality, and integrity. These are arduous to detect as they display similar traffic patterns to that of normal network functionality [1]. These threats have led to a rise in the generation of network intrusion detection systems (NIDS). NIDS protects SDNs by continually monitoring the network traffic for anomalous patterns indicative of an intrusion [5].

With improved computational resources, the use of machine learning (ML) has become an interesting technique for contemporary NIDS. This is due to ML's ability to detect correlations in network traffic patterns, therefore making them attractive for performing automated differentiation between benign and malevolent network circumstances [6], [7], [8]. Unfortunately, the increased usage of ML-based NIDS has also led to a rise in cyber attacks that aim to compromise the functionality of these algorithms [9], [10]. Adversarial attacks aim to corrupt ML operations by performing manipulations on the algorithm. By exploiting training data and model parameters and sensitivities, these attacks can affect the performance of the ML classifiers, putting the entire MSDN infrastructure at risk [11], [12].

A prominent subset of adversarial attacks is called poisoning attacks. These attacks tamper with or "poison" a target ML model by maliciously manipulating the model parameters or data, forcing the model to make incorrect predictions, and affecting system performance by triggering misclassifications [13]. Contemporary ways to perform them include logic corruption, data injection, and label manipulation (LM). LM attacks are poisoning attacks that adversarially perturb a subset of training labels to decrease the performance of a trained classifier [14]. An illustration of an LM attack on an ML pipeline can be seen in Fig. 1. As illustrated, an adversary maliciously alters a subset of training labels to ensure the model is trained incorrectly and provides sub-optimal performance when being evaluated against testing or real-time data. An adversary can be motivated

2327-4697 © 2024 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Mepco Schlenk Engineering College. Downloaded on August 05,2025 at 08:56:29 UTC from IEEE Xplore. Restrictions apply.

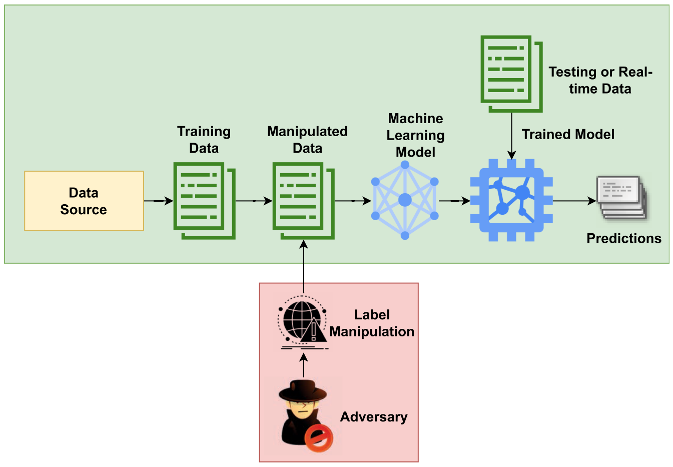

Fig. 1. Label manipulation attack on ML pipeline.

to perform LM attacks if they wish to disguise certain attack categories as benign network traffic to evade detection by the NIDS. Additionally, they can also aim to alter one attack category from another to trigger incorrect responses from the SDN administrators [15]. LM attacks can be detrimental to ML-based NIDS, as they can gradually shift or re-position the decision boundaries of the NIDS through the flipped labels. Due to this, data samples that should be classified correctly get incorrectly predicted. This can put the MSDN at risk [16]. Additionally, the effects of LM attacks on MSDNs can be exacerbated if the MSDN is an essential part of critical national infrastructures like the power grid and transportation systems.

Due to their imminent risk on ML pipelines, defensive tactics to protect ML-based NIDS for MSDNs must be developed. However, there has been limited research in this field. In this paper, we propose a novel adversarial LM detection mechanism for ML-based NIDS in MSDN setups, called Trans-controller Adversarial Perturbation Detection (TAPD). The proposed technique involves periodically transferring NIDS models across all the SDN controllers in the MSDN setup. During these transfers, the trained NIDS models are validated using the local dataset of each SDN setup to calculate the error in their predicted values and ground truth. Once all models have been parsed through all SDN controllers, each controller analyses all local errors using robust statistics to detect any anomalous distribution values. These can be indicative of potential LM or "poisoning" attacks on the NIDS for the controllers. Each controller votes on their suspected SDN controllers, and in the end, the comprehensive list of suspect SDN controllers is accumulated. For experimentation, we apply the proposed TAPD method to detect Random Label Manipulation (RLM) attacks on an open-sourced network intrusion detection dataset called UNR-IDD [17] which is a relevant NIDS dataset for our experiments. The main contributions of our proposed work include:

- Proposing a novel LM attack detection framework called TAPD to LM attacks like RLM in the context of MSDNs. This technique evaluates the performance of each ML-based NIDS on every SDN controller by conducting periodic model transfers between the multiple controllers. The main function is to analyze the performance of the ML model using the local data of each SDN, thereby creating a decentralized framework to evaluate the functionality of the ML-based NIDS.
- Performing a computational and resource requirement analysis to evaluate the feasibility of the proposed framework in practical MSDN setups. This analysis was performed by computing the computational and resource costs associated with ML-based NIDS in SDN and MSDNs, denoting the communication cost of the network protocols, and lastly, denoting the algorithmic and final cost of the TAPD framework.
- Evaluating the impact of the proposed solution on RLM attacks upon the open-sourced UNR-IDD dataset. This is conducted by observing the effect of the proposed TAPD framework on reducing the level of performance degradation experienced by the ML-based NIDS from RLM attacks. The performance metrics utilized for this evaluation are Accuracy, mean Precision, mean Recall, and mean F Measure scores.
- Studying the impact of the proposed solution upon varying parameters under RLM attacks, including number of infected samples in the dataset, the number of compromised SDN controllers, and the detection scale of the proposed TAPD framework. Lastly, we also showcase the impact of the framework on the voting results when the number of compromised SDN controllers is varied.

The rest of the paper is structured as follows: Section II provides the literature study. Multi-controller SDN (MSDN) setup and the system model for our research are presented in Section III, while the proposed methodology is illustrated in Section IV. Section V presents experimental results and discussions. Finally, conclusions are drawn in Section VI.

II. Related Work

Like many adversarial attacks, LM attacks have also become a point of compromise for many ML pipelines [18], [19] [20]. The authors in [18] proposed an optimization framework to perform LM attacks that aim to maximize the classification error of a supervised classifier. In [19], the authors proposed an effective attack model LafAK based on approximated closed form of graph neural networks and continuous surrogate of non-differentiable objective, creating attacks using gradient-based optimizers. The work in [20] proposes multiple LM attacks that aim to compromise the performance of Naive Bayes classifiers used on spam filtering systems. LM attacks in NIDS have also received limited attention [21], [22]. [21] proposed a targeted LM attack that aimed to flip between 0% and 50% of training labels using the Clever Hans library [23]. In [22], the authors performed LM attacks on two ML-based NIDS to evaluate their performance.

Defending against adversarial attacks like LM attacks is a non-trivial task as these attacks tend to be computationally complex, multivariate, non-linear, and non-convex in nature. The current methods that do exist can still be circumvented by adversaries as most solutions are focused on specific algorithmic

Authorized licensed use limited to: Mepco Schlenk Engineering College. Downloaded on August 05,2025 at 08:56:29 UTC from IEEE Xplore. Restrictions apply.

categories and applications. Certain techniques like adversarial training [24], [25] can prove to be ineffective against LM attacks due to the unknown nature of the perturbations needed to perform the training procedure. This technique can also be bypassed using two-step attacks as seen in [26]. Techniques like gradient hiding [26], [27] cater to gradient-based approaches like neural networks, while feature squeezing [28] can also prove to be ineffectual as these methods reduce the feature surface but will be unsuccessful against LM attacks. Gradient hiding also can be circumvented by learning a surrogate black box that contains visible gradient information and crafting examples using that [29]. Lastly, many of these defensive solutions can be ineffectual against black-box attacks that aim to manipulate the operational capability of these ML models [30]. So, to the best of our knowledge, there is no standard technique to protect against adversarial poisoning like LM.

Research in protection for NIDS against LM attacks is very limited and still, a new research area [31], [32]. In [31], the authors propose a novel detection technique called AWFC which detects flipped labels by identifying the difference of classes in the data. This method can be operationally expensive as the calculation of fully connected layer weights can be costly if a dataset has many features or classes. The work in [32] proposes SecFedNIDS, a novel LM attack detection technique using classpath similarity. The limitation of this work lies in the fact that the authors use the Jaccard Index to compute the similarity between class paths. The Jaccard Index is insensitive to the size of the set of similar items in two classes, which can be an issue as two items with a varying number of similar items between can have the same similarity score [33], [34].

This motivates the need for an LM attack detection mechanism for NIDS that does not depend on a machine learning-based or statistical mechanism that can be circumvented and resource exhaustive but utilizes the network architecture to its advantage. The proposed approach achieves that by utilizing the SDN controllers in the MSDN setting to jointly validate and vote on their observed malicious SDN controllers that may have been compromised by LM attacks.

III. System Model

A. Multi-Controller SDN Architecture

We assume an industrial MSDN setup, consisting of multiple SDN topologies, SDN controllers, and a single Command Center. Depending on the application, this architecture could span distributed geographical locations. An illustration of our MSDN setup is provided in Fig. 2, and consists of the following components:

- *Command Center:* This level contains essential SDN applications that actively communicate with the SDN controllers for operations like network management, automation, and analytics. These applications perform by leveraging crucial network metrics like topology, state, and statistics. The Command Center is essential as network administrators oversee network operations and ensure secure functionality from this point.
- *Control Plane:* This level contains the amalgamation of SDN controllers, each regulating its SDN topology. The SDN controllers regulate the Data Plane devices and administer network policies for the functionality of each governed SDN topology.
- *Data Plane:* This level carries the physical SDN infrastructure devices like switches, bridges, routers, and repeaters. Here, each physical topology communicates with its respective SDN controller in the Control Plane.
- *Southbound Interface:* This interface facilitates communication between the Control Plane and the Data Plane, and is typically conducted using southbound protocols like OpenFlow, Netconf, and Ovsdb.
- *Northbound Interface:* This interface facilitates communication between the Control Plane and the Command Center, and is typically conducted using northbound protocols like REST APIs.
- *East-West Interface:* This interface facilitates communication between the SDN controllers by allowing the transference of data and parameters to ensure transparency and synchronization. This is typically conducted using east-west protocols like BGP or OSPF.

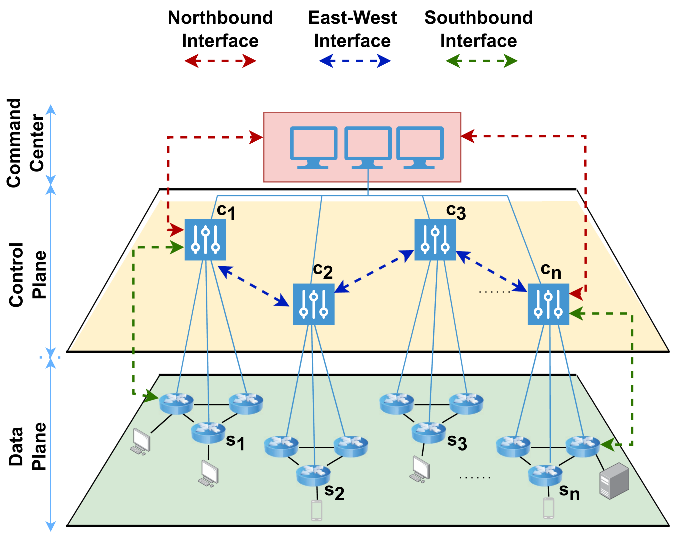

Fig. 2. Multi-controller SDN setup.

B. ML-Based NIDS Pipeline

In ML-based NIDS for MSDN architectures, each ML classifier is locally trained with collected training data. This training data is sourced from network sensors, metrics, and statistics from hosts, routers, and switches. According to the National Institute of Standards and Technology, adversarial attacks can occur within ML pipelines at one of four major Targets of Attacks (TAs) [35]. The four main TAs are:

- *Input:* This TA contains network metrics, states, and statistics from multiple devices like routers, switches, computers, mobile devices, vehicles, and sensors. Prospective attacks include malicious tampering with the collected data that is to be fed into the ML pipeline.
- *Data Pre-processing:* This TA includes techniques for data preparation before being fed into the ML model,

Authorized licensed use limited to: Mepco Schlenk Engineering College. Downloaded on August 05,2025 at 08:56:29 UTC from IEEE Xplore. Restrictions apply.

including noise processing, feature extraction, and sampling, which occur within the SDN controller. Here, attackers can manipulate collected data sets that are being pre-processed and maliciously alter them so that tampered data is presented to the machine learning model for training.

- *ML model:* The primary TA within the pipeline is the ML model itself. Any algorithm used for network intrusion detection, like neural networks, decision trees, random forests, multi-layer perceptrons, and reinforcement learning, can fall victim to this circumstance. Attackers can poison data or labels that are being processed or create generative adversarial data samples to deceive these ML algorithms. Attacks on ML training data, specifically, can result in incorrect training of the ML model, which can hamper system performance. Additionally, it is in this TA where adversaries can be creative in how they attack the ML model as every adversarial attack is designed to compromise a specific type of ML algorithm. For instance, attacks like LM can impact all supervised learning algorithms. However, they are most lethal against divide-and-rule algorithms like decision trees and random forests [15]. Similarly, attacks like the Fast Gradient Sign Method (FGSM) are geared towards compromising gradient-based algorithms like neural networks and multi-layer perceptrons [36]. Finally, attacks like reward poisoning are directed at compromising reinforcement learning-based systems [37].
- *Output:* This TA includes methods to output the predictions, classifications, and forecastings of the ML model. The primary function of this location is to perform network intrusion detection and provide system state, status, and anomalies to the system administrators from outputs received from the SDN controllers. Potential attacks can involve malicious tampering of the output sensor or display data that are interpretable to the remainder of the system and the system operators.

An illustration of the four main TAs in an ML-based NIDS pipeline is provided in Fig. 3.

For our system model, we present the following assumption:

Assumption 1: SDN setup: Within the real-world context, every physical SDN topology in an MSDN setup can vary depending on the number of users, nodes, devices, and geographical spread of the network. In certain cases, there may be shared switches between various topologies that are under the control of multiple SDN controllers. However, for simplicity purposes, in this work, we assume that all the physical topologies are the same with the same number of parameters, meaning that the data is Independent and Identically Distributed (I.I.D). We also assume that each SDN topology is administered by a unique SDN controller situated in the control plane.

IV. Methodology

This section describes the methodology for our TAPD framework. Table I defines the important parameters that are utilized in our mathematical formulations for the framework.

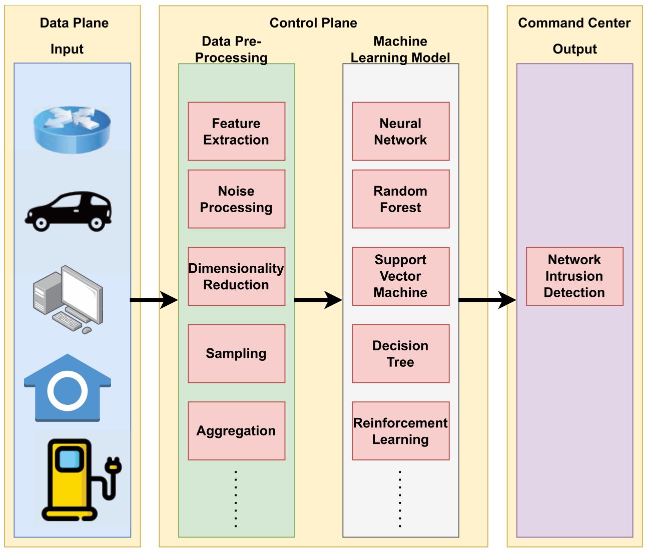

Fig. 3. Four targets of attack in ML-based NIDS.

TABLE I
Important Parameter Definitions

Symbol	Definition
S	Set of SDN Topologies
C	Set of SDN Controllers
N	Number of SDN Topologies and Controllers
N'	Number of Malicious SDN Controllers
X	Number of Data Samples in Target Dataset
Y	Number of Labels in X
E	Number of Data points in X and Y
F	Set of Number of Unique Labels in X
L	Number of Unique Labels in F
Θ	Attack Control Parameter
Φ	Set of untrained ML-based NIDS
Δ	Set of trained ML-based NIDS
η	Detection Scale

A. Raw Topology and NIDS Setup

We present an MSDN topology that contains a set of separate SDN topologies, denoted by S, where $S = \{S_1, S_2, \ldots S_N\}$. Correspondingly, we introduce a set of SDN controllers for every SDN topology, denoted by C, where $C = \{C_1, C_2, \ldots\ldots C_N\}$. The number of SDN topologies in the MSDN setup is assumed to be N. Next, we assume the subset of compromised SDN controllers that have been affected by LM attacks to be denoted by N', where $N' \subset N$. Let our target network intrusion detection dataset be X which consists of E total samples. In X, x_i represents a single sample where $x_i \in X, i = \{0, 1 \ldots E\}$. Similarly, let all labels of X be denoted by Y. y_i represents a single label corresponding to x_i and $y_i \in Y, i = \{0, 1 \ldots E\}$. The number of unique labels in Y is denoted with L. Here, l_j represents each unique label in L and $l_j \in L, j = \{0, 1 \ldots F\}$, and F represents the total number of unique labels. Also, we can assume that a

Authorized licensed use limited to: Mepco Schlenk Engineering College. Downloaded on August 05,2025 at 08:56:29 UTC from IEEE Xplore. Restrictions apply.

unique label l_j has M_j number of samples associated with it within X, such that:

$$\sum_{j=0} M_j = E, j = \{0, 1 \ldots F\} \tag{1}$$

B. *Random Label Manipulation*

For our LM attack, we are choosing the RLM attack as our threat. In this attack category, a subset or all of the training dataset's labels get randomly manipulated and altered. This attack scenario was chosen as our threat vector due to its current relevance in studying adversarial attacks in literature [19], [38]. It should be noted that the proposed approach is being tested against RLM attacks. However, our technique can also be adapted to detect other adversarial poisoning attacks like other kinds of LM, feature poisoning, data manipulation, and data injection.

To initiate an RLM attack, we must first decide the number of labels to maliciously alter. For this, we denote Θ as our *Attack Control* parameter, where $0 \leq \Theta \leq 1$. Θ represents the proportion of samples from X to alter and is selected by the adversary based upon their specific agenda. A low value of Θ means that the adversary is trying to alter as minimal labels as possible, thereby not incurring too much damage on the SDN, and staying more undetectable. Similarly, a high value of Θ means that the adversary is trying to alter as many labels as possible, thereby incurring more damage on the SDN, and not prioritizing undetectability. Using Θ, the adversary must select the number of samples that they will maliciously alter. This can be done using (2):

$$V = ceil(\Theta \times E) \tag{2}$$

where V represents the total number of samples that will be altered by the adversary. Therefore, the adversary can randomly choose V samples from the dataset and randomly change each of their labels.

All SDNs train their ML models for network intrusion detection. However, a subset of these models trains incorrectly as they have been exposed to the RLM attack. We denote the set of ML models Φ, where $\Phi = \{\Phi_1, \Phi_2, \ldots \Phi_N\}$. Each SDN controller contains a single ML model. Once the ML models become trained, then they become trained NIDS, denoted by Δ, where $\Delta = \{\Delta_1, \Delta_2, \ldots \Delta_N\}$. Finally, as a subset of these trained NIDS has been compromised by RLM attacks, we denote them as Δ', where Δ' contains the set of all compromised trained ML-based NIDS.

C. *Trans-Controller Adversarial Perturbation Detection (TAPD)*

When ML-based NIDS has been compromised with poisoning attacks like RLM, they can degrade the performance and functionality of both the local SDN environment and the whole MSDN infrastructure. To combat the effect of poisoning attacks, it would be beneficial to periodically validate the efficiency of the ML-based NIDS across the multiple controllers in the MSDN. This is the approach undertaken in our proposed framework, TAPD, to detect RLM attacks when they occur in MSDN setups. The TAPD framework can be broken into multiple stages, illustrated in Fig. 4. The TAPD operation begins when the Command Center sends a request out to the Control plane and correspondingly receives the list of voted malicious SDN controllers from all SDN controllers after a predetermined time interval. The entire operation ends when the Command Center can identify any prospective compromised SDN controllers from the list provided by the Control plane. The predetermined time interval is decided upon by the system administrators by leveraging the network operations and the overhead and latency costs that can occur from running the TAPD framework.

1) Send Request: In this stage, the Command Center sends a request to the Control plane to begin the TAPD framework functionality. This can be done using the Northbound interface which facilitates communication between the Control plane and the Command Center. These messages can be typically sent using northbound protocols like REST APIs. They can be sent to any SDN controller in the Control plane, keeping the system de-centralized. De-centralization prevents the message from being intercepted in the case the message is received by a malicious SDN controller. This stage can be event-triggered if an event causes suspicion to the network administrator that the ML-based NIDS has been compromised, or periodic, to ensure optimal functionality in the network at all times. Instances for events that can cause suspicion include if network administrators notice an anomalous trend with network traffic in the SDN, like unexpectedly high congestion stemming from a Denial-of-Service attack that can be felt by users, but the traffic conditions are not triggering the ML-based NIDS to classify the network as undergoing abnormal behavior, indicating a disruption with the ML-based NIDS algorithm. If the request is sent to a malicious SDN controller, it may be intercepted and the entire operation may not commence. At that time, the Command Center waits for the predetermined time interval for the reception of the list of voted malicious SDN controllers from all SDN controllers in the Control plane. If that list does not show up, the operation will begin again, with a different SDN controller being sent the request.

2) ML Model Transfer: Once an SDN controller receives this request to begin the TAPD algorithm, it sends a message to all other SDN controllers to begin their model transfers. This message is transmitted using non-standardized interfaces referred to as the east-west protocol which can facilitate communication between SDN controllers [39]. East-west protocols are newer communication frameworks that are becoming imminent [40], [41] [42]. These use a notification system or distributed routing protocol like Border Gateway Protocol or Open Shortest Path First. When all SDN controllers receive this message, they begin the model transfer stage.

The first step for each SDN controller is to generate a duplicate ML model for network intrusion detection. The operations of the normal model will be transferred to this duplicate model. Then, the transfer begins. We can assume the source SDN controller C_{Sr} contains the ML-based NIDS Δ_{Sr}. The destination SDN

Authorized licensed use limited to: Mepco Schlenk Engineering College. Downloaded on August 05,2025 at 08:56:29 UTC from IEEE Xplore. Restrictions apply.

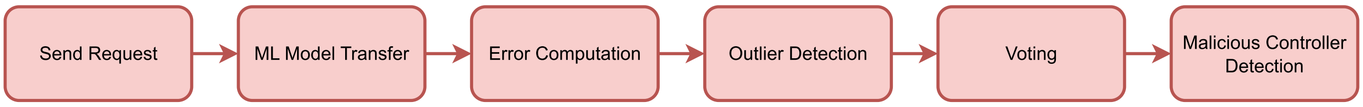

Fig. 4. TAPD framework stages.

controller for Δ_{Sr} is denoted as C_D. In C_D, Δ_{Sr} will undergo localized error checking and processing and then will be transferred to the next SDN controller. Once the target ML model Δ_{Sr} has passed through the processing of all the SDN controllers in the Control plane, it returns to the source SDN controller C_{Sr}.

3) Error Computation: When the target model Δ_{Sr} arrives at the destination SDN controller C_D, it must undergo local error checking and processing. The first is the prediction the local training data of C_D, X_d, through Δ_S using (3):

$$\Delta_{Sr_{Pred_d}} = predict(\Delta_{Sr}, X_d) \tag{3}$$

where, $\Delta_{Sr_{Pred_d}}$ represents the predictions of Δ_{Sr} for the local training dataset X_d. Next, the error of the predictions must be computed, in comparison to the ground-truth values of the local training labels, denoted by Y_d. We assume the number of samples in X_d and Y_d is denoted as J. Thus, the error computation is performed using (4):

$$\Gamma_{Sr} = \frac{1}{j}\sum_{i=1}^{J}(Y_{d_i} - \Delta_{Sr_{Pred_{d_i}}})^2 \tag{4}$$

where Γ_{Sr} represents the total computational error achieved by $\Delta_{Sr_{Pred_d}}$ in comparison to Y_d. Following the error computation, the target ML model Δ_{Sr} is transferred over to the next destination SDN controller for further analysis.

4) Outlier Detection: The most important step in the proposed framework for detecting the ML models that have fallen victim to RLM attacks is outlier detection. Once an ML model Δ_{Sr} has been parsed through all the SDN controllers in the Control plane, it returns to its source SDN environment C_{Sr}. At this moment, each SDN controller has a set of errors that have been computed for all the ML models from all the SDN controllers, including their own. This set can be denoted as F_{Sr}, which represents the set of errors computed by the source SDN controller on all the ML models in the Control plane that have passed through it. Now, the SDN controller must perform outlier detection to detect any ML models that have given anomalous error values, compared to the rest of the models. The intuition is that the ML models that have been trained using adversarially perturbed data will be noticeable as abnormal.

For our outlier detection, we are using the Inter-Quartile Range (IQR). This metric is used as it is not influenced by extreme values in distribution due to its usage of median to find the midpoint in the distribution and can be used as a measure of variability if the extreme values are not being recorded exactly as is [43]. To begin the outlier detection, we must compute the quartiles of the distribution present in F_{Sr}. Next, we must find the *25th* quartile, which can be done using (5):

$$\chi_{25} = (1/4) \times (N+1)th\,term \tag{5}$$

where χ_{25} represents the *25th* percentile of the distribution F_{Sr}. Similarly, we also find the *75th* quartile, which can be done using (6):

$$\chi_{75} = (3/4) \times (N+1)th\,term \tag{6}$$

where χ_{75} represents the *75th* percentile of the distribution F_{Sr}. We are utilizing the *25th* and *75th* percentiles as IQR represents the middle 50% of our distribution. This is essential to compute any existing outliers in the distribution by isolating the middle 50% which is emblematic of the normal values in the distribution [44].

Finally, we achieve the IQR of the distribution, denoted by χ, by the following (7).

$$\chi = \chi_{75} - \chi_{25} \tag{7}$$

The value of χ serves as an important magnitude for our outlier detection. We can assume that low error values in F_{Sr} correlate to local ML models for network intrusion detection. Meaning, that the lowest computed error signifies the local ML model for network intrusion detection. Correspondingly, we can also assume that the highest magnitudes of error are associated with ML models for network intrusion detection that have undergone adversarial perturbation training by RLM attacks. We are mostly interested in detecting these high magnitudes across all the SDN controllers.

Our computed threshold for outlier detection, ω, is done using (8):

$$\omega = \chi_{75} + \chi \times \eta \tag{8}$$

Here, η represents the *detection scale*, where $0 \leq \eta \leq 1$. Low values of η provide finer detection if the variance of the F_{Sr} is low, and the variance of the entire distribution of F_{Sr} is very close to the variance of the normal values in F_{Sr}. High values provide more flexibility to the outlier detection algorithm if the variance of the F_{Sr} is high, and the variance of the entire distribution of F_{Sr} is very close to the variance of the normal values in F_{Sr}.

5) Voting: The penultimate step in the TAPD approach, and the final step in the SDN controller, is to use the distribution threshold for outlier detection, χ, to detect the outlier error values and the associated SDN controllers. This can be performed using (9):

$$\lambda_s = \begin{cases} Compromised, & \text{if } F_{s_i} > \omega, \\ Safe, & \text{if } F_{s_i} \leq \omega, \end{cases} \tag{9}$$

Here, λ_s represents the list of compromised SDN controllers, according to the source SDN controller C_{Sr}, and is C_{Sr}'s vote as to which controllers are compromised. This list is then sent to the Command Center where the final analysis occurs to detect

Authorized licensed use limited to: Mepco Schlenk Engineering College. Downloaded on August 05,2025 at 08:56:29 UTC from IEEE Xplore. Restrictions apply.

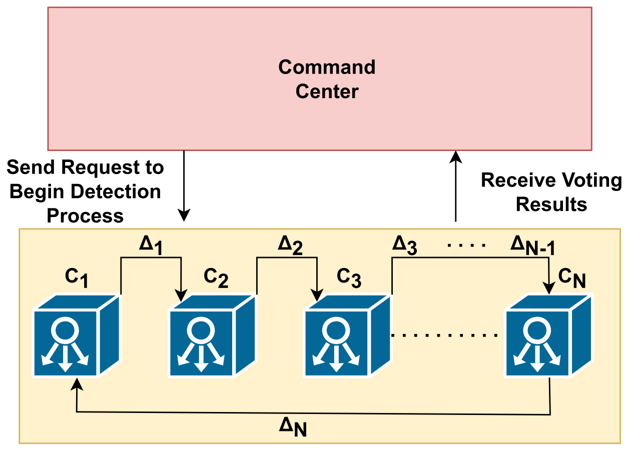

Fig. 5. Overview for the TAPD framework.

the compromised SDN controllers, based on the voting results of all N SDN controllers. $\lambda_s \in \lambda$ represents the final list of all the compromised controllers voted by all the controllers in the Control plane.

Fig. 5 illustrates an overview of the TAPD framework mechanism.

6) Malicious Controller Detection: The final step of the TAPD framework is malicious controller detection, which is performed in the Command Center. This is performed here to let the system administrators know which controllers have been subjected to the RLM or other adversarial poisoning. This is performed by conducting a frequency analysis of the voting results provided by the control plane, λ. This is illustrated using (10):

$$\zeta = \sum_{i=0}^{Z} \Lambda_j \,|j \in L \tag{10}$$

where ζ denotes the final list of compromised controllers, Z represents the total number of elements in λ, and L denotes the unique elements present in λ. Once, the compromised SDN controllers, ζ, are identified, the system administrators can take response measures like separating the SDN topologies from the rest of the MSDN setup or coming up with resiliency mechanisms to make more robust ML models for SDN NIDS that can withstand adversarial attacks like RLM.

For our proposed methodology, we present the following assumptions:

Assumption 2: Point of compromise: As the proposed RLM attacks target the training labels, the adversary needs to have access to this training data collection setup. Therefore, we can assume that the adversary can achieve this malicious access by compromising integral points in the SDN architecture. Compromising high-value targets like the SDN controller can provide complete access to the training labels, while lower-value targets like a data plane switch or end hosts can provide access to partial training labels. Ways an adversary can achieve this can include performing allergy attacks on signature-based intrusion detection systems [45], or even through false data injection/backdoor poisoning [46].

Assumption 3: Adversary possesses potential knowledge of the NIDS: Contemporary methods to perform NIDS primarily use ML-based methods, where training data is collected and trained offline, and a trained model is placed online. An attacker can assume that the target MSDN also uses an ML-based method or discover this practice through reconnaissance techniques network scanning, fingerprinting, enumeration, and traffic sniffing [47], [48]. If they can intrude upon or compromise the SDN infrastructure, then the labels that are being collected as part of network intrusion detection can also be unearthed by the adversary. Other ways an adversary can identify important training labels would be through a black-box attack where the attacker can simply feed data samples through the ML model and map the labels to the inputs.

D. Deployment Requirements

The main hosts for the TAPD algorithm will be the Command Center and the Control Plane. These systems should have adequate operational resources like rapid processing speeds, extensive storage resources, and minimal operational latencies. The application will also be developed to operate without hindering normal MSDN operations. Deployment of the TAPD algorithm will require a main program and multiple sub-programs that will work in a cooperative and coordinated fashion.

1) Main Program: The main program to initiate the Send Request step will be conducted in the Command Center of the MSDN setup. The application will be hosted as an SDN application that is supervised by the network administrators. The application is responsible for initializing the Send Request and waiting for the responses from the Control Plane. Once the responses have been met, this application will conduct the Malicious Controller Detection step of the TAPD framework by parsing through the voted results from all the SDN controllers in the Control Plane.

2) Sub-Program: The sub-programs for the TAPD algorithm will be deployed in all of the SDN controllers that are in the Control Plane. These applications will be responsible for actively listening to the Send Request message from the Control Center. Once received, the application is responsible for starting the ML Model Transfer step, the Error Computation step, the Outlier Detection step, and the Voting step in the local SDN controller. Once voting has concluded in the SDN controller, the application is also responsible for submitting the voting results to the Command Center.

E. Computational and Resource Requirement Analysis

In this section, we are analyzing the computational and resource requirements for deploying the TAPD framework in MSDNs. The primary areas include communication and algorithmic costs. However, since ML-based NIDS is still not substantially prevalent industrial practice, we are also including the cost of ML operations in this analysis, to provide a comprehensive evaluation for deploying this solution in practical MSDN setups.

1) SDN Cost: An ML model must be trained to perform network intrusion detection within the SDN controller. For this, it is necessary to ensure that the SDN controller possesses adequate memory and processing speeds. In this evaluation, we are only

Authorized licensed use limited to: Mepco Schlenk Engineering College. Downloaded on August 05,2025 at 08:56:29 UTC from IEEE Xplore. Restrictions apply.

analyzing supervised learning algorithms as that is what the current version of the TAPD framework is focusing upon. Dataset sizes can vary in these applications as they are dependent on the size, user count, number of devices, and activity in the SDN. The dataset requirement can range from a few megabytes in smaller enterprise networks [17] to hundreds of gigabytes in large-scale Internet of Things environments [49]. For our analysis, let us denote the dataset size cost in the SDN controller as D_{Tr}. ML model sizes for these applications also vary on the type of model being deployed. The memory resource requirements needed to store these models can range from tens of kilobytes to a few gigabytes, depending on the size of the model and the number of parameters used. For our analysis, let us denote the ML algorithm size cost in the SDN controller as M_{Tr}. The computational cost to train the ML models also varies, depending on the type of ML model being used. For instance, lightweight models like decision trees (DT) can have a training computational cost of $O(n \times mlog(m))$ where m corresponds to the number of leaves on the tree and n refers to the number of nodes in the tree [50], while heavier models like neural networks can reach computational costs up to $O(dnlog(n) + nd^2)$ where d symbolizes the depth of the network and n represents the number of evaluation points [51]. For our analysis, let us denote the computational cost of the ML model training in the SDN controller as C_{Tr}.

2) MSDN Cost: Our MSDN setup consists of a set of N SDN topologies and controllers. Here, each SDN has its ML-based NIDS. Depending on the type and intricacy, this ML model may have varying memory requirements and computational costs. They may also possess their dataset, based on the unique network topology, which can vary the dataset size costs. The computational and resource costs for the MSDN can be denoted by:

$$D_N = \sum_{0}^{N-1} D_{Tr} \tag{11}$$

$$M_N = \sum_{0}^{N-1} M_{Tr} \tag{12}$$

$$C_N = \sum_{0}^{N-1} C_{Tr} \tag{13}$$

Here, D_N, M_N, and C_N represent the dataset size, ML model size, and ML computational costs of the MSDN setup, respectively.

3) Communication Cost: The communication costs that occur in the TAPD framework include i) Northbound Interface cost and ii) East-West Interface cost. The Northbound Interface cost is associated with the initial Send Request to the Control Plane and the reception of the voted list of compromised SDN controllers from each SDN controller. The cost to perform these operations is dependent on the medium for the Northbound Interface communication and can be denoted by NB_C. Similarly, the cost of the East-West Interface is associated with the model transfers that occur in the TAPD framework. It should be noted that the costs of operating an East-West Interface will depend on the type and number of SDN controllers in the architecture [52]. We assume the actual cost of transferring

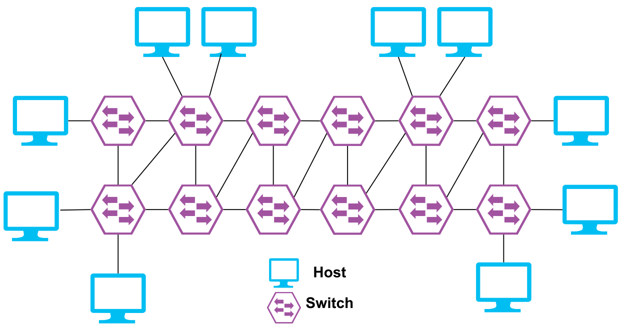

Fig. 6. SDN topology in the MSDN setup.

the ML models between SDN controllers as EW_C. Since each ML model Δ_N is being transferred between all N SDN controllers for validation, the number of transfers required is N^2. Therefore, the total communication cost for the TAPD framework, denoted by C_F, is:

$$C_F = NB_C + \left(EW_C \times N^2\right) \tag{14}$$

4) Algorithmic and Final Cost: The algorithmic costs of performing the TAPD algorithm include i) error computation and ii) outlier detection. The resource cost to perform the error computation ((4)), can be denoted by E_C, while the resource cost to perform outlier detection ((5), (6), and (7)) can be denoted by O_C. The final computational and resource cost F_C for operating the TAPD framework is provided by:

$$F_C = E_C + O_C + C_F + D_N + M_N + C_N \tag{15}$$

V. Experimental Results and Analysis

A. MSDN Setup

For our MSDN setup, we note $N = 10$, and all SDNs have their unique SDN controller. We also assume $N' = 1$ which performs RLM attacks on their respective ML-based NIDS in the SDN controllers. For each SDN setup, we are using the Open Network Operating System (ONOS) as our SDN controller (API version 2.5.0) along with Mininet to simulate the SDN environment. ONOS uses the Open Service Gateway Initiative (OSGi) service component at runtime for the creation and activation of components and for auto-wiring components together. The use of Mininet generates a realistic virtual network, and runs a real kernel, switch, and application code, on a single machine which creates a realistic testbed environment. The generated SDN topology consists of 10 hosts and 12 switches and is illustrated in Fig. 6. To perform the Southbound Interface, we are using the Open Flow (OF) 14 protocol. For the Northbound Interface, we are utilizing REST APIs. As we are running a simulation-based representation of an MSDN on a single machine, an East-West interface was not necessary. However, to conduct this on a physical testbed, protocols like BGP can be leveraged to perform model transfers. All SDN controllers in the MSDN communicate with a centralized Command Center as seen in Fig. 7. In our experiment, we assume that the ML-based NIDS for SDN controller C_9 has been compromised by an RLM attack.

Authorized licensed use limited to: Mepco Schlenk Engineering College. Downloaded on August 05,2025 at 08:56:29 UTC from IEEE Xplore. Restrictions apply.

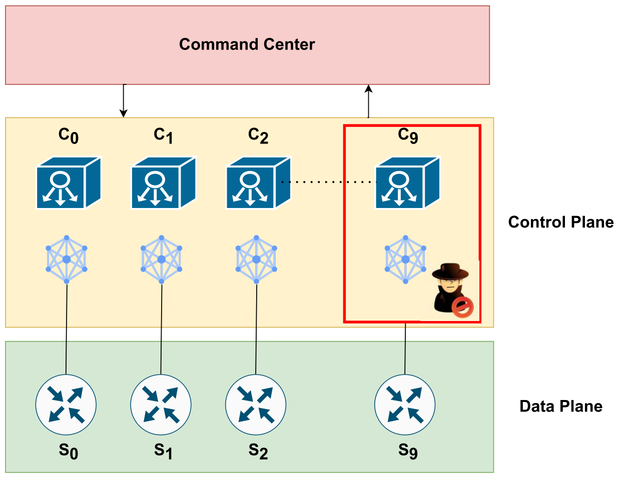

Fig. 7. Experimental setup for RLM attacks.

B. Dataset

For experimentation, we are using the UNR-IDD dataset [17], as it is a relevant NIDS dataset for our experiments. The dataset contains 37,412 data samples and 34 features per sample and possesses multiple attacks that can occur on SDN setups like TCP-SYN, Flow Table Overflow, Traffic Diversion, Blackhole, and Port Scanning. For the features, the dataset consists of port and delta port statistics that are accumulated from the switches within the Data Plane. Out of the 34 features, there are 5 categorical features and 29 numerical features. Categorical features include attributes like switch ID and port number, while numerical features include the number of received and sent bytes and packets. Training and testing are conducted by an 80-20 split, as empirical results in the literature show that this distribution yields the most effective ML models [53].

C. Model Training

To train the ML models in the SDN controllers, we first randomly scramble all the data samples in UNR-IDD. Next, we split the scrambled dataset into N equal sections. Here, each section consists of $\frac{E}{N}$ number of samples and forms an independent dataset that still carries the same distribution of feature magnitudes as all the other sections. Every SDN controller gets a section of data for training the ML-based NIDS. The ML model that we have used for our experimentation is a Random Forest (RF) due to its widespread usage in literature for SDN cybersecurity research [54], [55]. However, any supervised learning ML model can be utilized. We assume that $\Theta = 0.2$, as the magnitude of Θ should be low to avoid instantaneous detection, while still being able to incur a negative impact on the NIDS performance. In addition, our *detection scale*, $\eta = 0.1$, as the total variance of our error is assumed to be low, and the variance of the normal error values is very close to that of the entire distribution, as the majority of our ML models will not be compromised by the adversary. For performance evaluation, we utilize the performance metrics of Accuracy (A), the mean precision score (P_μ), the mean recall score (R_μ), and the mean F-Measure score (F_μ).

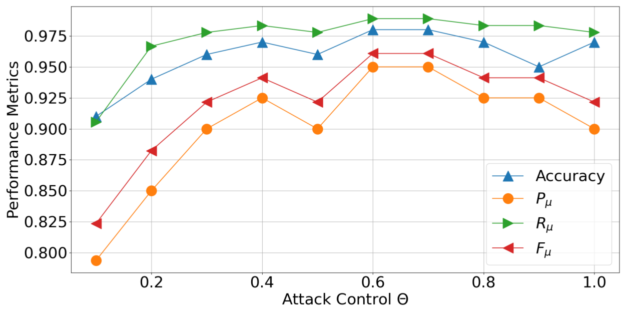

Fig. 8. Performance achieved against varying Θ.

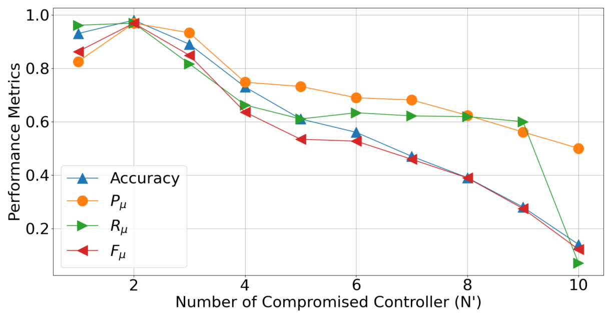

Fig. 9. Performance achieved against varying N'.

D. Experiments

We begin experimentation by observing the performance achieved by the proposed TAPD framework at detecting RLM attacks on ML models as the number of infected samples, Θ, gets varied. This is illustrated in Fig. 8. We note that as the value of Θ increases, the performance achieved by TAPD at detecting the RLM attacks increases across the observed A, P_μ, R_μ, and F_μ metrics. This can be attributed to the fact that at low values of Θ, a low percentage of training samples get randomly manipulated. This can make it difficult for the TAPD framework to detect compromised SDN controllers due to limited manipulation. At higher values of Θ, a high percentage of training samples get randomly manipulated. This type of malicious data handling makes it more evident to the TAPD framework that the SDN controller has been compromised, making it easier to detect.

Next, we observe the performance achieved by the TAPD framework as we vary the number of compromised SDN controllers N', illustrated in Fig. 9. Here, we note that as the number of compromised controllers, N', increases, the performance of the TAPD framework decreases across the observed A, P_μ, R_μ, and F_μ metrics. This can be attributed to the change in collected error metrics on the controller level, which sways which controllers are being voted as compromised. At lower values of N', ($N' = 1, 2, 3$), the number of compromised controllers is proportionately lower than the number of benign controllers. Hence, the size of abnormal computed errors in each controller is proportionately low to the total number of computed errors, making these errors show up outside the outlier detection threshold, χ. This makes it more evident which controllers are compromised, making it easier for the TAPD framework to detect it. However, when the value of N' increases, ($N' = 4, 5, 6 \ldots$), the number

Authorized licensed use limited to: Mepco Schlenk Engineering College. Downloaded on August 05,2025 at 08:56:29 UTC from IEEE Xplore. Restrictions apply.

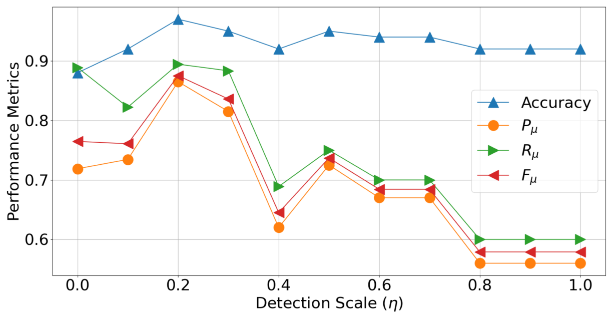

Fig. 10. Performance achieved against varying η.

TABLE II
MALICIOUS CONTROLLER FREQUENCY

	Flagged	**Controller**	**Frequency**							
N’	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	10	-	-	-	-	-	-	-	-	-
0,1	10	10	-	-	-	-	-	-	-	-
0,1,2	9	1	4	-	-	-	-	-	-	-
0,1,2,3	4	3	1	1	-	-	-	-	-	-
0,1,2,3,4	4	2	3	2	2	-	-	-	-	-
0,1,2,3,4,5	4	1	2	2	2	3	-	-	-	-
0,1,2,3,4,5,6	3	1	3	1	-	4	1	-	-	-
0,1,2,3,4,5,6,7	1	1	1	3	1	3	1	2	-	-
0,1,2,3,4,5,6,7,8	3	1	3	1	1	4	1	-	2	-
0,1,2,3,4,5,6,7,8,9	3	1	1	1	1	4	1	1	2	1

of compromised controllers is proportionately equivalent to or higher than the number of benign controllers. This makes it difficult to discern between which of the controllers are benign, and which have been compromised, as the model for normal functionality becomes skewed. The size of abnormal computed errors in each controller is proportionately equivalent or higher to the total number of computed errors. This shifts the value and location of the outlier detection threshold, χ, making it much more difficult to detect the compromised SDN controllers in the MSDN setup. From this experiment, we note that the performance of our proposed framework deteriorates if the number of compromised controllers $N' > 40\%$ in the entire architecture.

Next, we analyze the performance achieved by the TAPD framework when the detection scale, η, is varied, illustrated in Fig. 10. Here, we note that the performance achieved with lower values of η is higher than those that are obtained with higher values of η across the observed A, P_μ, R_μ, and F_μ metrics. This can be attributed to the fact that the variance of the observed errors in each SDN controller is low, and the variance of the entire distribution is proportionately close to the variance of only the normal values in the distribution. This occurs as the adversary is only compromising a single SDN controller out of 10 possible targets, thereby prioritizing being undetectable while incurring damage to the MSDN. This results in higher performance as seen in the metrics observed at low values of η, ($\eta' = 0, 0.1, 0.2, 0.3$). Correspondingly, the performance deteriorates as the values of η increase, ($\eta' = 0.4, 0.5, 0.6, \ldots$). High values provide more flexibility to the outlier detection algorithm if the variance of the observed errors in each SDN controller is high, and the variance of the entire distribution is proportionately close to the variance of only the normal values in the distribution. As the adversary is not compromising a high number of SDN controllers proportional to the total number of SDN controllers, high values of η will not be effective.

Finally, we analyze the SDN controllers that are being flagged by the TAPD framework through the votes that are being transmitted by each SDN controller. This is illustrated in Table II. On the left column, we provide a list of varying scenarios of malicious SDN controllers N' that are affected by RLM attacks. On the other side is the frequency with which each SDN controller is voted by the other controller using the TAPD framework for each scenario. We note that when the value of N' is low, ($N' = \{0, 1, 2, 3\}$), almost all the controllers in the SDN can detect the malicious SDN controllers. This is due to the errors that are being computed with the ML models from these controllers by the other SDNs being outliers. As the value of N' increases, ($N' = \{4, 5 \ldots\}$), we observe that fewer SDN controllers can detect the malicious controllers. This can be attributed to the fact that as the number of malicious controllers has increased, the computed errors from these controllers to the rest of the controllers no longer seem like outliers. This makes it difficult to detect the benign controllers from the ones that are affected by RLM attacks. Similar to the finding in Fig. 9, in this experiment, we note that the performance of our proposed framework deteriorates if the number of compromised controllers $N' > 40\%$ in the entire architecture. This deterioration in performance can be attributed to the present outlier detection mechanism used in the TAPD framework. We are utilizing IQR, a lightweight outlier detection mechanism. Under nominal conditions, where a minority of the total controllers are malicious, it is an effective method to detect the compromised controllers. However, as the total number of malicious controllers begins to approach the majority ($N' > 40\%$), its effectiveness reduces as IQR does not perform well under skewed data conditions [56].

In terms of attack space, compromising $> 40\%$ of the SDN controllers in the architecture is unlikely as adversaries will rarely have this level of access or will wish to allocate resources for this endeavor. For instance, if a similar system is deployed in a distributed microgrid environment like [57] and contains 1000 SDN controllers that are spread over a massive geographical area, it would be impractical for an adversary to compromise more than 400 controllers in this architecture. Additionally, when it comes to attacks like LM, adversaries would be mindful of being undetectable by the network administrators and NIDS. Hence, compromising $> 40\%$ of the SDN controllers would not be a beneficial strategy for the adversary who wishes to incur damage to the system while making it difficult for detection. However, if an adversary does plan to compromise $> 40\%$ of the SDN controllers, the current version of the TAPD framework must be augmented with another outlier detection mechanism. When $N' < 40\%$, we can continue to use IQR as our outlier detection mechanism. When $N' > 40\%$, we can use a more robust outlier detection mechanism that still performs positively under skewed data distributions. Instances of potential outlier detection mechanisms can include median-based techniques like median absolute deviation (MAD). MAD can be a functional outlier

Authorized licensed use limited to: Mepco Schlenk Engineering College. Downloaded on August 05,2025 at 08:56:29 UTC from IEEE Xplore. Restrictions apply.

detection technique for $N' > 40\%$, as it has been shown to perform robustly even under extreme values and non-normality in the data distribution [56].

E. Implementation Challenges and Limitations

Despite the efficient performance that has been achieved in our experiments, certain limitations are associated with the implementation of the proposed TAPD framework:

- *Lack of Scalability:* The current version of the TAPD framework is not a scalable solution. With the continuous rise of network usage, the number of devices, nodes, and users will also increase. The current dataset that is used to train our ML-based NIDS in the TAPD framework will no longer be representative of the new normal behavior of the SDN, thereby requiring continuous re-training of the ML model whenever an addition is made to the MSDN. This further extends to more executions of the TAPD framework to ensure consistency and security. This will lead to more operational overheads and inefficient functionality of the MSDN.
- *Delay between Heterogeneous Controllers:* The current version of the framework proposes the usage of an East-West Interface to facilitate communication between homogeneous SDN controllers in the MSDN. However, limitations can begin arising if we implement the TAPD framework under a heterogeneous SDN controller setup, as each unique SDN controller has their own operational capability and functionality. This can lead to communication delays during the execution of the TAPD framework, especially when transferring models between different SDN controller types. This can also result in increasing operational costs for running the TAPD framework in the MSDN.
- *Protecting against Communication-level Threats:* A final limitation of implementing the TAPD framework is protecting against communication-level threats. Modern MSDN infrastructures can be vast and can span geographically. This increases the threat surface for an attacker to conduct communication-level threats like man-in-the-middle (MITM) attacks. Under these circumstances, attackers can capture inter-controller communication packets and perform malicious manipulations on them before sending them to their intended destination. This can compromise the integrity of the TAPD framework and can put the MSDN at risk.

VI. Conclusion and Future Work

In this paper, we propose a novel LM attack detection framework called Trans-controller Adversarial Perturbation Detection (TAPD) for NIDS in MSDN setups. The detection framework focuses on the periodic transference of ML-based NIDS models across the SDN controllers in the topology, and the validations of these models using the local datasets to calculate errors in their predictions with the ground truth. We provide a computational and resource requirement analysis to evaluate the feasibility of the TAPD framework for practical MSDN setups. We evaluate the efficacy of this framework in detecting RLM attacks and localizing the malicious controllers in the MSDN. We analyze the performance of our framework as we vary multiple parameters like the number of compromised controllers, detection scale, and the number of compromised samples per ML model. Results show efficient detection and performance achieved by the TAPD framework in determining the presence of RLM attacks and the localization of the compromised controllers. We also note that the framework works best when there is a low number of compromised controllers in the topology proportional to the total number of SDN controllers. However, the performance begins to deteriorate when the number of compromised controllers increases. In our experiments, we have noticed the performance progressively drops after 40% of the SDN controllers are compromised. A limitation of the current work is that it is not a scalable solution, which can also be resource-consuming in certain applications. Other limitations of the proposed solution include inefficient performance under heterogeneous MSDN setups and integrity questions if exposed to MITM attacks. However, the observed experimental results showcase significant growth potential for further optimization and enhancements to make it more scalable, operationally inexpensive, and secure from outsider access. Future work in this line of research includes evaluating this technique for detecting other types of adversarial attacks against ML-based NIDS in MSDNs like data manipulation and other optimized LM attacks. Also, we are planning to address the performance deterioration issue when $N' > 40\%$, by augmenting the TAPD framework with more robust outlier detection techniques like MAD that perform robustly under skewed data conditions. Additionally, we are planning to conduct research to make this framework compatible with non-I.I.D data by incorporating practices like rapid reclassification of the datasets in each SDN controller using a proximity-based training method like K-Nearest Neighbors. By doing this, we can ensure that any manipulated labels can be quickly rectified so that they do not degrade ML performance when different data distributions are used by each SDN controller.

References

[1] T. Das, R. M. Shukla, and S. Sengupta, "The devil is in the details: Confident & explainable anomaly detector for software-defined networks," in *Proc. IEEE 20th Int. Symp. Netw. Comput. Appl.*, 2021, pp. 1–5.
[2] T. Das, O. A. Hamdan, S. Sengupta, and E. Arslan, "Flood Control: TCP-SYN flood detection for software-defined networks using openflow port statistics," in *Proc. 2022 IEEE Int. Conf. Cyber Secur. Resilience*, 2022, pp. 1–8.
[3] "The software-defined networking (SDN) market in 2022," Apr. 2022. [Online]. Available: https://www.enterprisestorageforum.com/networking/software-defined-networking-market/
[4] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, "Multi-controller based software-defined networking: A survey," *IEEE Access*, vol. 6, pp. 15980–15996, 2018.
[5] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems UNSW-NB15 network data set)," in *Proc. 2015 IEEE Mil. Commun. Inf. Syst. Conf.*, 2015, pp. 1–6.
[6] S. Zavrak and M. Iskefiyeli, "Flow-based intrusion detection on software-defined networks: A multivariate time series anomaly detection approach," *Neural Comput. Appl.*, vol. 35, no. 16, pp. 12175–12193, 2023.
[7] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Trans. Emerg. Telecommun. Technol.*, vol. 32, no. 1, 2021, Art. no. e4150.

Authorized licensed use limited to: Mepco Schlenk Engineering College. Downloaded on August 05,2025 at 08:56:29 UTC from IEEE Xplore. Restrictions apply.

[8] A. M. Mahfouz, D. Venugopal, and S. G. Shiva, "Comparative analysis of ML classifiers for network intrusion detection," in *Proc. 4th Int. Congr. Inf. Commun. Technol.: ICICT 2019*, London: Springer, 2020, vol. 2, pp. 193–207.

[9] M. Ozay, I. Esnaola, F. T. Yarman Vural, S. R. Kulkarni, and H. V. Poor, "Machine learning methods for attack detection in the smart grid," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 8, pp. 1773–1786, Aug. 2016.

[10] S. Tan, J. M. Guerrero, P. Xie, R. Han, and J. C. Vasquez, "Brief survey on attack detection method for cyber-physical systems," *IEEE Syst. J.*, vol. 14, no. 4, pp. 5329–5339, Dec. 2020, doi: 10.1109/JSYST.2020.2991258.

[11] C.-H. Huang, T.-H. Lee, L.-H. Chang, J.-R. Lin, and G. Horng, "Adversarial attacks on sdn-based deep learning ids system," in *Proc. Mobile Wireless Technol. 2018: Int. Conf. Mobile Wireless Technol.*, Springer, 2019, pp. 181–191.

[12] Q. Niyaz, W. Sun, and M. Alam, "Impact on SDN powered network services under adversarial attacks," *Procedia Comput. Sci.*, vol. 62, pp. 228–235, 2015.

[13] C. Wang, J. Chen, Y. Yang, X. Ma, and J. Liu, "Poisoning attacks and countermeasures in intelligent networks: Status quo and prospects," *Digit. Commun. Netw.*, vol. 8, no. 2, pp. 225–234, 2022.

[14] Z. Tian, L. Cui, J. Liang, and S. Yu, "A comprehensive survey on poisoning attacks and countermeasures in machine learning," *ACM Comput. Surv.*, vol. 55, no. 8, pp. 1–35, 2022.

[15] T. Das, R. Shukla, and S. Sengupta, "Poisoning the well: Adversarial poisoning on ml-based software-defined network intrusion detection systems," *IEEE Trans. Netw. Sci. Eng.*, 2024.

[16] M. Kloft and P. Laskov, "Online anomaly detection under adversarial impact," in *Proc. 13th Int. Conf. Artif. Intell. Statist. JMLR Workshop Conf. Proc.*, 2010, pp. 405–412.

[17] T. Das, O. A. Hamdan, R. M. Shukla, S. Sengupta, and E. Arslan, "UNR-IDD: Intrusion detection dataset using network port statistics," in *Proc. IEEE 20th Consum. Commun. Netw. Conf.*, 2023, pp. 497–500.

[18] H. Xiao, H. Xiao, and C. Eckert, "Adversarial label flips attack on support vector machines," in *Proc. ECAI 2012*. IOS Press, 2012, pp. 870–875.

[19] M. Zhang, L. Hu, C. Shi, and X. Wang, "Adversarial label-flipping attack and defense for graph neural networks," in *Proc. 2020 IEEE Int. Conf. Data Mining*, 2020, pp. 791–800.

[20] H. Zhang, N. Cheng, Y. Zhang, and Z. Li, "Label flipping attacks against naive Bayes on spam filtering systems," *Appl. Intell.*, vol. 51, no. 7, pp. 4503–4514, 2021.

[21] P. Papadopoulos, O. Thornewill von Essen, N. Pitropakis, C. Chrysoulas, A. Mylonas, and W. J. Buchanan, "Launching adversarial attacks against network intrusion detection systems for iot," *J. Cybersecurity Privacy*, vol. 1, no. 2, pp. 252–273, 2021.

[22] E. Alshahrani, D. Alghazzawi, R. Alotaibi, and O. Rabie, "Adversarial attacks against supervised machine learning based network intrusion detection systems," *PLoS One*, vol. 17, no. 10, 2022, Art. no. e0275971.

[23] N. Papernot et al., "Technical report on the cleverhans V2. 1.0 adversarial examples library," 2016, *arXiv:1610.00768*.

[24] A. Shafahi et al., "Adversarial training for free!," *Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.

[25] C. Szegedy et al., "Intriguing properties of neural networks," 2013, *arXiv:1312.6199*.

[26] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," 2017, *arXiv:1705.07204*.

[27] H. Zhang, H. Wang, Y. Cao, C. Shen, and Y. Li, "Robust data hiding using inverse gradient attention," 2020, *arXiv:2011.10850*.

[28] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," 2017, *arXiv:1704.01155*.

[29] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proc. 2017 ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 506–519.

[30] N. Carlini and D. Wagner, "Defensive distillation is not robust to adversarial examples," 2016, *arXiv:1607.04311*.

[31] Z. Lv et al., "AWFC: Preventing label flipping attacks towards federated learning for intelligent IoT," *Comput. J.*, vol. 65, no. 11, pp. 2849–2859, 2022.

[32] Z. Zhang, Y. Zhang, D. Guo, L. Yao, and Z. Li, "Secfednids: Robust defense for poisoning attack against federated learning-based network intrusion detection system," *Future Gener. Comput. Syst.*, vol. 134, pp. 154–169, 2022.

[33] L. da Fontoura Costa, "Further generalizations of the jaccard index," 2022.

[34] S. Lee, "Improving jaccard index for measuring similarity in collaborative filtering," in *Proc. Int. Conf. Inf. Sci. Appl.*, Springer, 2017, pp. 799–806.

[35] E. Tabassi, K. J. Burns, M. Hadjimichael, A. D. Molina-Markham, and J. T. Sexton, "A taxonomy and terminology of adversarial machine learning," *NIST IR*, vol. 2019, pp. 1–29, 2019.

[36] Z. Liu, W. Peng, J. Zhou, Z. Wu, J. Zhang, and Y. Zhang, "MI-FGSM on faster R-CNN object detector," in *Proc. 4th Int. Conf. Video Image Process.*, 2020, pp. 27–32.

[37] X. Zhang, Y. Ma, A. Singla, and X. Zhu, "Adaptive reward-poisoning attacks against reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2020, pp. 11225–11234.

[38] P. P. Chan, F. Luo, Z. Chen, Y. Shu, and D. S. Yeung, "Transfer learning based countermeasure against label flipping poisoning attack," *Inf. Sci.*, vol. 548, pp. 450–460, 2021.

[39] O. Blial, M. Ben Mamoun, and R. Benaini, "An overview on SDN architectures with multiple controllers," *J. Comput. Netw. Commun.*, vol. 2016, 2016, Art. no. 9396525.

[40] B. Almadani, A. Beg, and A. Mahmoud, "DSF: A distributed SDN control plane framework for the east/west interface," *IEEE Access*, vol. 9, pp. 26735–26754, 2021.

[41] H. Yu, K. Li, H. Qi, W. Li, and X. Tao, "Zebra: An east-west control framework for SDN controllers," in *Proc. IEEE 44th Int. Conf. Parallel Process.*, 2015, pp. 610–618.

[42] F. Benamrane et al., "An east-west interface for distributed SDN control plane: Implementation and evaluation," *Comput. Elect. Eng.*, vol. 57, pp. 162–175, 2017.

[43] S. Manikandan, "Measures of dispersion," *J. Pharmacol. Pharmacotherapeutics*, vol. 2, no. 4, 2011, Art. no. 315.

[44] G. Upton and I. Cook, *Understanding Statistics*. London, U.K.: Oxford Univ. Press, 1996.

[45] S. P. Chung and A. K. Mok, "Allergy attack against automatic signature generation," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*, Springer, 2006, pp. 61–80.

[46] B. Biggio, G. Fumera, and F. Roli, "Security evaluation of pattern classifiers under attack," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 984–996, Apr. 2014.

[47] W. Mazurczyk and L. Caviglione, "Cyber reconnaissance techniques," *Commun. ACM*, vol. 64, no. 3, pp. 86–95, 2021.

[48] S. A. Shaikh, H. Chivers, P. Nobles, J. A. Clark, and H. Chen, "Network reconnaissance," *Netw. Secur.*, vol. 2008, no. 11, pp. 12–16, 2008.

[49] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, "Edge-IIoTset: A new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning," *IEEE Access*, vol. 10, pp. 40281–40306, 2022.

[50] S. R. Chen, "6. Decision trees- hands-on-ML," Nov. 2020. [Online]. Available: https://rachelchen0104.medium.com/6-decision-trees-hands-on-ml-f0822ee43219

[51] M. V. de Hoop, D. Z. Huang, E. Qian, and A. M. Stuart, "The cost-accuracy trade-off in operator learning with neural networks," 2022, *arXiv:2203.13181*.

[52] N.-T. Hoang, H.-N. Nguyen, H.-A. Tran, and S. Souihi, "A novel adaptive east–west interface for a heterogeneous and distributed sdn network," *Electronics*, vol. 11, no. 7, 2022, Art. no. 975.

[53] A. Gholamy, V. Kreinovich, and O. Kosheleva, "Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation," *Int. J. Intell. Technol. Appl. Stat.*, vol. 11, no. 2, pp. 105–111, 2018.

[54] Y. Jin, Y. Shen, G. Zhang, and H. Zhi, "The model of network security situation assessment based on random forest," in *Proc. 7th IEEE Int. Conf. Softw. Eng. Serv. Sci.*, 2016, pp. 977–980.

[55] P. Negandhi, Y. Trivedi, and R. Mangrulkar, "Intrusion detection system using random forest on the NSL-KDD dataset," in *Emerging Research in Computing, Information, Communication and Applications: ERCICA 2018*, vol. 2. Berlin, Germany: Springer, 2019, pp. 519–531.

[56] B. Dastjerdy, A. Saeidi, and S. Heidarzadeh, "Review of applicable outlier detection methods to treat geomechanical data," *Geotechnics*, vol. 3, no. 2, pp. 375–396, 2023.

[57] A. Takiddin, S. Rath, M. Ismail, and S. Sahoo, "Data-driven detection of stealth cyber-attacks in DC microgrids," *IEEE Syst. J.*, vol. 16, no. 4, pp. 6097–6106, Dec. 2022.

Authorized licensed use limited to: Mepco Schlenk Engineering College. Downloaded on August 05,2025 at 08:56:29 UTC from IEEE Xplore. Restrictions apply.