University of Nevada, Reno

Modeling the Abnormality: Machine Learning-based Anomaly and Intrusion Detection in Software-defined Networks

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in
Computer Science and Engineering

by

Tapadhir Das

Dr. Shamik Sengupta - Dissertation Advisor
May, 2023

THE GRADUATE SCHOOL

We recommend that the dissertation
prepared under our supervision by

Tapadhir Das

entitled

Modeling the Abnormality: Machine Learning-based Anomaly and Intrusion Detection in Software-defined Networks

be accepted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

Shamik Sengupta, Ph.D.
Advisor

Engin Arslan, Ph.D.
Committee Member

Sergiu Dascalu, Ph.D.
Committee Member

David Feil-Seifer, Ph.D.
Committee Member

Hanif Livani, Ph.D.
Graduate School Representative

Markus Kemmelmeier, Ph.D., Dean
Graduate School

May, 2023

Abstract

Modern software-defined networks (SDN) provide additional control and optimal functionality over large-scale computer networks. Due to the rise in networking applications, cyber attacks have also increased progressively. Modern cyber attacks wreak havoc on large-scale SDNs, many of which are part of critical national infrastructures. Artifacts of these attacks may present as network anomalies within the core network or edge anomalies in the SDN edge. As protection, intrusion and anomaly detection must be implemented in both the edge and core. In this dissertation, we investigate and create novel network intrusion and anomaly detection techniques that can handle the next generation of network attacks. We collect and use new network metrics and statistics to perform network intrusion detection. We demonstrated that machine learning models like Random Forest classifiers effectively use network port statistics to differentiate between normal and attack traffic with up to 98% accuracy. These collected metrics are augmented to create a new open-sourced dataset that improves upon class imbalance. The developed dataset outperforms other contemporary datasets with an F_μ score of 94% and a minimum F score of 86%. We also propose SDN intrusion detection approaches that provide high confidence scores and explainability to provide additional insights and be implemented in a real-time environment. Through this, we observed that network byte and packet transmissions and their robust statistics can be significant indicators for the prevalence of any attack. Additionally, we propose an anomaly detection technique for time-series SDN edge devices. We observe precision and recall scores inversely correlate as ϵ increases, and $\epsilon = 6.0$ yielded the best F score. Results also highlight that the best performance was achieved from data that had been moderately smoothed ($0.8 \leq \alpha \leq 0.4$), compared to intensely smoothed or non-smoothed data. In addition, we investigated and analyzed

the impact that adversarial attacks can have on machine learning-based network intrusion detection systems for SDN. Results show that the proposed attacks provide substantial deterioration of classifier performance in single SDNs, and some classifiers deteriorate up to ≈60%. Finally, we proposed an adversarial attack detection framework for multi-controller SDN setups that uses inherent network architecture features to make decisions. Results indicate efficient detection performance achieved by the framework in determining and localizing the presence of adversarial attacks. However, the performance begins to deteriorate when more than 30% of the SDN controllers have become compromised. The work performed in this dissertation has provided multiple contributions to the network security research community like providing equitable open-sourced SDN datasets, promoting the usage of core network statistics for intrusion detection, proposing robust anomaly detection techniques for time-series data, and analyzing how adversarial attacks can compromise the machine learning algorithms that protect our SDNs. The results of this dissertation can catalyze future developments in network security.

Dedication

Dedicated to my parents, Dr. Tapas Kumar Das and Sudipta Das, my supervisor, Dr. Shamik Sengupta, and myself.

Acknowledgments

Firstly, I thank my Ph.D. advisor, Dr. Shamik Sengupta, for all his support and inspiration. I could not have imagined having a better supervisor, mentor, and friend to help navigate my graduate education. I also thank Dr. Sergiu Dascalu and Dr. David Feil-Seifer, for being excellent mentors and for all their helpful advice in my professional journey. Lastly, I'd like to acknowledge my committee members, Dr. Hanif Livani and Dr. Engin Arslan, for their helpful suggestions to refine my dissertation.

I thank my close friends: Sanjeevan, Shuvo, Pourya, and Prithul, for being there for me during this journey. I also acknowledge my lab mates: Ignacio, Suman, Shafkat, Bibek, Osama, Jay, and Dr. Raj Mani Shukla, for being great co-workers and friends. Additionally, I acknowledge the role that the Oregon Institute of Technology played in my life in building a strong foundation upon which I could achieve my doctoral degree, specifically Professors Todd Breedlove, Troy Scevers, and Kevin Pintong.

Finally, I would like to thank my parents, Dr. Tapas Kumar Das and Sudipta Das, for being exemplary role models and supporting me throughout my life. I also acknowledge my sister, Sagnika Das, and brother-in-law, Dr. Sourav Sikdar, for their constant encouragement and support. A special shout out to Michael, Hans, Alden, and Dennis who have been a constant source of support since high school and college. Last but not least, I would like to thank Jenna for always believing in me.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The creation of the ARPANET in October 1969 revolutionized computer networking forever [1]. Since then, networking has continued to be permanently embedded within the fabric of computer communications. Over time, traditional networks evolved, and their intelligence was distributed across physical network devices like routers and switches. Traditional networks combined the control and data planes. Configuration of network nodes and programming of the paths for data flow was administered by the control plane while, based on this control information, the data plane was responsible for data forwarding at the hardware level [2]. A visual representation of a traditional network is provided in Figure 1.1. However, this network configuration started becoming more challenging, as it became difficult to perform alterations regarding network policies because every physical switch and router needed to be individually reconfigured. Additionally, modern applications of computer networks like the Internet of Things (IoT), vehicular networks, and smart grid technology aim to be dynamic in nature, with computing devices connecting and disconnecting periodically. The network policies for these applications also may require frequent alterations. Also,

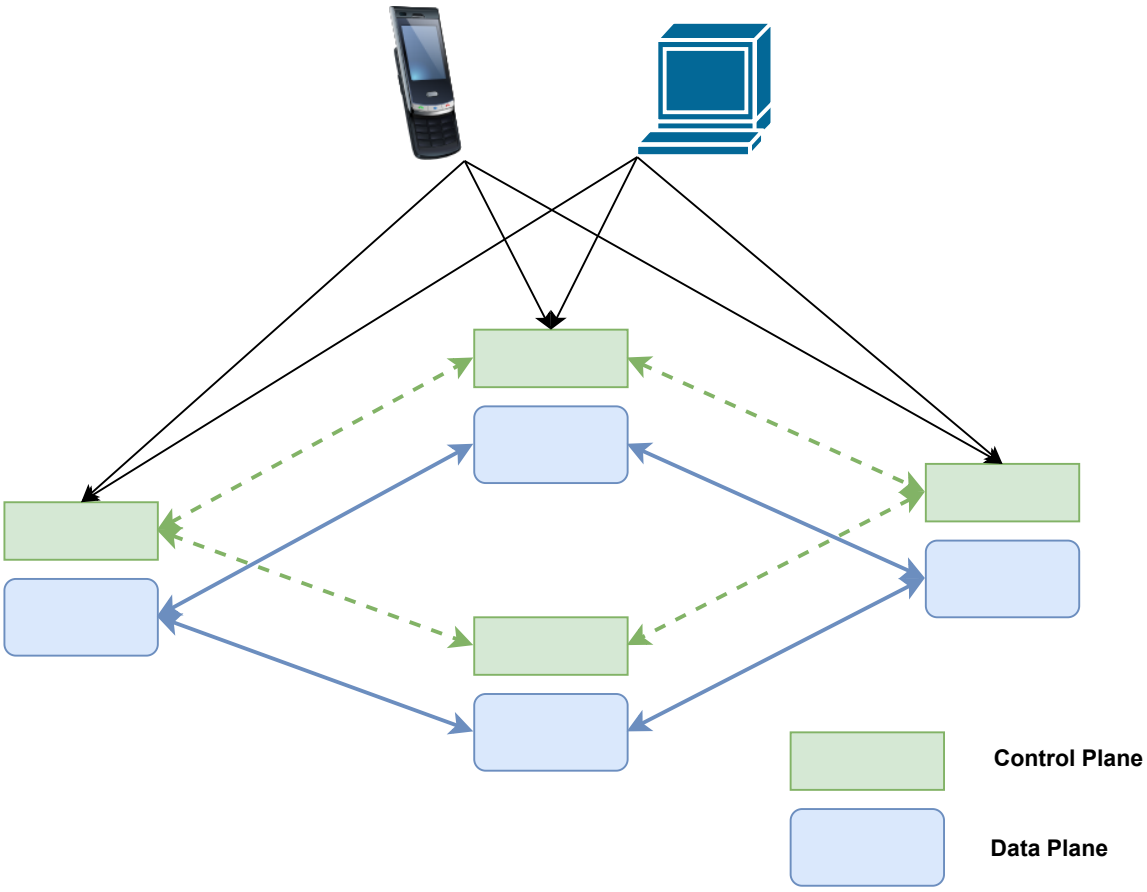


Figure 1.1: Visual Representation of Traditional Computer Networks

modern networking applications are interconnected with other networks, which makes them geographically distributed over a wide area [3]. Hence, the dynamic nature and wide area distribution of modern networking applications are not a proper fit for traditional computer networks.

Software-defined Networks (SDN) were introduced to address the highlighted shortcomings and to act as the new standard for computer networks. SDNs decouple the control and data planes and are administered by a logically centralized SDN controller, which controls mute switches. This consolidated architecture ensures that network issues are handled more easily. An illustration of the SDN architecture is provided in Figure 2.1. The SDN controller is responsible for injecting flow tables in

network switches which control network operations and policies. This makes it easier to enact network policy alterations, upgradations, and monitoring. Operations like network data mining and analysis are made easier due to this architecture. This can be crucial when performing network analysis for improved efficiency, or when dealing with cybersecurity threats that can lead to anomalous network patterns.

Network anomalies refer to situations where network operations and recorded metrics deviate from normal system behavior. These anomalies, traditionally, show themselves within the core of the network and can be problematic in SDNs as they can leave communications in a vulnerable state. Network anomalies can arise due to multiple causes such as internal factors, like physical device malfunction, environmental issues, and/or network overload. They can also be introduced due to cyber attacks like network intrusions including, but not limited to, Denial of Service (DoS), Distributed DoS (DDoS), malicious traffic insertion, Structured Query Language (SQL) injection, fake packet insertion, botnets, and false data injection. These anomalous events are difficult to detect unaided as they often tend to display similar network traffic patterns as that of normal functionality [4].

The problem of network anomalies gets exacerbated in traditional computer networks. Due to their design of dispersed intelligence across network devices, individual cyber threat monitors and anomaly detectors must be independently configured at every single network node. In addition, every single detector must operate in a coordinated manner, which can lead to synchronization problems and complexities for implementation. However, anomaly detection can be more easily performed in SDN setups, due to the logically centralized SDN controller which oversees all network operations and policies. This can make it easier to create a principal anomaly detector, that can be hosted in the SDN controller, and can monitor the whole network for network anomalies.

However, despite their centralized design, SDNs continue to be under attack. The progressive adoption of SDNs across industries, organizations, and infrastructures, along with the steady increase in network devices over time, has made these networks more vulnerable to cyber attacks like network intrusions. Network intrusions are expected to inflict damages up to $10.5 trillion by 2025 [5]. Therefore, organizations must protect their networks from harm as failure to do so can lead to loss of revenue, reputation, and intellectual property. This can be especially fatal if the SDN is connected to a country's critical national infrastructures like the power grid, transportation systems, and energy sectors [6]. Performing robust network intrusion detection in the network core to detect network anomalies can be an effective technique for ensuring critical SDNs are protected.

Another type of anomaly that exists in current SDNs is edge anomalies. These are anomalies that affect SDN edge devices and can rise from cyber attacks that directly target these edge devices like computers, hosts, sensors, servers, automobiles, virtual reality (VR) and augmented reality (AR) systems, and security cameras. In many cases, edge anomalies can lead to the rise of network anomalies within the core network as cyber attacks directed at these devices can affect their operation and, in turn, the recorded network metrics and statistics. Due to increased SDN applications like IoT, Unmanned Aerial Vehicles (UAV), micro-grid systems, smart city infrastructures, and smart home environments, a prominent portion of these edge devices tend to time-series sensors that monitor infrastructure, environment, and user activity. An eminent attack against these edge devices is false data injection, where criminals tend to strategically and maliciously modify/augment sensor data to corrupt the working functionality of these SDN edge devices [7]. Other significant attacks on SDN edge devices include DoS, Data Type Probing, Malicious Control, and Malicious Operation [8]. The ability to detect potential anomalies in edge devices can further

assist in creating more robust anomaly and intrusion detection approaches in the entirety of the SDN.

Due to the rise in new computing technologies, faster resources, and availability of more data, machine learning (ML) has become a very popular technique that is being investigated for SDN anomaly and intrusion detection [9], [10] in both the network core and edge. ML is a useful tool for this purpose due to its capability of finding correlations in traffic flow patterns [11] along with the ability to detect patterns over long sequences. ML techniques can be easily deployed in SDN infrastructures as it is expected to have sufficient computational resources like quicker memory and faster processing speeds. With the availability of open-sourced SDN intrusion detection datasets like [12] [13] and SDN edge device datasets like [14], creating ML-based SDN anomaly and network intrusion detection systems (NIDS) have become more sophisticated and accessible. However, certain voids are prevalent in the current state-of-the-art technology and research available in ML-based anomaly and intrusion detection for SDNs. The next few sections outline the identified voids in this research:

1.1 Network Intrusion Detection - Dataset Voids

In SDN security research, the usage of ML for intrusion detection has received considerable attention, primarily due to the prevalence of open-source datasets like [12] [13]. However, in this dissertation, I have identified some limitations, surrounding the available open-sourced datasets, that need to be addressed to facilitate the creation of more robust intrusion detection approaches to protect future SDN infrastructures. In this section, I highlight these specific limitations.

1.1.1 Unmatching Topologies

Most network intrusion detection datasets that are currently being used for SDN security research are generated/simulated on a singular topological setup. The generated and collected network metrics and patterns that exist in these datasets work well for their specific topologies. However, organizational SDN topologies vary depending on their application, size, and usage. Large organizational SDNs will have more users, spread, nodes, devices, hosts, and links, which can differ from small organizational SDN topologies. This can cause variances in the observed network metrics and collected statistics [15]. Therefore, training an ML-based network intrusion detection model on existing open-sourced datasets may not be fruitful for protecting an organization's SDN infrastructure as it may differ in the number of devices, users, spread, nodes, hosts, and links, and doing so can put the organization at risk of network intrusions.

1.1.2 Overdependence on Flow-based Statistics

Many of the network intrusion detection datasets rely mostly on flow statistics, which again can limit the transferability of solutions to different networks since the flow statistics depend on network topology and traffic characteristics [15]. For example, the original DARPA dataset provides artificially high feature values compared to real traffic data, as it was simulated in a military network environment [16]. The same problem exists in other datasets that are derived from DARPA such as NSL-KDD [12]. Similarly, the open-sourced datasets like the UNSW-NB15 dataset [17] contain multiple flow features like IP addresses, protocols, TCP/IP headers, payload information, recorded start time, SYN Flag, and ACK Flag count. These may end up as redundant features that play no impact in being able to detect potential TCP-SYN flood attacks. Depending on the approach being used to detect TCP-SYN floods,

these features can also increase the cardinality of the dataset, leading to increased training and inference times.

1.1.3 Imbalanced Classes/Labels

Another limitation associated with open-sourced datasets for network intrusion detection is the problem of class imbalance. As these datasets try to provide data from multiple attack scenarios, some of these classes get less represented over the majority classes. This can cause any intrusion detection approach to get skewed towards the majority class and in turn, decrease detection performance. Additionally, none of the datasets provide much information from the core devices of the network like switches and routers. Gathering and reporting this information can highlight insights that can be useful for SDN intrusion detection. Also, due to the primary usage of core devices like switches and their statistics, the number of observed features for intrusion detection can be reduced, helping with operational complexities while providing efficient intrusion detection performance.

1.1.4 Lack in Number of Datasets

The last limitation that exists with open-sourced datasets for network intrusion detection in SDNs is that there are just not that many datasets. Currently, there are three contemporary datasets available, upon which most academic and industrial research gets conducted: NSL-KDD [12], CIC-IDS-2018 [13], and UNSW-NB15 [17]. Due to the limited number of datasets, further research, and the creation of more robust intrusion detection methods in this field gets hindered. One of the restrictions with certain datasets is the lack of testbed and/or topological information for the data. For example, NSL-KDD provides no documentation on the testbed upon which their

dataset was generated. This limits further understanding and insight when performing anomaly detection. Other datasets have historically been a challenge to utilize. UNSW-NB15 is a complicated dataset to parse as data features and their magnitudes are similar between multiple classes. This leads to unsatisfactory performance when it comes to using ML techniques for intrusion detection. More extensive anomaly detection techniques have been proposed just for this dataset [18] [19]. However, these generated techniques are not suitable in live environments for real-time SDN intrusion detection, due to increased inference times resulting from the complicated nature of these techniques. Hence, generating a dataset whose features are rapidly trainable, with proper topological documentation, for live environments would be an attractive expansion for SDN intrusion detection applications.

1.2 Network Intrusion Detection - Algorithmic Voids

Along with the open-sourced dataset limitations in the previous section, there are also some algorithmic limitations for network intrusion detection that have been identified in this dissertation. Addressing these limitations can help create the next generation of network intrusion detectors.

1.2.1 Prevalence of Underconfident Classifiers

One current void in ML-based SDN intrusion detection is low confidence scores when it comes to predictions, which makes it difficult to ascertain the certainty of these predictions. Many classifiers have been proposed that provide high performance at detecting intrusions. However, there has not been much emphasis placed on the confidence with which these classifiers are forecasting their predictions, which can be a hindrance in real-time network intrusion detection. Prioritizing high confidence

levels, along with high accuracy, can allow ML systems to be deployed in real-time scenarios as predictions get forecasted correctly with high probability. Therefore, an ML-based service requires, not only high accuracies in train/test datasets but also high confidence scores. High confidence scores can allow ML predictions to automate changes in SDN flow tables that govern network policies.

1.2.2 Lack of Explainability/Interpretability in Classifiers

Another void is the lack of explainability in ML-based SDN intrusion detection. Many studies have conducted ML-based analysis in SDN intrusion detection, but this research was conducted in a “black-box" manner. This means that the ML approach lacked any transparency and interpretability. This can be an obstacle in network intrusion detection as users will not be able to explain the cause of an intrusion. Having an interpretation of the ML intrusion detector will make it easier for the system to understand which of the network features is more influential in detecting intrusion types. The lack of explainability can be fatal, specifically in SDN networks connected with critical infrastructures, where the intrusion detection process is automated and involves minimal human intervention. Providing explainability will assist in improving trust in the intrusion detector's predictions; specifically on the relevance of the dataset features, the confidence of the predictions, and the justification of the results [20]. Also, it would enable a network system to scrutinize and deduce information beyond a simple knowledge extraction using a model training process. This provides benefit to SDN intrusion detection as ML models do not just classify network flows as normal or anomalous, but also provide evidence for such predictions. It will also help network administrators improve analytics and manage system design policies that protect them from cyber threats like network intrusions.

1.3 Edge Device Anomaly Detection Voids

Performing anomaly detection on SDN edge devices also comes with certain voids in the current state-of-the-art. In this section, we discuss the identified voids surrounding anomaly detection in SDN edge devices.

1.3.1 Restrictions of Time-series Metrics

A prominent portion of current SDN applications includes IoT, UAV, smart city, and smart homes, and a significant chunk of these technologies rely on time-series sensors that monitor infrastructure and user activity. The first limitation is the lack of supervised datasets for these devices, as most of these devices are time-series and hence, unsupervised. This hinders the ability to design mechanisms for more robust anomaly detection on edge devices. There is a need for ML methods that can convert time-series data into a supervised format for more effective behavior modeling.

1.3.2 Sub-optimal Classifier Performance from Noise

Another noteworthy problem associated with real-time sensors for SDN edge applications is the subjection to sensor noise, which in turn, can affect anomaly detection performance. This noise can be introduced during actual analog measurements of the sensor, or even from random variables during data gathering. Due to this sensor noise, anomaly detection performance can degrade, which is why it is essential to propose techniques that can process out sensor noise without compromising anomaly detection performance on these SDN edge devices.

1.4 Detrimental Impacts of Anomaly/Intrusion Detectors to Adversarial Attacks

Lastly, there is limited research that has been conducted in the realm of how ML for robust anomaly and intrusion detection in SDNs can get affected by adversarial attacks. Adversarial attacks are cyber attacks that aim to manipulate ML operations and corrupt their functionality by performing adversarial manipulations on the parameters or other components in ML frameworks. By exploiting training data and model parameters and sensitivities, these attacks can affect the performance of the classifiers, putting the entire MSDN infrastructure at risk. Discovering how adversarial attacks affect SDN ML pipelines can help create new robust security mechanisms that can protect the SDN from these new kinds of threats.

The above-recognized limitations restrict the effectiveness of ML-based intrusion and anomaly detection from its full potential. Additionally, this restrictiveness can result in putting SDNs in jeopardy from cyber attacks and intrusions. Therefore, in this dissertation, we propose the following research directions and objectives:

- Develop a supervised SDN NIDS dataset that is capable of generalizing multiple anomalies and intrusions.
- Enhance the developed SDN NIDS dataset by utilizing network metrics other than traditional flow-based statistics.
- Attempt to minimize the class imbalance problem in the dataset for better anomaly detection performance.
- Create SDN anomaly detection algorithms that have high confidence scores for their predictions.

- Provide explainability to these anomaly detection algorithms to increase interpretability.
- Analyze the impact of adversarial attacks on the achieved performance for ML-based networks and anomaly detection for SDN.
- Create detection strategies for adversarial attacks that aim to compromise ML-based network and anomaly detection for SDN.
- Create ML techniques that can account for time series data types in edge devices, for more robust training.
- Develop methods that can process out SDN edge device sensor noise to improve anomaly detection performance.

The remainder of the dissertation is structured as follows: Chapter 2 provides the background for important concepts for this research, along with related research and their limitations that have been conducted in anomaly/intrusion detection in SDN. In Chapter 3, we introduce the usage of network port and differential/delta port statistics towards network intrusion detection. Chapter 4 highlights our established network intrusion detection dataset called UNR-IDD which improves upon contemporary datasets to improve upon the issue of class imbalance. In Chapter 5, we propose an ensemble learning-based network intrusion detector that prioritizes the confidence scores of its predictions for intrusion detection. In Chapter 6, we provide an anomaly detection framework that can perform anomaly detection on time-series SDN edge devices. In Chapter 7, we investigate and analyze the impact that adversarial attacks can have on ML-based network intrusion detection systems (NIDS) for SDN. Chapter 8 proposes an adversarial attack detection framework for multi-controller SDN setups that uses inherent network architecture features to make decisions. Finally, conclusions are drawn and future research ideas are presented in Chapter 9.

Chapter 2

Background and Related Works

In this chapter, we introduce some background topics that are related to the content presented in this proposal. These background topics include a general overview of modern-day SDN architecture and its benefits, the multiple classifications of anomaly types that exist and how they may exist within the confines of the SDN architecture, and some of the most prominent methods that can be employed to conduct anomaly and intrusion detection in SDN. We also present the related works of the proposed research to see what has already been proposed in the literature, and how we plan to improve upon the established techniques and results.

2.1 Background

2.1.1 Software Defined Network Architecture

An SDN architecture traditionally consists of three main layers: The infrastructure, control, and application layers. It also consists of two different communication in-

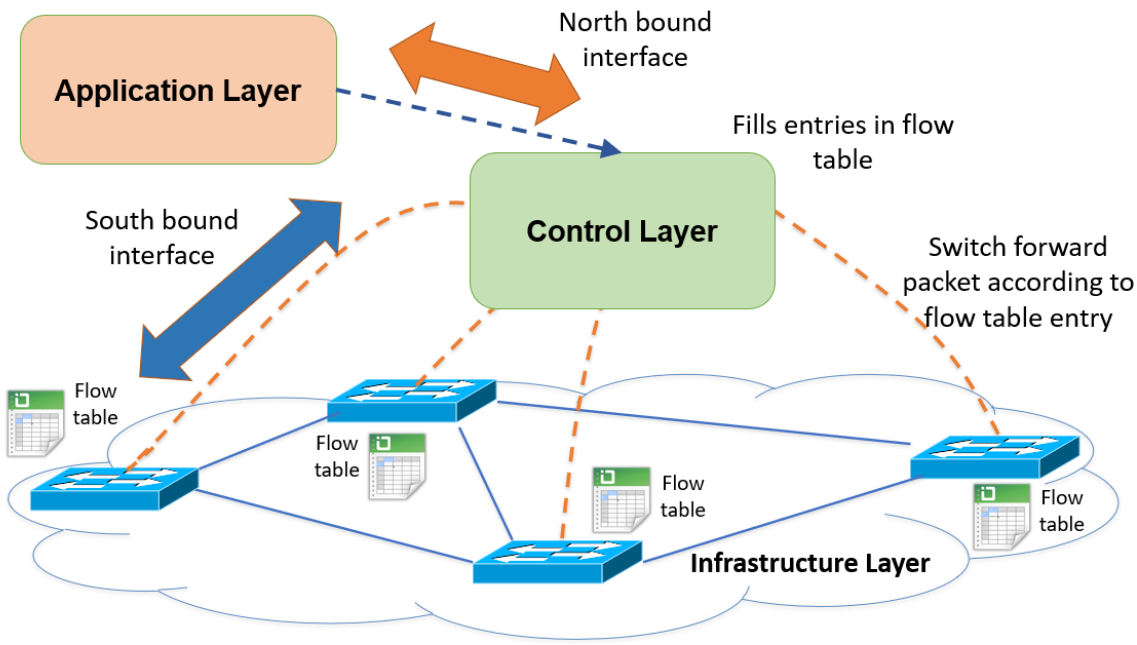


Figure 2.1: SDN Architecture Overview

terfaces: the northbound interface and the southbound interface [21]. Figure 2.1 illustrates a traditional SDN architecture setup.

1. **Infrastructure layer**: Contains physical network devices that make up the network topology like routers, switches, bridges, and repeaters. This layer serves as a medium over which network virtualization can be laid down through the control layer [21].

2. **Control layer**: Contains the centralized SDN controller which regulates the infrastructure network devices and network policies [22]. A lot of business logic is written in this layer to make the controller fetch and maintain various types of network details, state information, topology characteristics, and network statistics.

3. **South bound interface**: This interface is meant to facilitate communication between the infrastructure layer and the control layer of the SDN architecture.

This is typically conducted using southbound protocols like OpenFlow, Netconf, and Ovsdb.

4. **Application layer**: Houses SDN applications that communicate with the SDN controller by leveraging network information like topology, state, and statistics. Examples can be applications for network automation, management, monitoring, configuration, visualization, and analytics. These applications can provide end-to-end solutions for real-world enterprise and data center networks.

5. **North bound interface**: This interface is meant to facilitate communication between the control layer and the application layer of the SDN architecture. This is typically conducted using northbound protocols like REST APIs of the SDN.

As previously stated, SDN decouples the network's control and data planes. This control separation provides a multitude of benefits:

- **Management**: Network operations can be configured, monitored, and troubleshoot from the controller as a complete view of the network can be accessed [23]. This also makes it easier to enact network policy alterations, upgradations, and monitoring. Additionally, operations like network data mining and analysis are made easier due to this centralized management system.

- **Light-weight network equipment**: Due to this centralized architecture, physical network devices like routers and switches can become slimmer and less expensive. As the intelligence is computed at the control level, physical devices can be controlled by rules and guidelines pushed from the control level to the infrastructure level [23].

- **Network virtualization**: This benefit enables and leverages the full potential of the network elements. The SDN controller can abstract the underlying

physical network to allow administrators to program virtual networks for each tenant [23].

2.1.2 Types of Network Anomalies

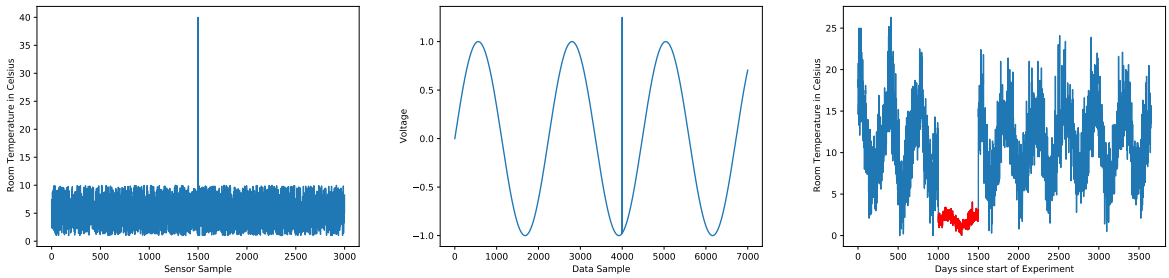
According to Douglas Hawkins, a statistician and Professor at the University of Minnesota, an outlier or anomaly is defined as “an observation which deviates so significantly from other observations as to arouse suspicion that it was generated by a different mechanism" [24]. Within the confines of SDN, it can be said that an anomaly is an unexpected deviation of data from an otherwise expected distribution. The anomaly can occur concerning both local and global norms for the data distribution [25]. There are several important observations when it comes to defining the nature of normal data in SDNs.

1. Most of the data that is captured can be defined as “normal" data, as it appropriately fits the expected characteristics and distribution for that network when functioning naturally.

2. The concept of “normal" operation in SDNs can change over time for multiple reasons, including but not limited to the number of active users, number of hosts, and dynamic changing of the network topology for certain applications.

Within the context of SDN anomalies, we can identify three types of anomalies that frequently occur: point, contextual, and pattern anomalies [25].

- **Point anomalies**: These are anomalies that tend to diverge for single or minimal observations. They are characterized by their return to normal system state within a few observations. These anomalies can represent statistical noise or signal noise within SDN edge devices or latency within the core SDN network due to device malfunction or faulty sensing. They can also represent a short



(a) Point Anomaly (b) Contextual Anomaly (c) Pattern Anomaly

Figure 2.2: Common Anomalies in SDN

time of interest to the network administrators. An example of a point anomaly can be seen in Figure. 2.2a.

- **Contextual anomalies**: Contextual anomalies are observations that deviate from a normally expected sequence of patterns in a time series. It is a deviation from a norm in the context of the surrounding observations. Within SDN applications, contextual anomalies can represent an attempt at a false data injection attack on an edge device. An example of a contextual anomaly can be seen in Figure. 2.2b.

- **Pattern anomalies**: Pattern anomalies refer to a collection of observations that are anomalous from the rest of the distribution. Individual observations may or may not look anomalous, but when viewed with the rest of the data, they arise suspicion. With the confines of SDN, pattern anomalies can represent sustained attacks within the network at both edge and core devices like DDoS. An example of a pattern anomaly can be seen in Figure 2.2c.

2.1.3 Anomaly Detection Methods

Anomaly detection is a research area that spans multiple domains like engineering, business, economics, and science. Hence, over time, multiple anomaly detection meth-

Table 2.1: Summary of commonly used anomaly detection methods in SDN

Anomaly Detection Methods	Prominent Algorithms	Advantages	Disadvantages
Statistical methods	- Wavelet analysis - Principal component analysis - Covariance matrix	-Innate ability to detect anomalies - No requirement of prior system knowledge	- Significant time requirements for training - Thresholds not representative of real-world scenarios
Clustering methods	- K-Means - DBSCAN	- More stable than statistical methods - Faster response during training	- Time consuming to train - May get trapped in local minima
Finite state machine methods	- Markov chains - hidden Markov models	- Robust - Flexible to data and changes	- Time consuming to train - May not detect uncommon anomalies
Classification methods	- Naive Bayes, - Support vector machine, - Deep neural networks, - Ensemble methods	- High accuracy and detection rates - Adaptive in nature	- High resource consumption - Tendencies of overfitting

ods have been proposed to better catch outliers embedded within normal system data [26]. In this section, we provide a list of the most used anomaly detection techniques that have previously been studied to perform SDN anomaly detection.

- **Statistical methods**: Statistical methods are widely employed in performing anomaly detection. These methods typically employ the use of probabilistic models associated with training data to track appropriate network behavior. Sudden changes in SDN data result in anomalies, which are caught using hard thresholds. Common techniques previously used include wavelet analysis, principal component analysis, covariance matrix, traffic filtering, and correlational paraconsistent machine.

- **Clustering methods**: Clustering techniques aim to congregate data points of similarity into groups or “clusters". These techniques are adaptable to dynamic changes, which do occur in SDN data. Clustering techniques can identify values that are far away from other clusters and hence can be flagged as potential anomalies. These techniques can also be used as a pre-processing step for other anomaly detection algorithms. Commonly used techniques for network anomaly detection includes K-means and DBSCAN.

- **Finite state machine methods**: Finite state machines are models comprising states, actions, and transitions. Each state stores information about the past and the changes that have occurred since entry to the state, beginning when

the system first started. A state transition occurs due to a condition, and a corresponding action can be taken during that time. These techniques can be used because they have high anomaly detection rates as there is considerable knowledge regarding normal and attack cases. Common techniques include Markov chains and hidden Markov models (HMM).

- **Classification methods**: The modern way to conduct SDN anomaly detection is using classification methods. This, typically, consists of two steps: training and testing. During training, a classifier is built using labeled training data. Then, testing data is parsed through the trained classifier to classify network metrics as either “normal" or “anomalous". Classification problems can be both binary or multi-class classification. Common classification methods include Naive Bayesian, support vector machines, deep neural networks, and ensemble-based approaches.

Table 2.1 provides a summary of the commonly used anomaly detection techniques used in SDN.

2.1.4 Adversarial Points of Compromise in Machine Learning Pipelines

Network intrusion detection is an essential component to keep each SDN environment safe from potential cyber attacks. In multi-controller architectures, each SDN is locally trained with collected training data. This training data is sourced from network sensors, metrics, and statistics from network 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) [27]. These four TAs are also significant targets within ML-based NIDS as they can compromise

network security and functionality. The four main TAs are:

- **Input domain**: This domain contains network metrics, states, and statistics from multiple network devices like hosts, routers, switches, computers, mobile devices, vehicles, charging stations, sensors, etc., and is located within the Data plane level of the MSDN architecture. Prospective attacks targeting this domain include malicious tampering with the collected data that is to be fed into the ML pipeline.

- **Data Pre-processing**: This domain includes techniques for data preparation before being fed into the ML model. These techniques include noise processing, feature extraction, dimensionality reduction, data sampling, etc., and are located within the SDN controller and a part of the Control plane of the multi-controller SDN setup. Here, attackers can manipulate collected data sets that are being pre-processed and maliciously alter them so that tampered data are presented to the ML model for training.

- **Machine learning model**: The primary TA within the pipeline is the ML model itself. This ML model can be any algorithm like neural networks, decision trees, random forests, support vector machines, reinforcement learning techniques, etc., and is located within the SDN controller and a part of the Control plane. Attackers can poison data or labels that are being processed or create generative adversarial examples of data that deceive ML algorithms to make misclassifications and false predictions. These attacks can occur during both the training and testing phases of the model. This TA is the most lucrative of all the TAs in the ML pipeline as any adversarial attack can incur the most damage to the multi-controller SDN setup, compared to the other TAs. Specifically, the attacks conducted during the training phase are the most

lethal, as their impact can lead to incorrect training of the ML model, and hamper performance by triggering misclassifications.

- **Output domain**: This domain includes methods to output the predictions, classifications, and forecastings of the ML model. The primary function of this domain in an ML-based NIDS is to perform network intrusion detection and provide system state, status, and anomalies to the system administrators. This can be achieved using displays, status lights, and other visualization methods. The domain is in the Command center and receives output from all the SDN controllers in the Control plane. Potential attacks can involve malicious tampering with the output sensor or display data that are interpretable to the remainder of the system and the system operators.

An illustration of the ML pipeline for ML-based NIDS and the four main TAs is illustrated in Figure 2.3.

2.2 Related Works

SDN intrusion detection has received appreciable attention and has been liberally researched in literature. An important driving force behind this research has been the prevalence of SDN datasets upon which ML algorithms and other approaches have centered. The original dataset created was the DARPA dataset developed at MIT Lincoln Laboratory [28], which consisted of 41 features and a variety of attack types. However, this dataset has been rendered obsolete as it no longer represents real-world network data scenarios. DARPA also does not represent general networks as it was simulated in a military network environment. Thus, it contains artificially high feature magnitudes compared to real traffic data. The next major dataset was the KDD Cup 99 [29] introduced in 1999, which also suffered from redundant and

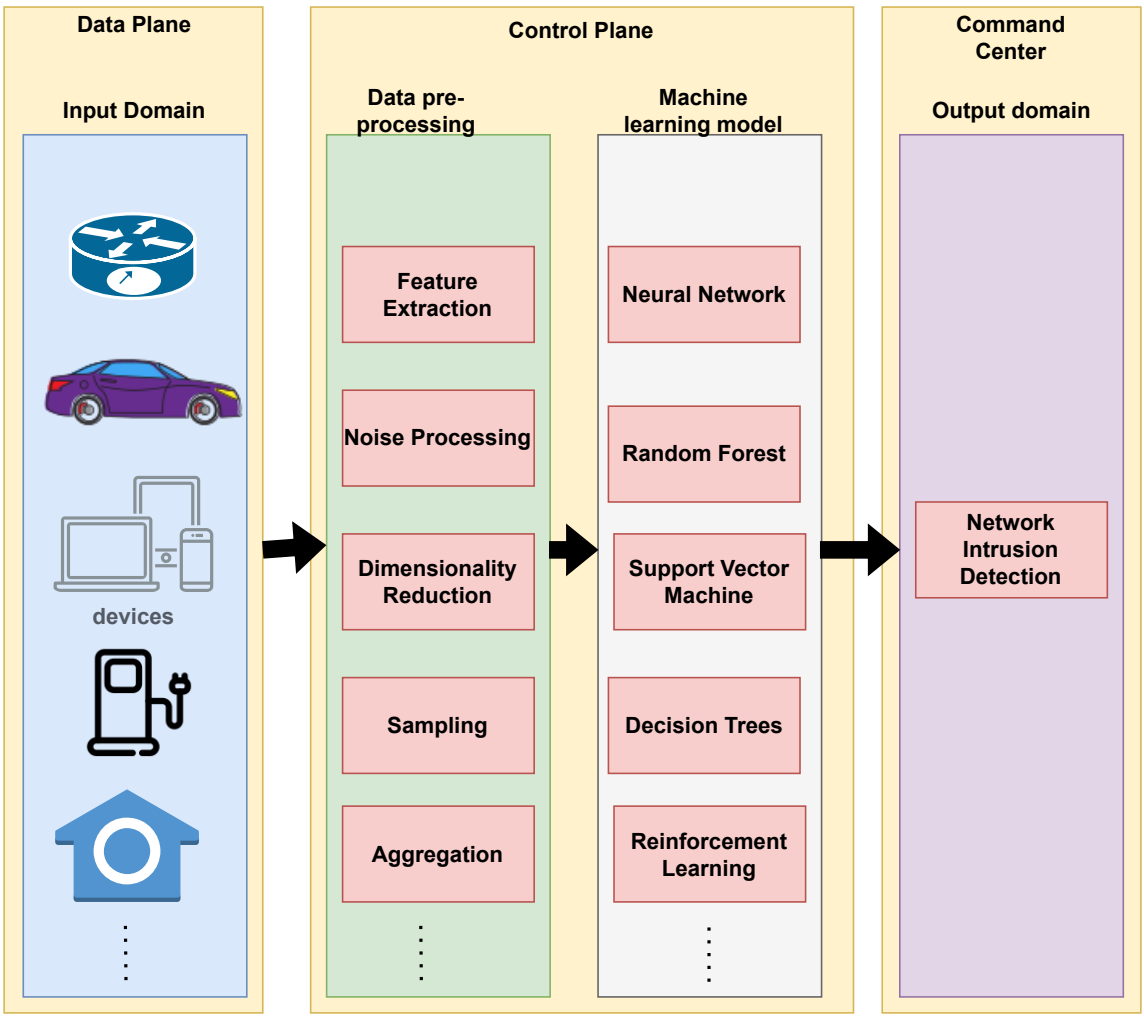


Figure 2.3: Four Targets of Attack in ML-based NIDS

duplicate data samples, rendering intrusion detection inefficient on this dataset. Other prominent datasets that also were generated included CAIDA [30], CDX [31], Kyoto [32], Twente [33], and ISCX2012 [34]. CAIDA dataset is limited as it contains only 20 features and is solely focused on detecting DDoS attacks. Similarly, the CDX dataset from the United States Military Academy contains only 5 features and focuses on detecting Buffer Overflows. The dataset from Kyoto University performs SDN intrusion detection; however, it also contains a limited number of dataset features and only performs binary classification. Due to the relatively simple setup of the

above datasets, intrusion detection approaches may be missing additional insights which could be vital to system performance. The dataset from the University of Twente is solely focused on IP flows at the network level. However, the size of this dataset is small, and the scope of the attack types is limited [35]. Similarly, the ISCX dataset also only consists of IP flows.

Currently, the most prominent datasets are NSL-KDD, UNSW-NB15, and CICIDS-2018. Yet, these datasets also come with their limitations. For instance, NSL-KDD has no documentation on the testbed upon which their dataset was generated. This limit further understanding and insights when performing intrusion detection. UNSW-NB15 is a complicated dataset where dataset features between various labels are similar in magnitude, leading to inefficient intrusion detection performance on basic intrusion detection techniques. More extensive detection techniques have been proposed just for this dataset [18] [19], but these cannot be deployed in a live environment due to increased training and inference times. The NSL-KDD and CIC-IDS-2018 datasets suffer from missing data samples within their datasets. Additionally, all three datasets suffer from class imbalance, which degrades intrusion detection performance. Lastly, none of these datasets capture any network information from the core devices like switches and routers, which might provide additional insights and improve intrusion detection performance.

Using open-sourced datasets, multiple intrusion detection studies have been conducted in the literature. These techniques are continually being proposed to provide improved performance for the current-day network architecture and modern attack types. Researchers in [36] addressed intrusion detection in SDNs using Hidden Markov Models (HMMs) to create an adaptive intrusion detection technique. The authors in [37] attempted to use an HMM to make out the presence of noise in the training dataset for detecting attacks against web applications. The work conducted

in [38] performed SDN intrusion detection using a multi-class HMM where each of the HMM layers is geared for a specific type of network traffic. The main problem of using HMMs or any other finite state machine method is the inefficient intrusion detection performance if they encounter uncommon anomalies [26].

Other researchers have employed mechanisms like statistical approaches to perform SDN intrusion detection. The work in [39] used Principal Component Analysis (PCA) to analyze SDN traffic and detect potential DDoS attacks. The authors in [40] employed Discrete Wavelet Analysis to identify intrusions in SDN network traffic. Researchers in [41] attempted to perform flooding-based DoS detection in cloud environments using Covariance Matrix. The limitation in these works is that statistical techniques require the use of a hard statistical threshold that does not represent real-world threat scenarios due to their limited and static nature [26].

Researchers, also, looked at clustering techniques to perform SDN intrusion detection. The authors [42] used a distributed SDN and k-means clustering to achieve optimal intrusion detection in smart grid communications. Similarly, the authors in [43] used a meta-heuristic clustering technique called WOA-DD to detect DDoS attacks in SDN. The limitation of clustering algorithms is that they could get stuck in local minima and provide incorrect predictions [26].

The emergence of machine and deep learning has enabled researchers to investigate SDN intrusion detection using these mechanisms. For instance, the authors in [11] developed a deep learning mechanism for intrusion detection in SDN. Similarly, in [44], the researchers proposed a Gated Recurrent Unit Recurrent Neural Network to conduct intrusion detection in SDN. The work conducted in [45] proposed the usage of sFlow, adaptive polling-based sampling, and deep learning to detect various DDoS attacks in an SDN. The knock on these works is the low priority given to the

confidence scores of their predictions. In live environments where real-time intrusion detection must be performed, the intrusion detection mechanism must provide high confidence scores for their predictions. This can enable ML systems to be deployed in real-time scenarios as predictions get forecasted correctly with high probability. The above methods also do not investigate the concept of explainability in SDN intrusion detection. These studies are conducted in a “black box" approach and provided no interpretability.

The concept of explainability in SDN intrusion detection has received limited attention, recently. The work in [46] investigated the usage of a variational autoencoder for intrusion detection in conjunction with a gradient-based fingerprinting technique for explainability. The research in [47] developed an SDN intrusion detection method that uses a random forest classifier that incorporated explainability. Similarly, the authors in [48] explored the usage of multiple classifiers with the SHapley Additive exPlanations (SHAP) framework to explain detected SDN intrusions in a single SDN dataset. These published studies focused on intrusion detection performance on singular algorithms on openly available SDN datasets. The limitations of these datasets have been described above.

Some works investigated the usage of network-based methods to detect intrusions, where detection and response methods are deployed at network devices like routers and switches [49] [50] [51] [52]. In [49], the researchers proposed a route-based packet filtering method as an expansion to ingress filtering at the core of the network. The work in [50] proposed Watchers that detected misbehaving routers that launched DDoS attacks by absorbing, discarding, and misrouting packets. In [51], the authors presented SAFETY that provided early detection of DDoS floods like TCP-SYN by harnessing the programming and wide visibility approach of SDN with entropy method to determine the randomness of flow data. The experiments in this work

were conducted on only singular destination victims, which is not always the case in TCP-SYN attacks. [52] proposed AEGIS that detected and mitigated SYN floods against the SDN controller by regularly checking if there is a performance lag in the controller during floods. This method can detect SYN floods after the controller performance drops, by which significant damage could have already been done to the controller. Additionally, direct network-based techniques suffer from high storage and processing overhead at the core devices, which can cause deficient performance [53].

For time-series-based SDN edge devices, there has been a limited amount of research for robust anomaly detection. The main reason for this is the lack of available supervised datasets for timer-series SDN edge devices. On top of this, statistical approaches like PCA and Wavelet Analysis do not work on time-series data. Clustering approaches have been a viable option for time-series-based SDN edge device anomaly detection. The authors in [54] proposed CLAPP, a self-constructing feature clustering technique for anomaly detection on edge devices. Similarly, [55] proposed using a fuzzy clustering-based artificial neural network for anomaly detection in cloud computing devices. Clustering methods can be vulnerable in edge devices, as they depend on normal system state to make decisions. Time-series-based SDN edge devices, on the other hand, may dynamically change their normal state over time. This can lead to misclassifications.

Deep learning has also been another viable option to conduct anomaly detection in these devices. The authors in [56] proposed a convolutional neural network (CNN) based anomaly detector that can detect point, contextual, and discord anomalies in time-series data. However, the limitation of this technique is that CNNs cannot account for the historical value of a time-series data point. That may be essential for efficient anomaly detection and forecasting. The work in [57] investigated anomaly detection using a Long-Short Term Memory (LSTM) network, where anomalies were

detected using error computation. Lastly, the researchers in [58] proposed a supervised anomaly detection approach, based on the time-series data statistics. The limitation of the above techniques is that they are susceptible to noisy data, which can be a common additive from sensor values and can lead to misclassifications.

As the study of adversarial attacks against ML algorithms is slowly increasing in popularity, research has also extended to studying their impact on ML-based NIDS. Certain adversarial attacks have been proposed in the literature that is geared towards ML-based NIDS [59] [60] [61]. In [59], the authors propose a novel adversarial attack against deep learning NIDS. The method consists of two techniques which include model extraction, to replicate the black-box model of a deep learning method, and a saliency map, to disclose the impact of each packet attribute on the detection results. The work in [60] proposed a novel hierarchical adversarial attack generation method to realize the level-aware black-box adversarial attack strategy that targets graph neural network-based intrusion detection systems. The researchers in [61] explored a DoS adversarial attack to craft adversarial samples on an artificial neural network-based intrusion detection system.

Some techniques that have been established to counteract the impact of adversarial attacks include adversarial training [62], gradient hiding [63], and defensive distillation [64]. However, these techniques are not completely robust to adversarial attacks. Adversarial training can be bypassed using two-step attacks as seen in [63]. Gradient hiding can be circumvented by learning a surrogate black box that contains visible gradient information and crafting examples using that [65]. Lastly, defensive distillation is ineffective against black-box attacks [66].

Like many adversarial attacks, label manipulation attacks have become a point of compromise for many ML pipelines [67] [68] [69]. The authors in [67] proposed an

optimization framework to perform label manipulation attacks that aim to maximize the classification error of a supervised classifier. In [68], 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 [69] proposes multiple label manipulation attacks that aim to compromise the performance of Naive Bayes classifiers used on spam filtering systems. Label manipulation attacks in NIDS have also received limited attention [70] [71]. In [71], the authors performed label manipulation attacks on two ML-based NIDS to evaluate their performance. The authors in [70] propose a targeted label manipulation/poisoning attack that aims to flip 0%-50% of labels in a training dataset for an SVM-based network-based intrusion detection system using the Clever Hans python library [72]. However, the experiments conducted in this work only focused on label manipulation attacks against Support Vector Machine (SVM)-based NIDS. They did not study its effects on other customarily used network intrusion detection system algorithms like Random Forest (RF) and Multi-Layer Perceptron (MLP).

Research in protection for NIDS against label manipulation attacks is very limited and still, a new research area [73] [74]. In [73], 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 [74] proposes SecFedNIDS, a novel label manipulation attack detection technique using classpath similarity. The limitation of this work lies in the fact that the authors use 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 [75] [76].

The work being proposed in this Ph.D. dissertation aims to address the issues and current voids that have been identified in the previous chapter, concerning SDN anomaly and intrusion detection. The proposed work aims to generate a brand new SDN dataset for conducting intrusion detection. The dataset aims to capture network metrics from a simulated SDN environment, to provide network topology which can add contexts and additional insights to intrusion detection. The dataset will be supervised and will include network metrics and statistics captured from a multitude of attack scenarios as well as under normal conditions. The dataset will include, primarily, port-level information from the core network devices like switches and routers. The goal of using this switch information is to reduce the number of dataset features, while still attempting to achieve optimal intrusion detection performance. The prospective dataset will also try to address the issue of class imbalance in existing datasets, by attempting to ensure minimal variability between the number of examples between classes in the dataset. This provides novelty as it adds to the number of existing SDN intrusion detection datasets while being unique as it is trying to conduct supervised anomaly detection using, primarily, port-level information from switches. It also adds novelty as the issue of class imbalance can be addressed without using oversampling and undersampling techniques. This can ensure that intrusion detection is performed optimally, without any skew toward a subset of classes.

The proposed work also aims to address the limitation of low confidence scores and the lack of explainability when it comes to SDN intrusion detectors. Having low or mediocre confidence scores for anomaly detection makes it difficult to ascertain the certainty of these predictions. Therefore, this work also aims to create anomaly detection approaches that provide a high mean confidence score for predictions. The work also focuses on providing explainability to SDN anomaly detection, which has

received minimal attention in the current state-of-the-art. Providing interpretability to anomaly detection algorithms will help the system understand which of the network features is more influential in detecting anomaly types. This provides novelty as most published research have given low priority to the confidence scores of their model predictions. Having high confidence scores can help make anomaly detection approaches more assured in their predictions, and it would also help automate ML systems to be deployed in real-time scenarios as predictions get forecasted correctly with high probability. The inclusion of explainability is also novel as it improves trust in the anomaly detector's predictions; specifically on the relevance of the dataset features, the confidence of the predictions, and the justification of the results.

Additionally, the proposed work aims to conduct anomaly detection in SDN edge devices like time-series sensors, due to their wide application and usage. The work aims the limitations in current research like the lack of supervised datasets and the adverse effect of sensor noise on anomaly detection in SDN edge devices. This provides novelty as finding a technique that can convert unsupervised time-series data to a supervised format can help make more robust anomaly detectors. It also eliminates the need to create a supervised dataset, which is already difficult to do for time-series edge devices. Also, coming up with appropriate sensor noise processing techniques will ensure that the device is minimally affected by raw sensor noise. This will lead to better anomaly detection performance over time and will ensure these SDN edge devices are protected from cyber attacks.

Finally, the proposed work investigates the impact of adversarial attacks on ML-based NIDS in SDN. The work aims to run label manipulation attacks against NIDS to analyze their impact on performance. Negative results will motivate the need for a label manipulation attack detection mechanism for NIDS that does not depend on an ML-based or statistical mechanism that can be circumvented and resource exhaustive,

but by utilizing other inherent network features to its advantage.

Chapter 3

SDN Attack Detection using Network Port Statistics

To prevent the limitation of unmatching topologies and an overdependence on flow-based statistics, we propose an ML-enabled Transmission Control Protocol (TCP)-Synchronization (SYN) flood detection framework. To restrict the overdependence on flow-based metrics, we use OpenFlow port statistics as our primary network metrics. To ensure that organizations were capable of modeling their infrastructure topologies, an SDN simulation environment is utilized to simulate network activity. We demonstrate that ML models such as Random Forest classifiers can differentiate normal traffic from SYN flood traffic with high performance. We also introduce novel threat detection and localization techniques that can pinpoint where the attack traffic originates from in the network.

3.1 Effect of Cyber Attacks on SDN

SDNs were originally introduced to provide modularity and simplicity to modern networking infrastructures by decoupling the data and control planes. Due to their increased sophistication, they have become a permanent component in modern network communications. Cyber attacks such as Distributed Denial of Service (DDoS) threaten the healthy operation of SDN networks. The number of global DDoS attacks is expected to reach 15.4 million by 2023 [77]. The main goal of a DDoS attack is to consume and exhaust the resources, like bandwidth and memory, of a target machine, thereby preventing it from serving legitimate and legal requests. An eminent DDoS attack against SDNs is the TCP-SYN flood, which takes advantage of the Transmission Control Protocol handshake mechanism, by not returning the final Acknowledgement (ACK) packet to the target node. Attackers keep sending SYN packets to the target, thereby consuming the target device's resources [78] as illustrated in Figure 3.1.

To protect against DDoS attacks like TCP-SYN floods, we propose a TCP-SYN flood detection framework using an SDN simulation environment that addresses the above-mentioned limitations. Organizations can gear the simulation environment and record metrics towards their topology, instead of relying on using the recorded metrics derived from static topologies from the SDN datasets. This can provide better performance and security. We also investigate different network statistics to conduct attack detection. Flow statistics may contain many redundant features and can increase cardinality, leading to higher training, and inference times. Another option is to utilize port-level statistics instead of flow statistics where we capture statistics from every single SDN port in the infrastructure.

Our proposed approach focuses on utilizing port statistics in conjunction with delta/differential port statistics. Differential port statistics refer to the change in magnitude of observed

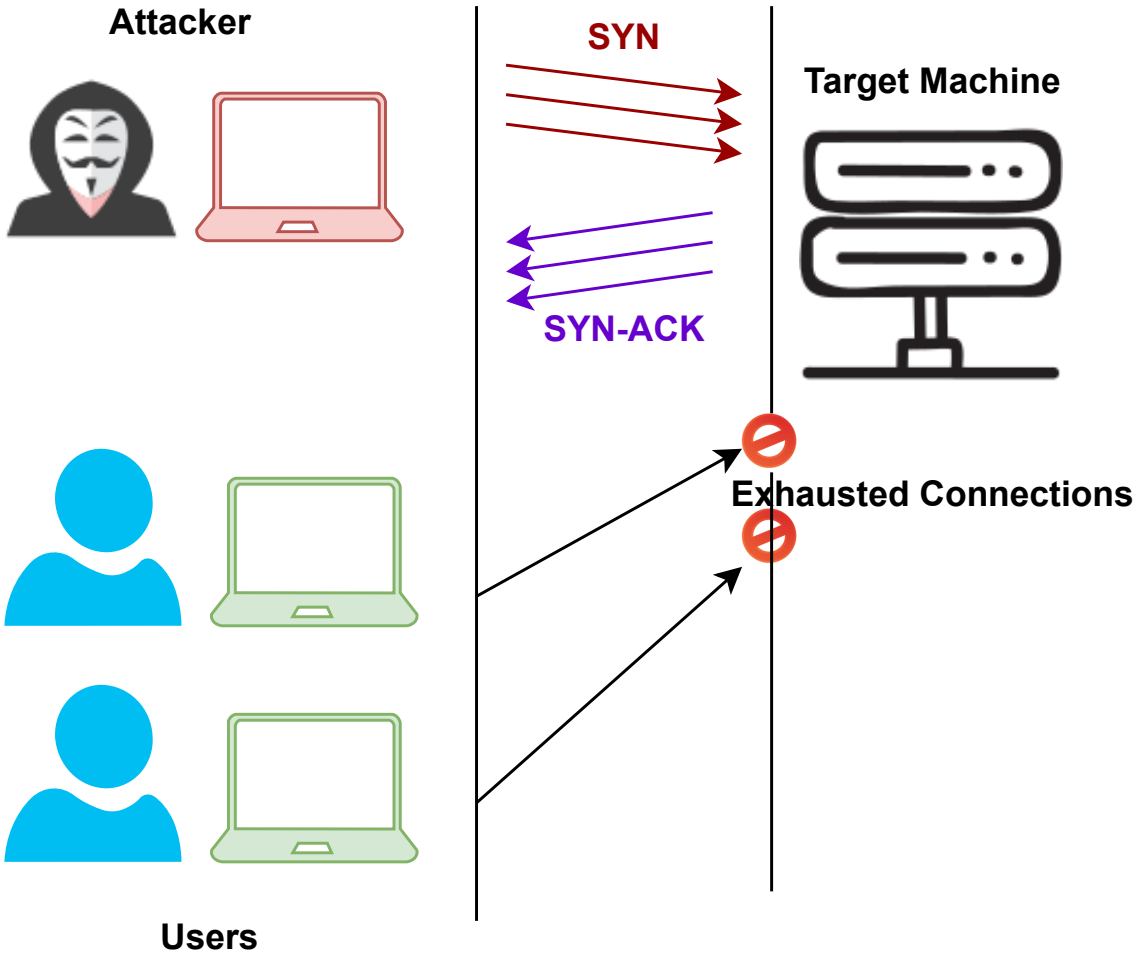


Figure 3.1: TCP-SYN flood attack

statistics in switch ports in a time interval. Differential port statistics can provide a more fine-grained analysis of network flows from the port level. Using both types of port statistics, this range can be higher over the magnitude scale and can lead to more accurate performance and faster identification. This can allow us to rapidly capture and identify potential TCP-SYN flood attacks from the network traffic before they cause harm. To avoid the class imbalance problem, we gather an equal number of logs for both normal and malicious traffic.

Our main contributions proposed in this work are:

- Setting up an SDN simulation environment that each organization can gear

towards their topology.

- Capturing various port statistics under both normal and attack conditions to model both circumstances, while minimizing class imbalance by ensuring an equal representation.
- Training an ML model on our captured data to generalize both conditions.
- Testing the performance of our approach towards real-time detection of TCP-SYN floods.

3.2 Methodology

Our proposed approach primarily consists of using an SDN simulation environment to build custom topologies that each organization can cater to its use case. Network flows are simulated to showcase the appropriate functionality of the generated network. During these flow simulations, port statistics are collected, which form the basis of modeling normal behavior in the simulated SDN. Following this, a TCP-SYN flood attack is launched that targets a particular host in the network. The same network metrics are collected under this scenario. The accumulated network metrics form the dataset to train an ML model in an offline manner. The trained ML model can then be placed in the SDN controller, equipped to detect TCP-SYN floods in real time. Figure 3.2 illustrates our proposed framework.

3.2.1 SDN Simulation Generation

To set up the simulation environment we used Open Network Operating System (ONOS) SDN controller (API version 2.5.0) alongside Mininet. ONOS uses Open Service Gateway Initiative (OSGi) service component at runtime for the creation and

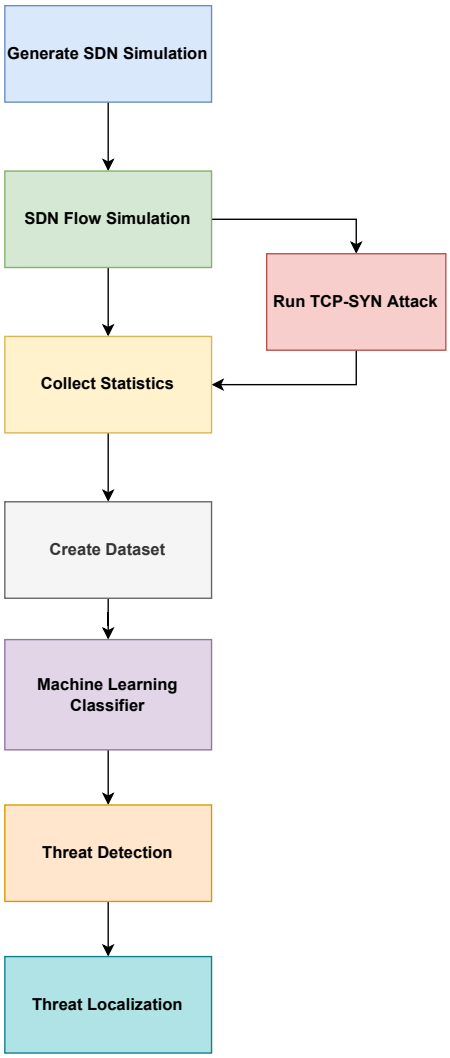


Figure 3.2: Proposed TCP-SYN detection framework

activation of components and auto-wiring components together, which makes it easy to create and deploy new user-defined components without altering the core components. Mininet creates a realistic virtual network and runs a real kernel, switch, and application code on a single machine which creates a realistic testbed environment. We also created our ONOS application (component) to collect the needed statistics. We were able to gather port, flow entry, and flow table statistics for each connected Open vSwitch in the Mininet topology. We created a custom Mininet topology using Mininet API (version 2.3.0) with Open Flow (OF) 14 protocol deployed to the

switches.

3.2.2 SDN Flow Simulation

IPerf is a tool for testing, measuring, and simulating network functionalities. It is used to create TCP and UDP data streams simulating network flows in virtual and real networks with pre-chosen source and destination IPs and with a dummy payload. By using the Mininet API with IPerf, we created a Python script to simulate realistic network flows. Every 5 seconds, we created a flow between a randomly chosen source-destination host pair with a bandwidth of 10 Mbps that lasted for 5 seconds. Hence, we identify the total number of normal flows by N. This represents the number of simulated flows under normal working conditions of the simulated SDN and not under any attack.

3.2.3 Collect Statistics

As mentioned in Section 3.2.1, we created a custom application to collect and log the available statistics that get polled on a configured interval (5 seconds) from OF switches. We use port statistics which were collected by the message exchanging of OFPPortStatsRequest and OFPPortStatsReply and computed on the controller side by taking the difference between the last two collected data instances. We created a key-value map of this data by gathering it from the data storage service using the "Device Service" API provided by ONOS. After this, we logged the map of the collected statistics to a Javascript Object Notation (.json) file with a name $N_i.json$. Table 3.1 shows the collected statistics and their descriptions per port on every switch in the simulated SDN. These port statistics are chosen as they are customarily available in most SDN setups. Also, in the case of any kind of DDoS threat like TCP-SYN, met-

Table 3.1: Port statistics collected for every port on every switch

Port Statistic	Description
Received Packets	Number of packets received by the port
Received Bytes	Number of bytes received by the port
Sent Packets	Number of packets sent by the port
Sent Bytes	Number of bytes sent
Port alive Duration	The time port has been alive in seconds
Packets Rx Dropped	Number of packets dropped by the receiver
Packets Tx Dropped	Number of packets dropped by the sender
Packets Rx Errors	Number of transmit errors
Packets Tx Errors	Number of receive errors

rics that get influenced by byte/packet transmission along with network throughput are key in detecting if a DDoS attack is actively exhausting network resources.

Additionally, we collect other network metrics as shown in Table 4.3. These metrics provide the real-time state of the OF ports within the switches of the SDN. Also, they are contemporary measures that can be collected from any SDN setup, including any SDN simulation environment like the one utilized in our proposed framework. We denote the dataset of collected port statistics under normal conditions as X_N. The total number of samples in X_N can be symbolized as E. Here, x_{n_i} represents a single data sample and $x_{n_i} \in X_N, i = \{0, 1.....E\}$. All the samples in X_N are labeled as *normal*.

3.2.4 TCP-SYN Attack

To capture port statistics during a threat, we launch a TCP-SYN flood attack on the network. This attack is launched with a particular host machine as the intended victim. The compromised machines that launch the attack, from here on referred to as attackers, can be any of the remaining host machines in the simulated SDN topology. Let the total number of attack flows be A. This represents the number

Table 3.2: Additional statistics collected for intrusion detection

Statistic	Description
Connection Point	Network connection point expressed as a pair of the network element identifier and port number
Total Load/Rate	Obtain the current observed total load/rate (in bytes/s) on a link
Total Load/Latest	Obtain the latest total load bytes counter viewed on that link
Unknown Load/Rate	Obtain the current observed unknown-sized load/rate (in bytes/s) on a link
Unknown Load/Latest	Obtain the latest unknown-sized load bytes counter viewed on that link
Time seen	When the above-mentioned values were last seen
is_valid	Indicates whether this load was built on valid values

of simulated flows under attack conditions of the simulated SDN. To create minimal variance and class imbalance, we ensure that $A = N$.

Let the dataset of collected port statistics under attack conditions be denoted as X_A. The total number of samples in X_A can be symbolized as F. Here, x_{a_i} represents a single data sample and $x_{a_i} \in X_A, i = \{0, 1, ..F\}$. All the samples in X_A are labeled as *attack*.

3.2.5 Dataset Creation

After the collection of the port statistics, we must combine both datasets N and A into one dataset X, where $X = \{X_N, X_A\}$. This combination ensures that all data is aggregated into one source, from which the ML algorithm can begin learning.

3.2.6 ML Classifier

The ML classifier is utilized to perform predictions on network flows, to see if the SDN infrastructure is under attack. For this ML procedure, X gets separated into the X_{train} and X_{test} datasets. We denote the total number of samples in X_{train} as Y, while the total number of samples in X_{test} is represented by Z and $Y > Z$. Let the ML classifier be represented by μ. The classifier is trained on X_{train} to obtain a model ν, that is saved:

$$\nu = \mu(X_{train}) \tag{3.1}$$

Once the ML classifier is trained, the values in X_{test} are parsed through to obtain the accurate prediction of the network flow γ:

$$\gamma = \nu(X_{test}) \tag{3.2}$$

The main objective behind running the testing dataset X_{test} through the trained ML classifier ν is to obtain performance metrics like precision, recall, and F-Measure scores only. The primary objective of an anomaly detector for an SDN environment is to rapidly parse through real-time network traffic. This traffic does not have any labels associated with the data samples. Hence, it is entirely contingent upon the trained anomaly detector ν to decipher between normal and attack traffic in an online setting.

3.2.7 Threat Detection

This is a primary component of the proposed TCP-SYN flood detection framework. As we do not collect any flow-level metrics for classification, we do not have access

to any host-level information like IP addresses. So, to accurately detect TCP-SYN floods and where they are originating from on the network, we depend on the Threat Detection and Threat Localization modules. These modules will also be tested on real-time traffic that have no predicted and ground-truth labels associated with it.

For Threat Detection, let the list of real-time flows be U, and the total number of flows be denoted with V. Here, u_i represents a single flow and $u_i \in U, i = \{0, 1, ..V\}$. Within every single flow u_i, multiple switches can be denoted as u_{i_s}. Within each switch u_{i_s}, some ports predict, using ν, if that flow is normal or under attack. We assume the total number of switches per flow is denoted as P_i. Let the total number of ports per-flow u_i be W_i, and list of ports for each flow u_i get denoted by $u_{i_{s_p}}$ such that $u_{i_{s_p}} \in u_i, p = \{0, 1, ...W_i\}$.

First, we must detect the number of ports that are classifying a flow as under attack. That can be determined by running each port and its corresponding port statistics through the pre-trained classifier to obtain their prediction ω_p:

$$\omega_p = \begin{cases} 0 & \nu(u_{i_{s_p}}) == Normal, \\ 1 & \nu(u_{i_{s_p}}) == Attack, \end{cases} \quad p = \{0, 1, ...W_i\} \tag{3.3}$$

The predictions of the flow then get summed up, denoted as η_p:

$$\eta_p = \sum_{0}^{W_i} \omega_p \tag{3.4}$$

We introduce a hyperparameter called *port threshold*, denoted as Φ. Φ can be defined as a threshold ratio where any value over this ratio infers that the SDN is under a potential attack. Φ allows us to possess more control over the threat detection approach by acting as a control variable. This value is determined by the system

administrator(s) who oversee the SDN infrastructure and security, and the magnitude of Φ is determined by the important parameters in the SDN topology like the number of hosts, switches, links, and ports. We see from previous equations that, for every single flow, a decision is made within every single port from all switches that the flow passes through. The port classifies, based on observed port statistics, if that flow is undergoing an attack using the trained ML model ν. The SDN is categorized as undergoing a potential TCP-SYN flood attack if the ratio of ports that classify the flow as an attack to the total number of ports that the flow passed through is greater or equal to Φ. Hence, a flow is classified as under attack if:

$$\omega_p = \begin{cases} Attack & \eta_p < \Phi, \\ Normal & \eta_p \geq \Phi, \end{cases} \quad p = \{0, 1, ...W_i\} \tag{3.5}$$

Using Equation 3.5, we can classify if a network is undergoing a TCP-SYN flood attack currently.

3.2.8 Threat Localization

For threat localization, we introduced another hyperparameter called the *switch threshold* denoted as Θ. Θ allows us to possess more control over the threat localization approach by acting as a control variable and is responsible for recognizing the switches where at least $\Theta + 1$ ports are classifying the flow as under attack. This value is also selected by the system administrator(s) in charge of the SDN infrastructure and security, and the magnitude of Θ is determined by the important parameters in the SDN topology like the number of hosts, switches, links, and ports. Every SDN has its unique topology. But the basic building blocks will primarily be the same. There will be various hosts that communicate with each other using network switches.

The main goal of Θ is to accumulate the total list and frequency of flagged switches using the trained ML model ν. The primary goal will be to localize the switch that is flagged most often, as the malicious hosts performing the TCP-SYN flood will be connected to that switch. Finding the flagged switches from a network flow can be accomplished by:

$$\zeta_s = \begin{cases} Attack & \sum_{0}^{p} \eta_p > \Theta, \\ Normal & \sum_{0}^{p} \eta_p \leq \Theta, \end{cases} \tag{3.6}$$

where ζ_s represents if a switch has flagged the flow and where $p = \{0, 1..W_i\}, s = \{0, 1..P_i\}$. Equation 3.6 allows us to detect all the switches that flag a flow as under ongoing attack. The main goal here is to find all the flagged switches to pinpoint which switches are passing attack flows through their ports. This can be used to localize the hosts in the network that are malicious. To perform the final localization of the switches to find out which hosts are malicious, we must compute the frequency with which the switches get flagged. The switch that is flagged the most ψ, will be directly connected to the malicious hosts that are performing the TCP-SYN flood attack. This can be detected by:

$$\psi = \arg\max(\sum_{0}^{V} \zeta_s), s = \{0, 1, ..P_i\} \tag{3.7}$$

Using the above methodology, we can perform anomaly detection to detect ongoing TCP-SYN flood attacks in the SDN. Additionally, we are also able to localize the attackers by flagging the closest switch that the attackers are connected to. The proposed method employs the usage of port statistics, which are customarily available

in most normal networking setups. For our experimental purpose, we are emphasizing on SDNs due to the wide relevance and usage of SDN architectures across industries, organizations, and critical infrastructures over normal networking architectures. That is why our experimentation and results are conducted using an SDN and metrics collection is regulated using OpenFlow statistics. However, this research and the proposed method can be expanded to other modern computer network architectures like SD-WAN, SASE, or other traditional networks.

3.3 Experimentation and Results

3.3.1 Setup

The simulated topology for our experimentation can be seen in Figure 3.3 which consists of 10 hosts and 12 switches. For the attack scenario, we launched a TCP-SYN flood from Hosts 1 and 2, while the intended target/victim machine was Host 8. The goal of the proposed framework is to be able to detect the ongoing attack and ultimately, localize which of the hosts are malicious by finding the switch closest to the attackers. The attack deployment was conducted using the scapy library while ML analysis, threat detection, and localization were conducted using TensorFlow, sklearn, and python.

3.3.2 Results and Analysis

For our experiments, we utilized the following initial values: $N = 50, A = 50, U = 100, \Phi = 0.3, \Theta = 3$. Our generated datasets were split using an 80%-20% ratio between training and testing data. For performance metrics, we are using Accuracy (A), Precision (P), Recall (R), and F-Measure (F) scores. The ML classifier chosen for

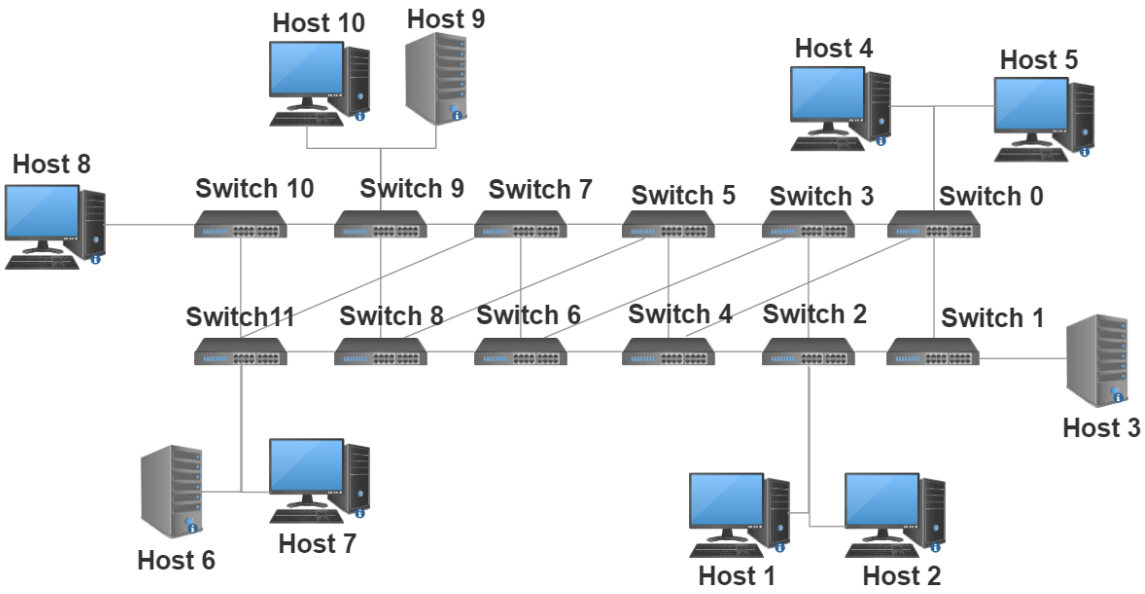


Figure 3.3: Simulated SDN Topology

Table 3.3: Performance of RF, MLP, and SVM to the generated dataset

Classifier	Accuracy
Random Forest	99.342%
Multi Layer Perceptron	51.409%
Support Vector Machine	97.852%

our experiments was a Random Forest (RF) classifier. This algorithm was chosen due to its relevance and wide usage when studying SDN anomaly detection in literature.

First, we look at the performance that is being achieved by various ML classifiers on our collected port statistics dataset. The classifiers tested were RF, Multi-Layer Perceptron (MLP), and Support Vector Machine (SVM) due to their relevance and wide usage when studying SDN anomaly detection in literature. The model with the highest accuracy can be correlated with better performance during TCP-SYN flood attack detection. Table 3.3 illustrates the performance of the algorithms when running the generated datasets. We notice that the RF gives us the best performance followed by the SVM, while the MLP provides us with mediocre performance for our dataset.

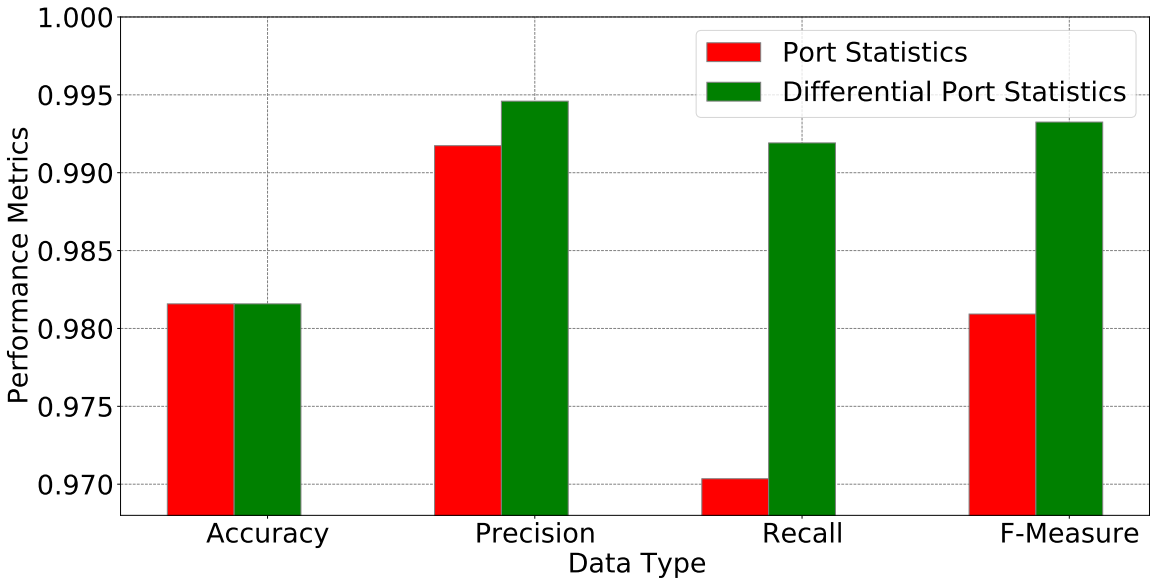


Figure 3.4: ML performance on port statistics and combinational port statistics

Next, we analyze the impact of differential port statistics on the performance of our RF anomaly detector. For this, we observe the performance achieved by our ML algorithm when facing normal port statistics versus the combination of both normal and differential port statistics, which we also call combinational port statistics. Figure 3.4 shows us the performance achieved on just port statistics and combinational port statistics. The ML classifier achieves good performance on both data types, but the algorithm performs better on combinational port statistics as it attains better P, R, and F scores. This can be attributed to combinational port statistics being collected and reported as normal statistics and their magnitude differences in a time interval. This prevents the aggregation of metrics over time, which may be occurring in normal port statistics. This also allows a more fine-grained analysis of network flows from the port level and provides better performance metrics. Superior performance is preferred as this anomaly detector will be placed to analyze live/real-time traffic which has no labels.

For a comprehensive analysis of the degree of influence, our captured network met-

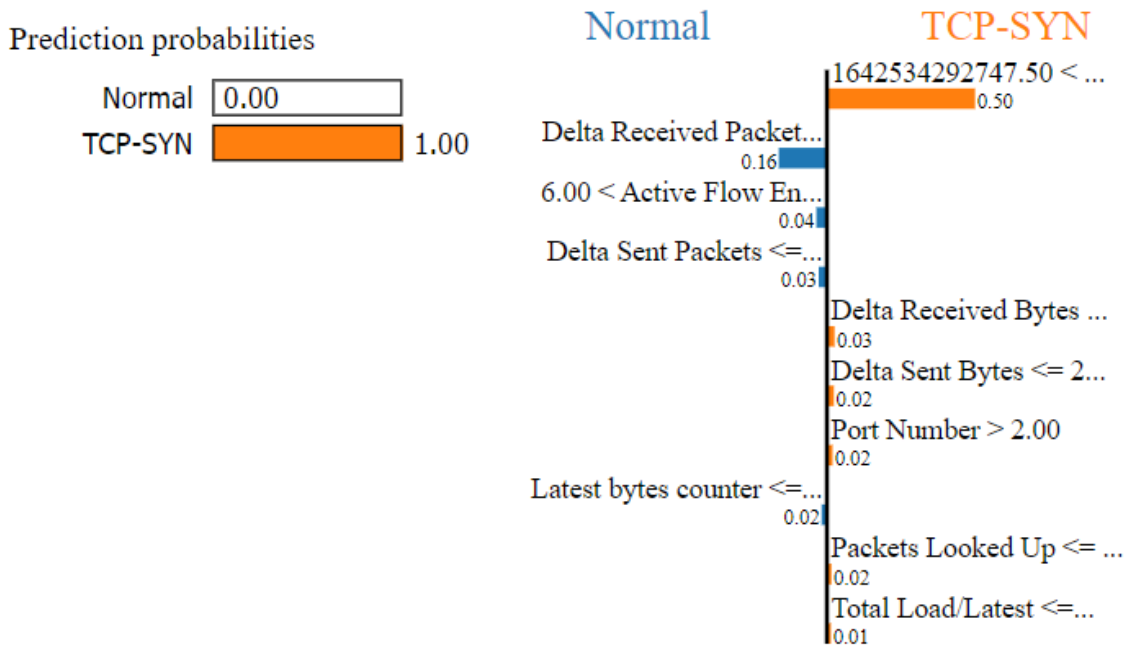


Figure 3.5: Using LIME to find most influential features

rics exhibit over the classification performance of our RF model, we also analyze the explainability of the model. This is conducted by analyzing the predictions of the model, on a random single testing sample, using the Local Interpretable Model-agnostic Explanations (LIME) framework [79] and the results are provided in Figure 4.2. This helped provide explainability and interpretability to our TCP-SYN detection framework. We observe that, for the sample, the model predicted it as undergoing a *TCP-SYN flood* attack. The main dataset features that were influential in the prediction were port statistics like *Received Packets*, *Sent Packets*, *Received Bytes*, and *Sent Bytes*. From this, we can see the impact these metrics have on the detection of TCP-SYN attacks detection. This also reinforces customary knowledge that network metrics revolving around byte/packet transmission, along with network throughput, are key in detecting if a DDoS attack is actively exhausting network resources.

Subsequently, we observe the performance of our proposed method by varying the number of normal and attack flows used to generate the dataset. For this, we obtain

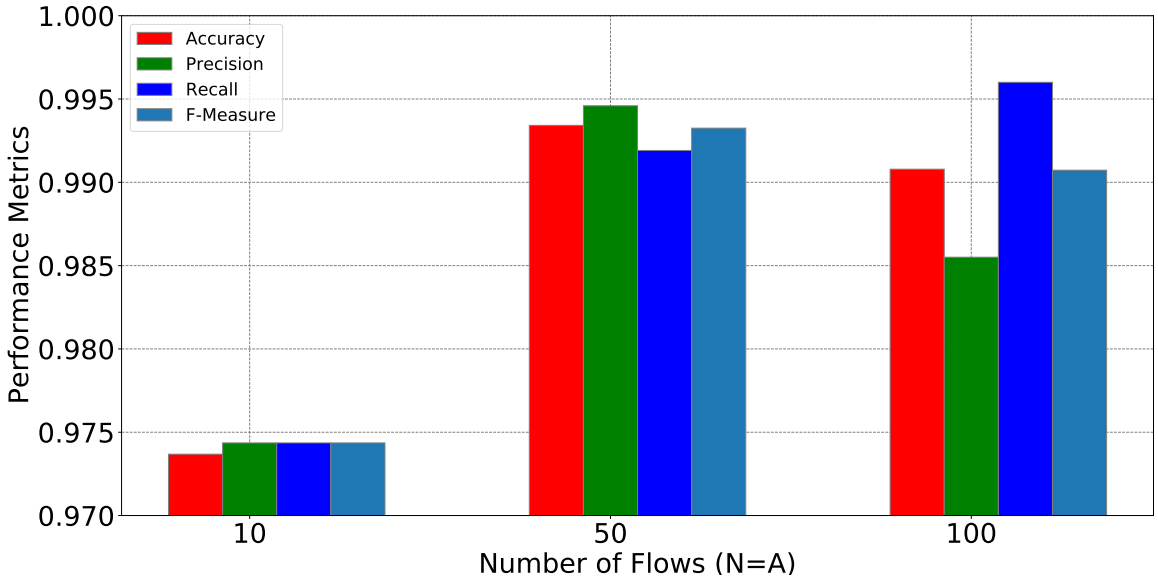


Figure 3.6: ML performance on varied number of flows for N and A

performance metrics while we are varying the values of $N = A = \{10, 50, 100\}$. The results of this experiment are illustrated in Figure 3.6. We observe that $N = A = 10$ provided decent A, P, R, and F scores, but those scores rose when $N = A = 50$. The magnitudes continue to be steady when $N = A = 100$. At $N = A = 10$, the model was not optimally learned. It achieved optimal learning at $N = A = 50$ and continued to hold the same performance till $N = A = 100$.

The above results were conducted using the testing dataset from our generated TCP-SYN flood detection dataset. However, the primary aim of this research is to propose a framework that can conduct anomaly detection on real-time data. The generated ML algorithm can be placed in the SDN controller, and it can observe live traffic to provide rapid inferences to protect the SDN infrastructure from TCP-SYN flood attacks. The remaining analysis is conducted using real traffic from the SDN infrastructure while it is undergoing the attack.

Next, we observe the performance of the proposed approach under varying values of $\Phi = \{0, 0.05, ...0.35\}$. The goal here is the observe the performance metrics when

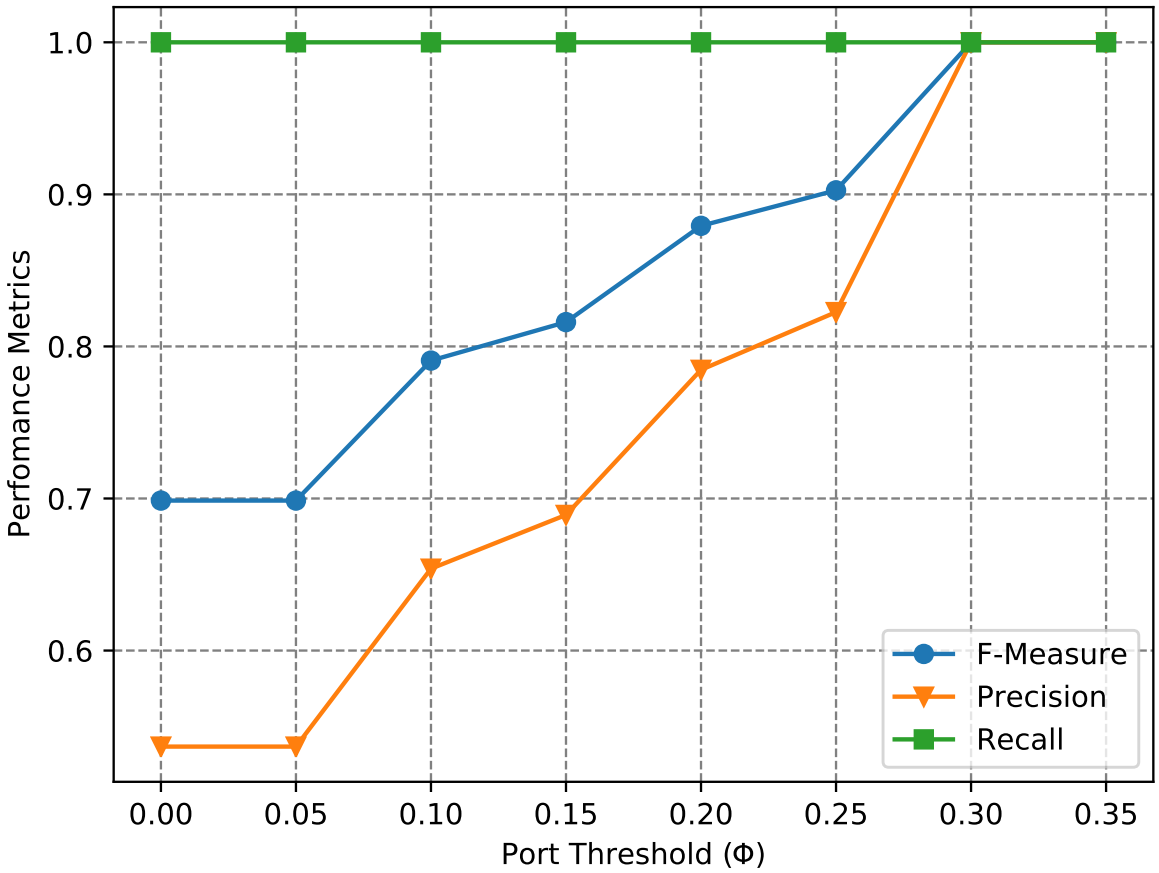


Figure 3.7: ML performance on varied values for Φ

the value of Φ is diversified. The analysis is illustrated in Figure 3.7. We can see that P and F are low at the lower values of Φ. As the value of Φ increases, the values of P and F progressively increase till $\Phi = 0.3$. The value of R remains constant throughout. This suggests that lower values of Φ restrict our ability to optimally detect TCP-SYN floods, as many normal flows get flagged as under attack, leading to high false positivity rates. This can limit the ability of the framework to optimally detect ongoing TCP-SYN floods. Higher values of Φ provide more balance to the framework and lead to better performance.

Similarly, we observe the performance of the proposed approach under varying values of $\Theta = \{0, 0.5, ...4\}$. The goal here is to observe the number of flagged switches when

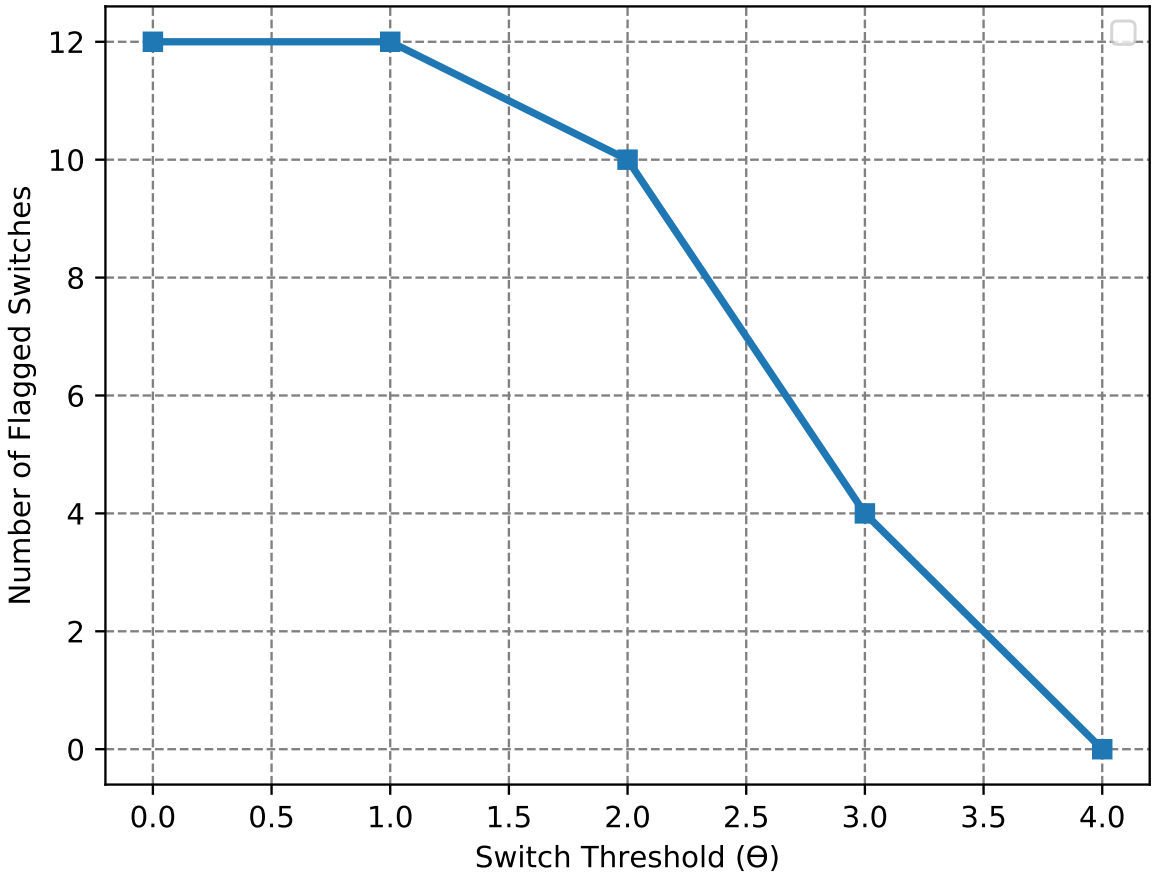


Figure 3.8: Detecting Number of Flagged Switches by varying Θ

the value of Θ is diversified and to identify the value of Θ that minimizes the number of flagged switches. The analysis is illustrated in Figure 3.8. We observe that lower values of Θ yield a high number of flagged switches. That maximizes the number of switches through which a flow under attack passes and does not help identify and localize the malicious hosts, which is an optimization problem. As the value of Θ increases, it provides more control to the anomaly detector to appropriately detect an attack and localize the malicious hosts as it increases the threshold for how a switch becomes flagged. $\Theta = 3$ provides the best performance as it identifies the minimal number of flagged switches before that value becomes null, which is 4. This means that the malicious hosts must be connected to these 4 switches.

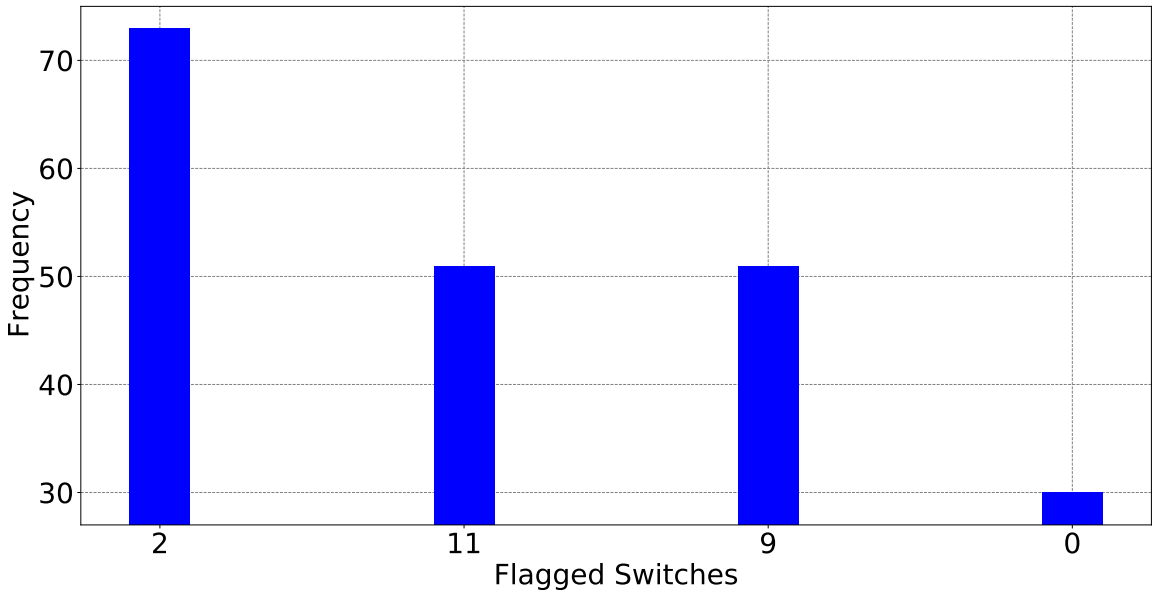


Figure 3.9: Checking the most frequently flagged switches over U number of real-time flows

Lastly, we perform the final step of threat localization in which we compute the frequency with which certain switches are flagged over the total number of real-time flows being observed. This can be observed in Figure 3.9. Here, we observe that over U number of real-time flows, Switches 2, 11, 9, and 0 are flagged the most times with frequencies of 72, 51, 51, and 30, respectively. Switch 2 is flagged the highest number of times, which indicates that the malicious hosts must be closely connected to this switch. This is confirmed as we can observe, from the topology diagram in Figure 3.3, that the malicious hosts 1 and 2 are both directly connected to Switch 2. Using the proposed methodology, we can identify that the SDN infrastructure was undergoing a TCP-SYN flood attack using port statistics. On top of identification, we are also able to localize the malicious hosts 1 and 2 using our Threat Localization module.

Through these research efforts, we create a TCP-SYN flood detection framework for SDN that can be catered to dynamic organizational topologies. The proposed technique primarily uses network port statistics from core devices like switches and

routers in the topology to monitor the network's security. Evaluated results showcase that ML classifiers like RF can differentiate normal traffic from malicious traffic with high performance. This research highlights the ability to perform binary classification using network port statistics. The subsequent step would be to expand this data collection to other attack categories for multi-class classification, by generating a new NIDS dataset with various attack labels.

Chapter 4

University of Nevada, Reno - Intrusion Detection Dataset

To reduce the limitations of imbalance classes and labels and a lack of dataset numbers, we propose the University of Nevada - Reno Intrusion Detection Dataset (UNR-IDD) that provides researchers with a wider range of samples and scenarios. The proposed dataset utilizes network port statistics for more fine-grained control and analysis of intrusions. To promote balance within the classes, the dataset ensures that each label is minimally variable from the rest in the number of samples. The proposed dataset also includes topological information and can be effectively trained in a short time interval due to its dimensionality. Using different ML algorithms, we provide a benchmark to show efficient performance for both binary and multi-class classification tasks. The chapter further explains the intrusion detection activities rather than providing a generic black-box output of the ML algorithms.

4.1 Usage of ML Models for NIDS

The usage of ML for NIDS has gained traction in the last decade as various open-sourced datasets have been proposed and established. Customary NIDS datasets include NSL-KDD [12], UNSW-NB15 [17], and CIC-IDS-2018 [13]. However, a common problem that has been identified with many of these datasets is inadequate modeling of tail classes [80]. Tail classes refer to certain labels with limited samples compared to other labels, leading to poor performance when fitting the ML model. Researchers have been looking at methods to address this issue of tail classes. Commonly investigated methods include undersampling and oversampling. However, oversampling increases the size of the dataset, increasing training time, memory, and complexity. Correspondingly, undersampling can reduce data samples from the majority classes, affecting the overall performance of prediction models [81]. It can also be argued that because these techniques manipulate existing data samples, they do not add any new insights.

Another limitation of the current datasets is that they mostly depend on flow-level statistics. This can limit the transferability of the NIDS solutions to other network configurations since the flow statistics depend on the network topology and traffic characteristics. In addition, many of these datasets contain redundant features that play no impact in being able to detect potential network intrusion types. Depending on the ML approach being utilized, these features can also increase the cardinality of the dataset, leading to increased training and inference times. Finally, some existing datasets suffer from incomplete or missing records. These records or samples must be ignored or dropped from the overall dataset, which leads to sub-optimal performance. Addressing the above-mentioned limitations is vital to ensuring that proper NIDS are being developed to adequately protect networking infrastructures from intrusions.

We propose the University of Nevada - Reno Intrusion Detection Dataset (UNR-IDD). The main difference between UNR-IDD and existing NIDS datasets is that UNR-IDD consists primarily of network port statistics. These refer to the observed port metrics recorded in switch/router ports within a networking environment. The dataset also includes delta port statistics which indicates the change in magnitude of observed port statistics within a time interval. Compared to datasets that primarily use flow statistics, UNR-IDD can provide a more fine-grained analysis of network flows as decisions are made at the port level versus the flow level. This can lead to rapid identification of potential intrusions. We also address the limitation of tail classes. Our dataset ensures that there are enough samples for ML classifiers to achieve high F-Measure scores, uniquely. Our dataset also ensures that there are no missing network metrics. The proposed observable dataset metrics can be obtained through most networking architectures. However, for our testbed, we have used a software-defined network (SDN) environment, due to the wide relevance of and usage of SDN architectures across industries, organizations, and critical infrastructures. In summary, the main contributions of our dataset include:

- The primary usage of port and delta port statistics to model the various intrusions in the dataset.
- Confirms enough samples to ensure high-performance metrics across all tail classes.
- Ensuring all data samples are filled and there is no missing data in the dataset.
- Provides performance comparison of intrusion detection models when using UNR-IDD and other NIDS datasets.

4.2 UNR-IDD Dataset

We setup up our testbed using an SDN simulation environment due to the ease of usability and implementation. It also ensures that the dataset is not dependent on any static topology and can be configured to reproduce the network activity of various topologies. Following the testbed configuration, we perform flow simulations within the SDN topology to replicate appropriate functionality. During these flow simulations, desired network statistics are collected, under normal and attack conditions.

4.2.1 Testbed Configuration

To set up the testbed, we use Open Network Operating System (ONOS) SDN controller (API version 2.5.0) alongside Mininet for the network topology generation. ONOS uses the Open Service Gateway Initiative (OSGi) service component at runtime for the creation and activation of components and auto-wiring components together, making it easy to create and deploy new user-defined components without altering the core constituents. Mininet creates the desired virtual network, and runs a real kernel, switch, and application code, on a single machine, thereby generating a realistic testbed environment. We also implemented our ONOS application to collect network statistics. Specifically, we gathered delta and cumulative port, flow entry, and flow table statistics for each connected Open vSwitch in the Mininet topology. We created a custom Mininet topology using Mininet API (version 2.3.0) with Open Flow (OF) 14 protocol deployed to the switches. The generated SDN topology for our experiments is illustrated in Figure 4.1, which consists of 10 hosts and 12 switches.

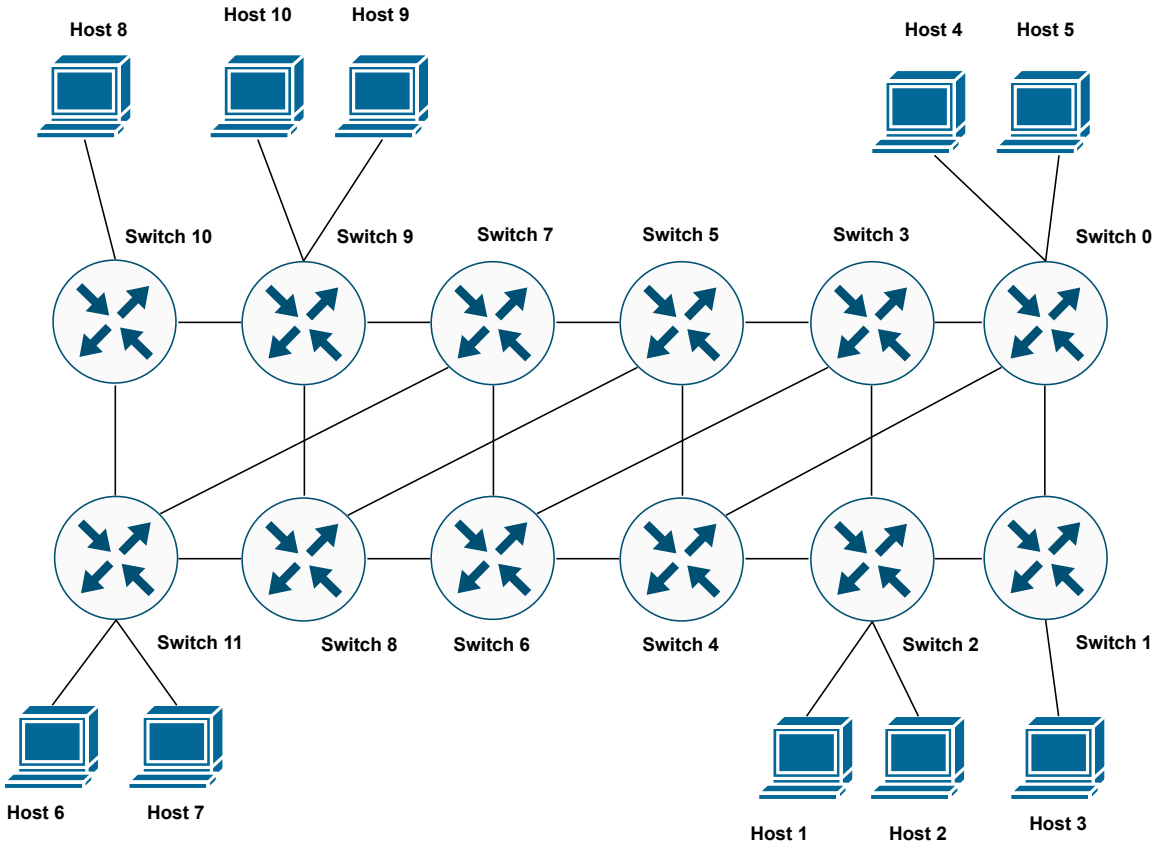


Figure 4.1: Simulated SDN topology

4.2.2 Flow Simulation

IPerf is used to create TCP and UDP data streams simulating network flows in virtual and real networks using dummy payloads. By using the Mininet API and IPerf, we created a Python script to simulate realistic network flows. Once every 5 seconds, we initiated Iperf traffic between a randomly chosen source-destination host pair with a bandwidth of 10 Mbps and a duration of 5 seconds. These values must be carefully chosen as they are dependent on the number of nodes, hosts, switches, and geographical spread of the simulated network. We then simulate flows under normal and intrusion conditions to gather data in every scenario. To ensure that each normal and intrusion category is minimally variable and adequately represented, we execute the same number of flows while simulating each scenario.

4.2.3 Data Collection

We create a custom application to collect and log the available statistics that are captured periodically (once every 5 seconds) from OpenFlow (OF) switches. The statistics are collected utilizing *OFPPortStatsRequest* and *OFPPortStatsReply* messages between controller and switches. The delta port statistics are computed on the controller side by taking the difference between the last two collected data instances. We create a key-value map of this data by gathering it from the data storage service, using the "Device Service" API provided by ONOS. After this, we logged the map of the collected statistics to a Javascript Object Notation (.json) file with a name $N_i.json$. Table 4.1 shows the collected port statistics and their descriptions per port on every switch in the simulated SDN. These statistics relay the collected metrics and magnitudes from every single port within the SDN when a flow is simulated between two hosts. Table 4.2 illustrates the collected delta port statistics and their descriptions per port on every switch. These delta statistics are used to capture the change in collected metrics from every single port within the SDN when a flow is simulated between two hosts, at a time interval of 5 seconds. Additionally, we also collect some flow entry and flow table statistics to work in conjunction with the collected port statistics as seen in Table 4.3. These metrics provide information about the conditions of switches in the network and can be collected in any network setting.

4.2.4 Intrusions

The following intrusions are simulated in the data collection phase:

- **TCP-SYN Flood**: A Distributed Denial of Service (DDoS) attack where attackers target hosts by initiating many Transmission Control Protocol (TCP) handshake processes without waiting for the response from the target node. By

Table 4.1: Port statistics collected for every port on every switch

Port Statistic	Description
Received Packets	Number of packets received by the port
Received Bytes	Number of bytes received by the port
Sent Packets	Number of packets sent by the port
Sent Bytes	Number of bytes sent
Port alive Duration	The time port has been alive in seconds
Packets Rx Dropped	Number of packets dropped by the receiver
Packets Tx Dropped	Number of packets dropped by the sender
Packets Rx Errors	Number of transmit errors
Packets Tx Errors	Number of receive errors

doing so, the target device's resources are consumed as it must keep allocating some memory space for every new TCP request.

- **Port scan**: An attack in which attackers scan available ports on a host device to learn information about the services, versions, and even security mechanisms that are running on that host.

- **Flow Table Overflow**: An attack that targets network switches/routers where attacks compromise the functionality of a switch/router by consuming the flow tables that forward packets with illegitimate flow entries and rules so that legitimate flow entries and rules cannot be installed.

- **Blackhole**: An attack that targets network switches/routers to discard the packets that pass through, instead of relaying them on to the next hop.

- **Traffic Diversion**: A attack that targets network switches/routers to reroute the direction of packets away from their destination, to increase travel time, and/or to spy on network traffic through a man-in-the-middle scenario.

These intrusion types were selected for this dataset as they are common cyber attacks that can occur in any networking environment. Also, these intrusion types cover attacks that can be launched on both network devices and end hosts.

Table 4.2: Delta port statistics collected for every port on every switch

Delta Port Statistic	Description
Delta Received Packets	Change in number of packets received by the port
Delta Received Bytes	Change in number of bytes received by the port
Delta Sent Packets	Change in number of packets sent by the port
Delta Sent Bytes	Change in number of bytes sent
Delta Port alive Duration	Change in the time port has been alive in seconds
Delta Packets Rx Dropped	Change in number of packets dropped by the receiver
Delta Packets Tx Dropped	Change in number of packets dropped by the sender
Delta Packets Rx Errors	Change in number of transmit errors
Delta Packets Tx Errors	Change in number of receive errors

4.2.5 Labels

This dataset can be broken down into two different ML classification problems: binary and multi-class classification. The goal of binary classification is to differentiate intrusions from normal working conditions. Binary classification can estimate if a network is under attack but does not provide any information about the type of attack. The labels for binary classification in UNR-IDD are illustrated in Table 4.4.

The goal for multi-class classification, however, is to differentiate the intrusions not only from normal working conditions but also from each other. Multi-class classification helps us to learn about the root causes of network intrusions. The labels for multi-class classification in UNR-IDD are illustrated in Table 4.5.

Table 4.3: Flow statistics collected

Statistic	Description
Connection Point	Network connection point expressed as a pair of the network element identifier and port number
Total Load/Rate	Obtain the current observed total load/rate (in bytes/s) on a link
Total Load/Latest	Obtain the latest total load bytes counter viewed on that link
Unknown Load/Rate	Obtain the current observed unknown-sized load/rate (in bytes/s) on a link
Unknown Load/Latest	Obtain the latest unknown-sized load bytes counter viewed on that link
Time seen	When the above-mentioned values were last seen
is_valid	Indicates whether this load was built on valid values
TableID	Returns the Table ID values
ActiveFlowEntries	Returns the number of active flow entries in this table.
PacketsLookedUp	Returns the number of packets looked up in the table.
PacketsMatched	Returns the number of packets that successfully matched in the table
MaxSize	Returns the maximum size of this table.

4.3 Experimentation, Results, and Analysis

To showcase the functionality of our proposed dataset, UNR-IDD, we run evaluations using the dataset and demonstrate the performance achieved. We illustrate results across multiple scenarios by varying the classification type, the ML algorithms, and other prominent NIDS in the literature. For performance evaluation, we are using accuracy (A), precision (P), Recall (R), and F-Measure (F) scores as the metrics. In addition, we are also using the mean scores for precision (P_μ), recall (R_μ), and f-measure (F_μ) across all label types in the datasets during multi-class classification. This will provide us with the mean performance achieved on the dataset for all label

Table 4.4: Multi-class Classification Labels

Label	Description
Normal	Normal Network Functionality
Attack	Network Intrusion

Table 4.5: Multi-class Classification Labels

Label	Description
Normal	Normal Network Functionality
TCP-SYN	TCP-SYN Flood
PortScan	Port Scanning
Overflow	Flow Table Overflow
Blackhole	Blackhole Attack
Diversion	Traffic Diversion Attack

types.

First, we observe the performance that is being achieved from the proposed dataset on both binary classification and multi-class classification scenarios. For this, we are utilizing a Random Forest (RF) as our ML algorithm. We chose to use an RF due to its relevance and wide usage when studying NIDS in literature. The binary classification performance and multi-class classification performance achieved from the dataset can be observed in Table 4.6 and Table 4.7, respectively. We can see that in Table 4.6, both label types are providing a performance of 1.0 for P, R, and F scores. This means that the dataset is linearly separable, and the RF has no trouble detecting if a given network flow is normally functioning or under any potential intrusion. Similarly, in Table 4.7, we see the RF achieving excellent performance as well. All the label types achieve high scores for P, R, and F scores. These can be attributed to the fact that each label type has adequate representation and enough data samples thereby, makes them linearly separable from each other. This makes it easier for ML classifiers to recognize them individually, which does not deteriorate performance. These results demonstrate one of the main contributions of this proposed dataset, which is ensuring

Table 4.6: Binary Classification Performance

Label	***P***	***R***	***F***
Attack	1.0	1.0	1.0
Normal	1.0	1.0	1.0

Table 4.7: Multi-class Classification Performance

Label	***P***	***R***	***F***
Blackhole	0.98	0.98	0.98
Diversion	0.99	0.97	0.98
Normal	1.0	1.0	1.0
Overflow	0.98	0.76	0.86
PortScan	0.91	0.94	0.92
TCP-SYN	0.91	0.92	0.92

that all labels have enough data samples to achieve high performances, individually and as a collective.

Next, we observe the performance that is being achieved from the proposed dataset using multiple ML algorithms: RF, Multi-layer Perceptron (MLP), Support Vector Machine (SVM), Bagging Classifier (BC), KNeighborsClassifier (KNC), and AdaBoost Classifier (ABC). We chose to use these algorithms due to their relevance and wide usage when studying NIDS in literature, along with their ease of accessibility through the sklearn libraries. The performance achieved by the algorithms on UNR-IDD is provided in Table 4.8. We can see that the best performance is achieved by the RF and BC classifiers as they achieve the near-optimal P_μ, R_μ, and F_μ scores. This is followed by the SVM, KNC, and ABC classifiers which achieve above-average scores, succeeded by the MLP which achieves substandard performance. These results can be associated with the fact that an RF classifier is an ensemble classifier consisting of multiple decision trees and can overcome the problem of overfitting. Similar functionality occurs in BC as it also is an ensemble classifier whose default classifier is a decision tree. Accuracy and variable importance are also automatically generated in

Table 4.8: Multi-class Classification Performance using Machine Learning Algorithms

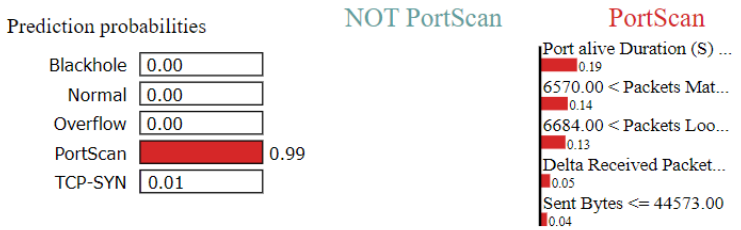
Algorithm	P_μ	R_μ	F_μ
SVM	0.89	0.79	0.81
MLP	0.59	0.54	0.54
RF	0.96	0.92	0.94
BC	0.95	0.93	0.94
KNC	0.79	0.75	0.77
ABC	0.69	0.59	0.55

RF and BC [82], compared to the other classifiers observed.

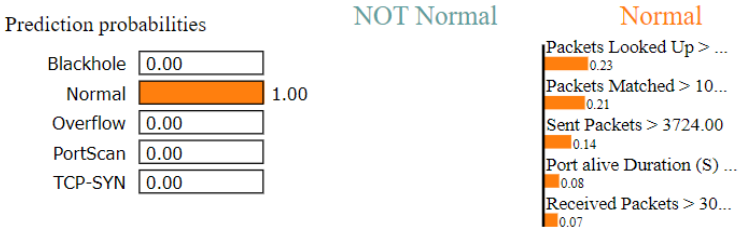
We also analyze the explainability of the RF model across various labels for a more comprehensive analysis. This is conducted by analyzing the predictions of the model, on testing samples with varying predicted labels, using the Local Interpretable Model-agnostic Explanations (LIME) framework [79]. These results are provided in Figure 4.2. We exhibited another proposed contribution to the UNR-IDD dataset, which incorporates the primary usage of port and delta port statistics to model the various intrusions in the dataset.

Lastly, we compare the performance that is being achieved from the proposed UNR-IDD dataset to two open-sourced NIDS datasets: NSL-KDD and CIC-IDS-2018. These two are established NIDS datasets that are frequently used for researching network intrusion and anomaly detection. We use the same RF classifier for all three datasets and their performance is evaluated using A, P_μ, R_μ, and F_μ. We also introduce a new metric, min F, which represents the minimum F-Measure score that is achieved for any label in that dataset. This metric can highlight the variability between F_μ and the min F value in each dataset.

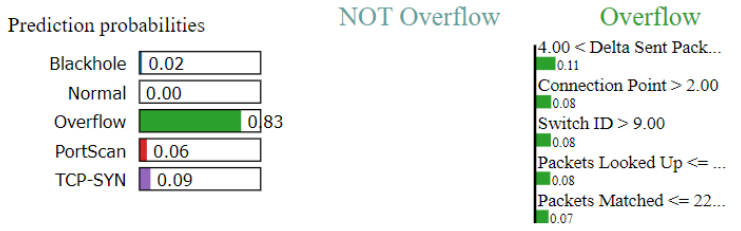
We observe the impact of the dataset sizes on the training times. We provide this comparison in Table 4.9 where we provide the dataset dimensions (samples and features) for all the datasets. As observed, CIC-IDS-2018 contains 6,291,450 samples



(a) Explanations for PortScan



(b) Explanations for Normal



(c) Explanations for Overflow

Figure 4.2: LIME Explanations for UNR-IDD

and 80 features, NSL-KDD contains 125,974 samples and 43 features, and UNR-IDD contains 37,412 samples and 34 features. This emphasizes that the proposed UNR-IDD dataset has the lowest operational footprint out of the observed NIDS datasets. For evaluation, we note both the Overall Training Time (OTT) in seconds (s) and Normalized Training Time (NTT) in milliseconds (s). We observe that UNR-IDD, due to its smaller dimensions, takes less time to train at 4.03s than NSL-KDD at 9.84s and much less time to train than CIC-IDS-2018 at 9056.23s. NTT can be defined as the time taken to train one sample and can be computed by dividing the OTT by the number of samples. We notice that the NTT is least for the NSL-KDD with 0.078 ms/sample, with UNR-IDD achieving comparable performance to it with 0.107 ms/sample. This can be attributed to NSL-KDD having only 3 categorical

Table 4.9: Training Analysis of the NIDS Datasets

Dataset	Samples	Features	OTT (s)	NTT (ms)
UNR-IDD	37,412	34	4.03	0.107
NSL-KDD	148,517	43	9.84	0.078
CIC-IDS-2018	6,291,480	80	9056.23	1.439

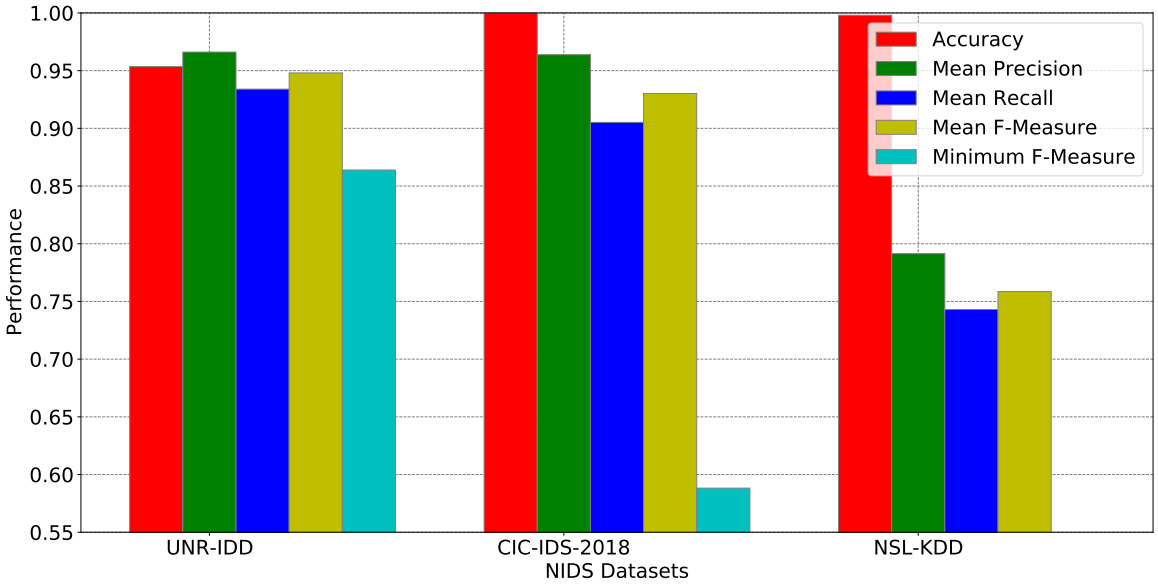


Figure 4.3: Performance Analysis of UNR-IDD, NSL-KDD, and CIC-IDS-2018 datasets

features per sample, whereas UNR-IDD contains 5 categorical features per sample. CIC-IDS takes the most NTT out of the three datasets with 1.439 ms/sample. From the perceived OTT and NTT, we observe that the UNR-IDD provides the quickest OTT and very comparable NTT. This signifies that using UNR-IDD, a competent ML model for intrusion detection can be generated much quicker as it can train the overall dataset the fastest while training each sample with a comparable speed to that of the other observed NIDS datasets.

In Figure 4.3, we observe that both the NSL-KDD and CIC-IDS-2018 datasets achieve 99% A scores. Relatively, the UNR-IDD dataset achieves comparable performance with an A score of 95%. This can be attributed to the UNR-IDD dataset being

smaller, overall, than NSL-KDD and significantly smaller, overall, than CIC-IDS-2018 in terms of both the number of samples and features. We also note that the P_μ score for the UNR-IDD dataset is equivalent to that of CIC-IDS-2018 at 96%. Compared to this, NSL-KDD achieves a P_μ score of 79%. Similarly, the R_μ score for UNR-IDD is higher than CIC-IDS-2018 and NSL-KDD at 93% versus 91% and 74%, respectively. The most important contribution of the proposed UNR-IDD dataset is its effect on F-Measure scores. Since each tail class is adequately represented in the dataset, it achieves the highest F_μ out of all three datasets with 94% compared to 93% and 76% for CIC-IDS-2018 and NSL-KDD, respectively. Similarly, the minimum F score that is achieved across all three datasets is highest in the UNR-IDD dataset with 86%, while the CIC-IDS-2018 and NSL-KDD datasets achieve a minimum F score of 58% and 0% respectively. This highlights the UNR-IDD's prioritization of the F-Measure score as it achieves the least variability between the F_μ and the min F value observed among all the datasets.

In this research effort, we generated a new NIDS dataset called UNR-IDD that focuses primarily on the usage of network port statistics to perform both binary and multi-class classification. The dataset prioritizes representation for all tail classes and ensures that each label achieves high performance and F scores. Compared to customary datasets, UNR-IDD is a smaller operational footprint. However, the dataset still provides efficient performance across all the labels. Due to this, anomaly/intrusion detection could be trained more easily in resource-constrained network devices or low-end servers. Presently, this dataset can be publicly accessed on Kaggle [83]. The next step in this research is to generate ML classifiers capable of performing real-time network intrusion detection to protect SDNs from cyber attacks.

Chapter 5

Confident and Explainable Anomaly Detection

To diminish the prevalence of underconfident and non-interpretable ML-based NIDS classifiers, we propose an SDN anomaly detection application, Confident and Explainable Anomaly Detector (CEAD), that automatically detects malicious network flows in SDN-based network architectures. The proposed application employs a set of ML classifiers to improve the confidence score of a prediction, thereby creating improved trust in the prediction while providing interpretability to the anomaly detector. The method utilizes the Explainable Artificial Intelligence (XAI) framework to provide interpretation to predictions, in contrast to their traditional “black box" nature. It unearths network features that establish the most influence between predicted intrusion types.

5.1 Increasing Trust on ML Classifiers

A common problem associated with ML-based predictions is mediocre confidence scores. This makes ascertaining the certainty of predictions difficult. High confidence scores allow us to use an ML system in a live environment, as predictions are forecasted correctly with high probability. Therefore, an ML-based service requires, not only high accuracies in train/test datasets but also high confidence scores. The high confidence in an ML prediction assists to automate the changes in SDN flow tables that govern network policies. Another limitation of ML-based systems is the lack of explainability, as most ML research is conducted in a “black box" manner, lacking transparency and explainability. This can be problematic in network anomaly detection as users will not be able to explain the cause of the anomaly. An interpretation of the ML anomaly detector makes it easier for the system to understand which of the network features is more influential in detecting anomaly types.

The low confidence scores and lack of explainability can be catastrophic, specifically, in critical infrastructures like power, energy, and transportation systems as these systems are automated and contain minimal human interaction. Adding explainability will improve trust in the anomaly detector’s predictions; specifically, the relevance of the dataset features, the confidence of the predictions, and the justification of the results. Also, it can be used to scrutinize and deduce information from the SDN beyond a simple knowledge extraction. Hence, SDN anomaly detection models can not only classify network flows as benign or anomalous but also provide evidence for such predictions; this can help network administrators with improved analytics. They can then use this knowledge to design system policies that protect them from cyber attacks. The major contributions of this research include:

- This chapter presents an ML pipeline, called CEAD, that uses different models

to improve the confidence score of an SDN-based automated network anomaly detector.

- We use the proposed pipeline and incorporate an Explainable Artificial Intelligence (XAI) framework to interpret the type of predictions made by the proposed ML pipeline.
- We test the proposed pipeline on diverse SDN network datasets. We also uncover dataset features that maximally influence the predictions from the proposed framework.

5.2 System Model

As previously mentioned, the three main components in an SDN architecture are the infrastructure, control, and application layers. Automatic detection of anomalies and their type is done at the application layer along with the other hosted applications like network monitoring or management modules (Dynatrace or SolarWinds Network Performance Monitor). This ensures the availability of adequate resources like low latency memory and faster processor speeds to conduct online anomaly detection. Upon the detection of a particular type of anomaly, the flow tables are updated via the SDN control plane to take any action at the infrastructure or network switch layers. For example, the flow tables could be updated to block packets from a malicious IP address in case a Denial of Service (DoS) attack is detected. The presence of the XAI module enables to update of variables in the flow table like source or destination address/port, based on its influence on prediction. Another possible instance is if XAI shows that the source port is influencing the prediction, then flow tables could be updated to block it. It will enable only particular kinds of traffic to be blocked rather than benign traffic.

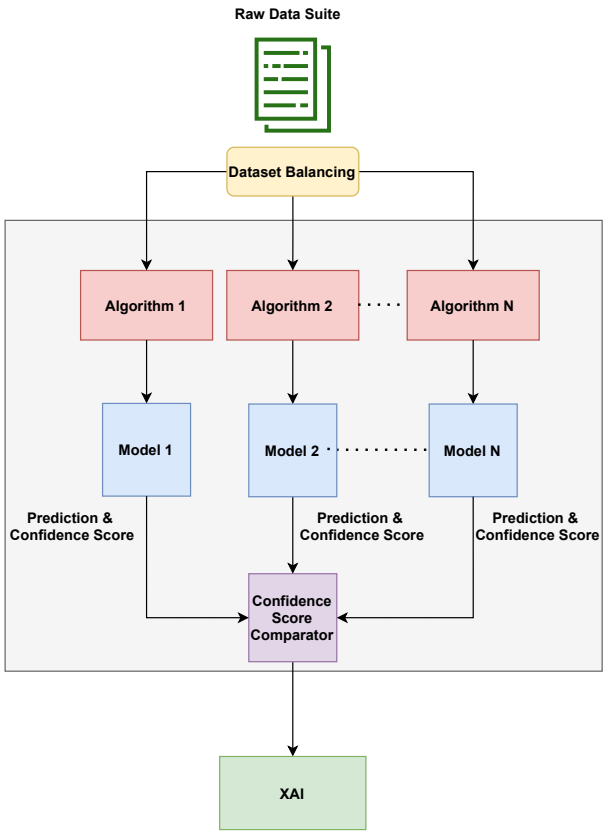


Figure 5.1: Proposed ML pipeline for anomaly detection and interpretation

Furthermore, the type of attacks and the features responsible for the attack can be stored in a log file. The log information is analyzed in an offline manner by the network administrator. Based on the collected log, long-term network policies are designed to make the system robust against different kinds of attacks.

5.3 Methodology

The proposed approach primarily consists of a high-confidence anomaly detector whose predictions are then interpreted by an XAI framework, from which the influential features are logged. Fig 5.1 illustrates our proposed methodology.

5.3.1 Raw dataset for Machine Learning pipeline development

The proposed approach has wide network applications and is agnostic to the dataset features. Let the network dataset be denoted using the variable X, consisting of E total samples. x_i represents a single data sample and $x_i \in X, i = \{0, 1...E\}$. The full data suite X is partitioned into the training X_{train} and test X_{test} sets. We assume the total number of training samples is U, while the total number of testing samples is V. After the preparation of train and test sets for the ML pipeline, we perform data balancing.

It should be noted that X_{test} is used in this chapter, primarily, to showcase testing purposes and performance evaluation. However, in a live system for which this pipeline is intended, the input to the model would be network features in real time. Based on the network features as input, the attack type and its interpretation are performed.

5.3.2 Data balancing

We balance the data in the train set to avoid overfitting due to the majority classes. Data balancing is performed using the random oversampling method. Random oversampling involves augmenting the dataset with multiple copies of minority classes to provide more representative data samples for minority classes in X_{train}. This ensures that both minority and majority classes are proportionate.

5.3.3 Machine Learning pipeline anomaly detection and classification

Following dataset balancing, we develop the ML pipeline for prediction, interpretation, and log information. The proposed pipeline combines many ML classifiers. Let an ML classifier be represented by μ and the total number of classifiers be denoted as N. Then, each classifier is trained on X_{train}, to obtain a model M_j that is saved and stored:

$$M_j = \mu_j(X_{train}), j = \{1, 2.....N\} \tag{5.1}$$

Once the ML classifiers are trained, the values in X_{test} are parsed through every trained ML classifier to obtain the predicted type of the anomaly γ_j and the confidence level ϵ_j of the associated prediction:

$$\gamma_j, \epsilon_j = M_j(X_{test}) \tag{5.2}$$

After parsing, the next step is comparing the confidence scores.

5.3.4 Confidence Score Comparator

In this module, the aim is to determine the trained ML model μ_j that provides the best confidence score for each sample $x_i \in X_{test}$. This can be conducted using the following equation:

$$\eta_i = \arg \max_{x_i \in X_{test}} (\epsilon_j), i = \{0, 1.....V\} \tag{5.3}$$

Here, η_i corresponds to the trained ML classifier μ_j that provides the best confidence score and corresponding prediction γ_j, for the testing data sample x_i.

5.3.5 Prediction interpretation using XAI

The detected attack type is interpreted using an XAI framework. For our technique, we use Local Interpretable Model-agnostic Explanations (LIME) [79]. We perform interpretability analysis for X_{test}, using each testing sample x_i's optimally trained ML classifier η_i. We introduce β that represents the fitted and simple, interpretable model around the data sample $x_i \in X_{test}$ and supplies the local explanations of the classifier η_i and operates along the same interpretable representation as x_i. The interpretable representation of x_i is denoted as x_i'. Therefore, the explanation τ is provided using:

$$\tau(x_i) = \arg\min_{\beta \in B} \Lambda(\eta_i, \beta, \pi_{x_i}) + \Omega(\beta) \tag{5.4}$$

Here, B represents the family of explainable linear models. The loss function of the framework is symbolized by Λ, while π_{x_i} denotes the locality around the value of x_i. Finally, Ω represents the complexity penalty of the local model β. The locally weighted square loss function Λ is represented by:

$$\Lambda(\eta_i, \beta, \pi_{x_i}) = \sum_j \pi_{x_i}(z_j)(\eta_i(z_j) - \beta(z_j'))^2 \tag{5.5}$$

which denotes the weighted euclidean distance where the sum iterates over a set of sample perturbed points z around x_i, $\{(z_k, z_k'), k = 1, 2....M\}$. Here, $z_k \in z$ is a perturbed original data sample while z_k' is the corresponding interpretable representation. The samples are weighed using $\pi_{x_i}(z_k)$, based on their similarity to x_i. Here, $\pi_{x_i}(z_k)$ is represented as:

$$\pi_{x_i}(z) = exp(-D(x_i, z)^2/\sigma^2) \tag{5.6}$$

where D represents a distance function with width σ. Many open-sourced XAI frameworks use a Lasso regression [84], as the distance function D, to prompt any sparsity in explanations and depend on a hyperparameter to restrict their explorations and complexity. In this case, the hyperparameter is K which represents the number of most influential features for the local interpretable explanation. After performing Equations 5.4, 5.5, 5.6, the resultant value for each $\tau(x_i) = \{F_0, F_1, F_2, ...K\}$, where F represents the influential dataset feature for sample x_i. Lastly, we must compute the complete list of influential features within the whole testing dataset. This is computed using a frequency counter that is represented by:

$$\phi = \sum_{i=0}^{V} \tau(x_i) \tag{5.7}$$

where ϕ represents the list of the most influential dataset features or features that were most frequently encountered by the XAI algorithm when providing local interpretable explanations throughout X_{test}.

5.4 Simulation and Results

Our proposed methodology was implemented in python and utilized the TensorFlow, sklearn, imblearn, and LIME frameworks.

5.4.1 Datasets

We performed our experiments using two open-sourced SDN datasets: NSL-KDD [12] and CICIDS-2017 [13]. The features in these datasets contain network metrics that can be gathered from any SDN setup, like inter-arrival time, source and destination bytes, flow duration, transaction bytes, etc.

Dataset	Classifier	Accuracy	Mean Confidence Score
NSL-KDD	**RF**	99.82 %	0.979
	MLP	96.09 %	0.997
	SVM	93.06 %	0.595
	CEAD	97.25%	0.998
CICIDS-2017	**RF**	99.86 %	0.994
	MLP	84.44 %	0.824
	SVM	84.16 %	0.451
	CEAD	98.94 %	0.995

Table 5.1: Performance of ML classifiers using NSL-KDD and CICIDS-2017 datasets

5.4.2 Machine Learning classifiers

We selected three ML classifiers for our framework: random forest (RF), multi-layer perceptron (MLP), and support vector machine (SVM).

5.4.3 Experimentation

In our experiments, we set up the following initial values: $N = 3, K = 5, L = 1000, M = 1000$. All the datasets were split using a 70%-30% ratio between training and testing data, respectively. For performance metrics, we are using accuracy and confidence scores.

Pipeline Accuracy and Performance

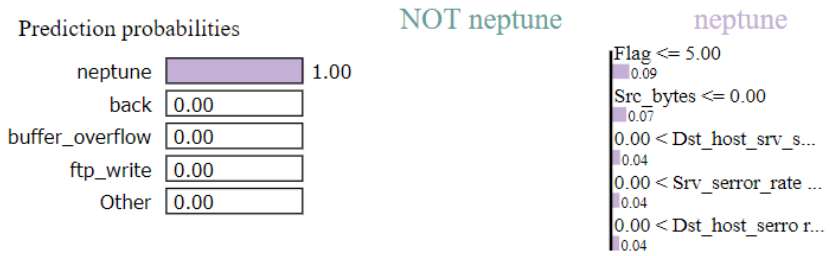
First, we look at the performance achieved by our proposed method. Table 5.1 provides the metrics and performance achieved between RF, MLP, SVM, and CEAD. From Table 5.1, we can see that SVM gives the least accuracy on both datasets, followed by the MLP classifier. The proposed CEAD classifier gives better accuracy than the SVM and MLP and provides comparable accuracy against the RF, which gives the best accuracy across both the SDN datasets. However, the CEAD achieves the best mean confidence score for their predictions. This can be attributed to the

fact that the CEAD prioritizes decisions made with the highest confidence score and associates a dataset sample with the classifier that provided the best confidence score. In contrast, RF, MLP, and SVM give lower confidence scores as the algorithms may not be able to properly differentiate between certain labels whose features are similar.

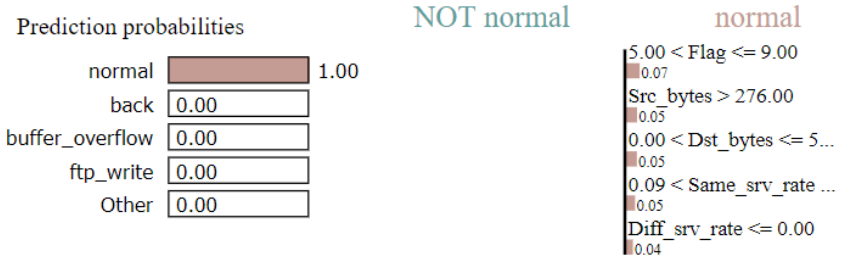
XAI Results

Next, we observe the raw data sample and the XAI explainability results obtained from the CEAD framework. This is achieved by running individual and random testing samples through the approach to get their explainability and most influential features. Due to resource constraints, we adapted our testing dataset X_{test} for this XAI analysis. Instead of performing experiments on the entirety of the testing data, we performed them on a subset of the testing data. For adapting to the new format, we introduced two new variables. L represents the number of new samples in the testing dataset and M represents the iterations for sample gathering. To generate the new testing data, we start by randomly selecting L samples from X_{test}, per iteration. In total, we run this random selection over M iterations to generate a total of $L * M$ testing samples. We, then, check the frequency of all the randomly selected X_{test} samples. From here, we select the top L most frequently occurring random data samples as our testing data. The remainder of the analysis has been conducted using this adapted testing dataset.

In Figure 5.2, we illustrate the XAI graphs for two random testing samples from the NSL-KDD dataset. We can observe, from Figure 5.2a, that the testing sample was classified as a *neptune* attack, while the most corresponding influential features were *Flag* and *Src_bytes*. Additionally, the data sample in Figure 5.2b was classified as *normal* network flow with *Flag*, *Src_bytes*, and *Dst_bytes* being the most influential features:



(a) Test Sample 1



(b) Test Sample 2

Figure 5.2: XAI Analysis of Random Testing Samples

Next, we analyze the CEAD framework's ability to discover dataset features that are influential for predictions for the whole testing dataset. Here, we look at all of the labels that are in our testing dataset and observe, collectively, which dataset features exhibited the most influence and the frequency with which these features were encountered by the XAI framework when providing interpretable explanations. This analysis was conducted using equations 5.4, 5.5, 5.6, 5.7. This information gets stored in log files and is then used by network administrators in making robust system design decisions. Figures 5.3 and 5.4 illustrate the most influential and interpretable dataset features for the NSL-KDD and CICIDS-2017 datasets.

In Figure 5.3, we can see that across all label types, *Src_ bytes* and *Flag* were the two most influential dataset features, followed by *Dst_ bytes*. *Src_ bytes* measures the number of data bytes transferred from source to destination in a single connection. *Flag* indicates the status of the connection, normal or error. *Dst_ bytes* represents the number of data bytes transferred from destination to source in a single connection.

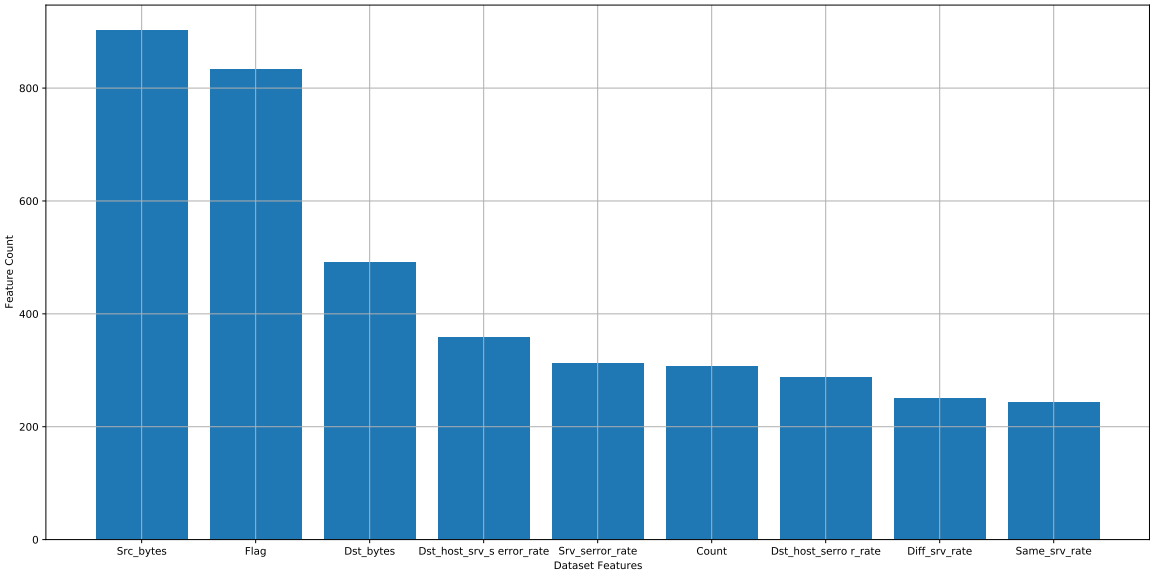


Figure 5.3: Most influential dataset features in NSL-KDD

From this, we can note that the number of data bytes or packets transmitted in a single connection within this SDN, along with the status of the connection, is the highest indicator regarding if this SDN is under attack.

In Figure 5.4, we can see that, across all label types, *Bwd Packet Length Std* and *Destination Port* were the two most influential dataset features, followed by *Packet Length Mean* and *Bwd Packet Length Min*. *Bwd Packet Length Std* measures the standard deviation size of a packet in the backward direction. *Destination Port* was flagged as an influential dataset feature. However, that is a misleading feature resulting from a dataset artifact because *Destination Port* is represented as a numeric value, instead of a categorical value. We did not address this as the goal of our experiments was to analyze the performance of our framework on default publicly available datasets. *Packet Length Mean* represents the mean length of a flow. Correspondingly, *Bwd Packet Length Min* signified the minimum size of the packet in the backward direction. From this, we can note that the average length of a network flow along with the statistics of packet lengths from destination to source is the primary and

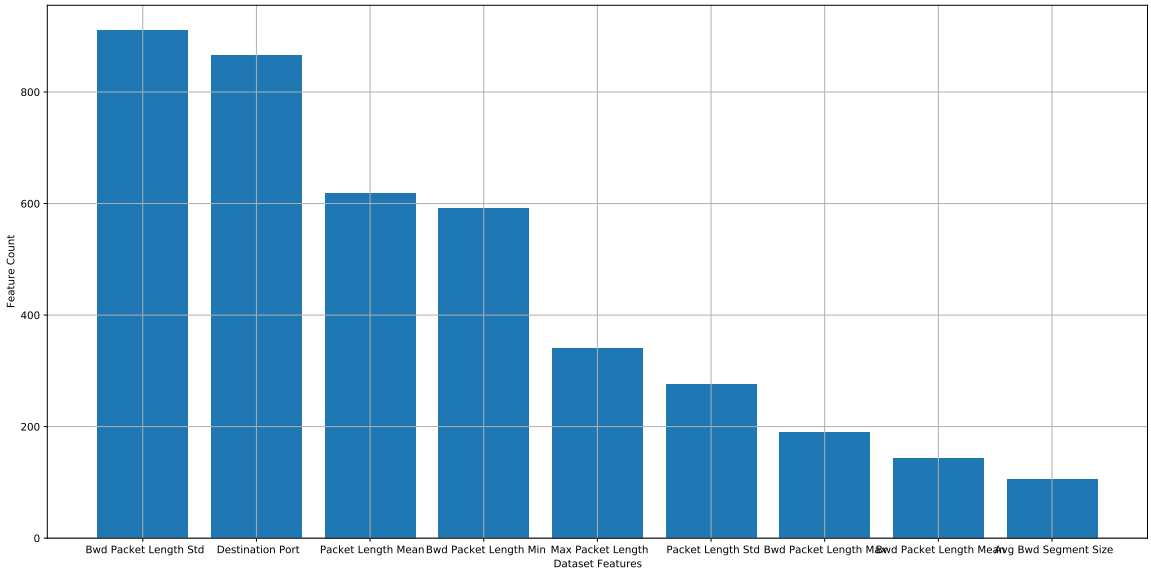


Figure 5.4: Most influential dataset features in CICIDS-2017

influential features to indicate if this SDN is under attack.

Next, we analyze the CEAD framework's ability to discover influential dataset features for predictions, for every unique label. Here, we look at each label to determine which dataset features exhibited the most influence and the frequency with which these features were encountered when providing interpretable explanations, for that label. This analysis was conducted using equations 5.4, 5.5, 5.6, 5.7 and is illustrated in Figures 5.5 and 5.6.

In Figure 5.5, we can see that, across all label types, certain dataset features are specifically influential. The feature counts for labels *neptune* and *normal* are higher than the rest of the labels, due to an increased number of samples for these two labels in X_{test}. Also, we note that all of the labels exhibit similar behavior patterns across the feature indexes, i.e certain features are counted more often than others and these features are similar across all label types. The general list of influential features for NSL-KDD is presented in Table 5.2. Also in Figure 5.5, if we observe more thoroughly, it can be noticed that certain features are more influential than others

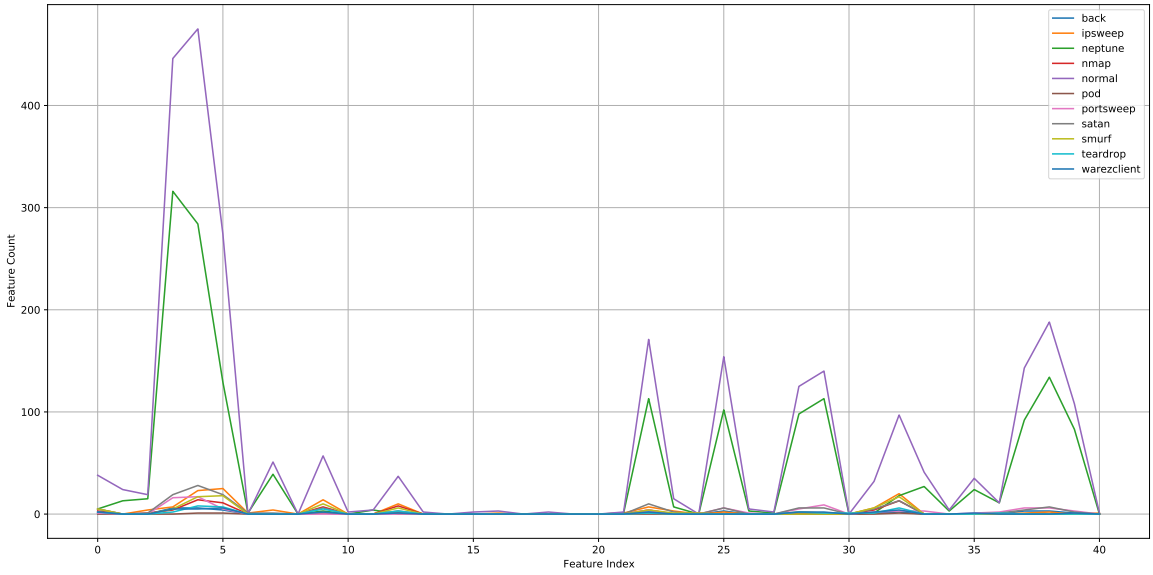


Figure 5.5: Most influential dataset features, per label, in NSL-KDD

in detecting different attack types. For example, in the NSL-KDD dataset, feature indexes 33 and 35 are more influential when detecting *neptune* or TCP SYN attacks. Correspondingly, feature indexes 9 and 12 are more influential when detecting *ipsweep* or ICMP sweep attacks on the network.

Similarly, in Figure 5.6, we can see that, across all label types, certain dataset features are specifically influential. The feature counts for label *BENIGN* are higher than the rest of the labels, due to an increased number of samples for *BENIGN* in X_{test}. Also, we note that all of the labels exhibit similar behavior patterns across the feature indexes, i.e certain features are counted more often than others and these features are similar across all label types. The general list of influential features for CICIDS-2017 is presented in Table 5.3. Also in Figure 5.6, if we observe more thoroughly, it can be noticed that certain features are more influential than others in detecting different attack types. For example, in the CICIDS-2017 dataset, feature indexes 11, 13, and 40 are more influential when detecting *PortScan* attacks and *DoS-Hulk* or Hulk flood attacks. Also, feature index 40 is another influential feature to detect *DDoS* attacks

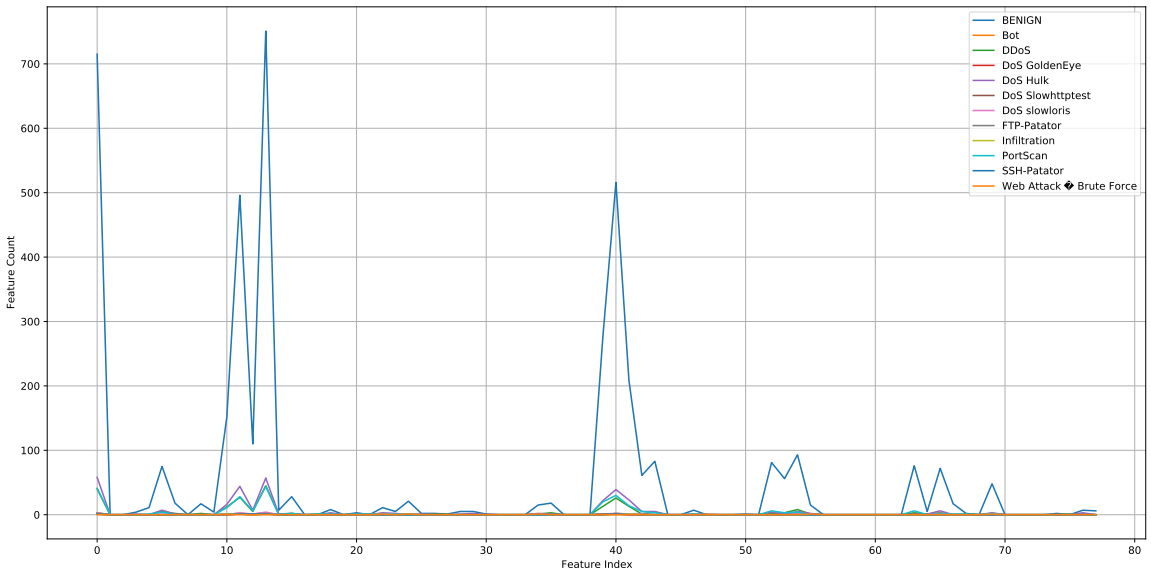


Figure 5.6: Most influential dataset features, per label, in CICIDS-2017

on the network.

From a complete overview of the analysis conducted in Figures 5.3, 5.5, 5.4, and 5.6, across two separate SDN datasets, we can see that the transmission of bytes and packets and their correlated statistics like mean, standard deviation, maximum, and minimum values, from a source to a destination, and vice-versa, can be significant indicators if an SDN is working normally or not. This correlates with contemporary knowledge that if the productivity of a network degrades over a time interval, there may be an event that is hindering its functionality. We also note from using the proposed CEAD framework across both datasets, all labels are generally influenced by similar features. Additionally, in both datasets, we have noted that specific features can be very influential in detecting specific threats to the SDN. These insights can have a positive influence as an SDN-based anomaly detection algorithm can primarily focus on these significant features that influence the detection of anomalous scenarios, thereby reducing overhead and the usage of resources.

In this research, we developed the CEAD framework that is capable of providing

Feature Index	Feature Name
2	Service
3	Flag
4	Src_bytes
5	Dst_bytes
6	Land
22	Count
25	Srv_serror_rate
28	Same_srv_rate
29	Diff_srv_rate
38	Dst_host_srv_s error_rate

Table 5.2: Most influential features, across labels, for NSL-KDD

Feature Index	Feature Name
0	Destination Port
10	Bwd Packet Length Max
11	Bwd Packet Length Min
12	Bwd Packet Length Mean
13	Bwd Packet Length Std
40	Packet Length Mean
41	Packet Length Std

Table 5.3: Most influential features, across labels, for CICIDS-2017

highly confident predictions for ML classifiers when conducting real-time network intrusion detection. This prioritization for high confidence in the predictions helps create increased trust in the predictions. This framework also employs an XAI framework to provide interpretability to the predictions to unearth the network features that maximally influence predicted intrusion types. This research provides a complete solution for intrusion detection at the core of the network. Now, it is essential to focus on protecting the edges of SDNs from anomalies that can be emblematic of cyber attacks on edge devices.

Chapter 6

Anomaly detection in SDN Edge Devices

Anomalies in time-series devices can be emblematic of cyber attacks that aim to compromise the edge of the SDN. Therefore, having a robust anomaly detection algorithm is essential for protecting edge devices from cyber attacks. For combating the restrictions of time-series data and sub-optimal performance from device/sensor noise, we propose an anomaly detection mechanism for unsupervised time-series SDN edge devices. This methodology is novel as it provides a technique that can be employed to create a supervised training approach from time-series data. This eliminates the need for generating an independent supervised dataset for time-series SDN edge devices, which is already a non-trivial task. We also highlight a methodology to process out noisy sensor data, so that the resulting dataset is minimally affected by sensor noise, leading to improved classifier performance. This ensures that anomaly detection is performed optimally in these edge devices with minimal false positive rates.

6.1 ML for the Security of Edge Devices

Modern SDN architectures and edge devices can be used from an extensive list of applications: IoT, smart homes, UAVs, vehicular networks, and smart cities. In many cases, these SDN edge devices are time-series in functionality and data collection. Oftentimes, these devices are available commercially off the shelf (COTS). Therefore, these devices do not come with adequate security monitoring capabilities. Also, due to their inexpensiveness and commercial allurement, security is not provided much priority [85]. Therefore, it is important to ensure that these SDN edge devices are protected from potentially damaging anomalous behavior that can corrupt their functionality and prevents any propagation of this behavior to the rest of the core network. We also attempt to ensure that analog sensor noise does not contaminate or compromise the optimal anomaly detection performance.

For this research, we propose an anomaly detection method using a Long-Short Term Memory (LSTM) neural network. An LSTM is employed as it is a standard architecture that is used to conduct ML analysis for time-series data, due to its capability of identifying patterns over long sequences. We use an open-source IoT sensor dataset, consisting of unsupervised sensor data. Our approach first performs data smoothing on this data to remove inherent noise from the dataset. Following this, we create a supervised dataset from the data, that is used to train the model, which is used for anomaly detection. Our main contributions include:

- Performing data smoothing on our dataset to remove sensor noise.
- Converting unsupervised time-series data into a supervised format for LSTM training.
- Modeling normal sensor behavior using an LSTM network, using robust statistical properties.

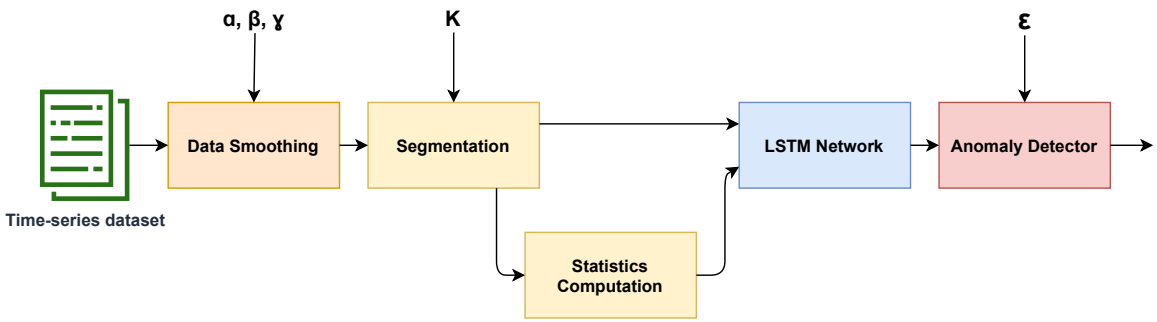


Figure 6.1: Anomaly Detection Architecture

- Mathematically modeling anomalies to check model efficacy at anomaly detection.

6.2 System Model and Methodology

The dataset consists of time-series data from an indoor environment [14]. The sensor being polled is a temperature sensor from a single location. The sensor gets polled every 31 seconds. We are considering the temperature readings from February 28th to March 21st, 2004. We use a time unit i as a discrete value and a natural number, where i represents a particular time slot. The temperature sensor readings are represented by x. The temperature reading at a particular time unit i is represented as x_i. The goal is to determine if these temperature readings x_i are anomalous or normal. The final sensor reading time is denoted as E.

The proposed anomaly detection architecture is illustrated in Fig. 6.1. The main steps of the proposed approach are highlighted in the following subsections:

6.2.1 Data Smoothing

To ensure that our data is minimally affected by noise, we perform Holt-Winters

Exponential Smoothing [86]. This method was chosen as it is an eminent method to perform smoothing on data containing seasonality and trend, which the target dataset carries. Data smoothing methods that do not take seasonality and trend into consideration are avoided as they may reduce performance. There are three operating parameters: the *data smoothing factor* denoted by α, where $0 \leq \alpha \leq 1$, the *trend smoothing factor* denoted by β, where $0 \leq \beta \leq 1$, and the *seasonal change smoothing factor* denoted by γ, where $0 \leq \gamma \leq 1$. The magnitudes of α, β, and γ, are inversely proportional to the amount of smoothing performed. These smoothing constants determine how quickly the weights of the series decay for the current observation. Values closer to 1 weigh recent observations heavily, while values closer to 0 give weight to past observations [87]. α is the primary variable for data smoothing, as it establishes the most influence on the level of smoothing. $\alpha = 1$ means that the dataset has not undergone any smoothing, while $\alpha = 0$ indicates maximal smoothing. The seasonal period of the time-series data is symbolized by ρ. The *smoothed data level* L_i at time i is given by:

$$L_i = \alpha(x_i - S_{i-\rho}) + 1 - \alpha(L_{i-1} + T_{i-1}), i = \{0, 1, ...E\} \tag{6.1}$$

The *trend* T_i of the data at time i is illustrated with:

$$T_i = \beta(L_i - L_{i-1}) + (1 - \beta)T_{i-1}, i = \{0, 1, ...E\} \tag{6.2}$$

The trend T_i represents the slope of the data trend at time i [87]. Correspondingly, the *seasonal component* S_i of the data at time i is provided by:

$$S_i = \gamma(x_i - L_i) + (1 - \gamma)S_{i-\rho}, i = \{0, 1, ...E\} \tag{6.3}$$

The *seasonal component* S_i symbolizes a weighted average between the current seasonal index, and the seasonal index of the same time during the last season [88]. Finally, the *forecasted time-series value* $\hat{x_i}$ for the data at time i can be computed using:

$$\hat{x_i} = L_{i-1} + T_{i-1} + S_{i-\rho}, i = \{0, 1, ...E\} \tag{6.4}$$

Using this technique, the time-series data becomes more representative of the normal state of the sensor, with minimal interference from noise.

Finding optimal data smoothing factor

Data smoothing should be performed optimally so that minimal information is lost from the original data. Simultaneously, data smoothing should also minimize the noise. This makes it important to select the data smoothing factor α that ensures maximum performance while minimizing the information loss from smoothing. The appropriate α can be selected by abiding by the inequality:

$$M - M' < \lambda \tag{6.5}$$

where M is the mean of the standard deviations of all the non-smooth data segments. Dataset segmentation is explained in the following subsection. M can be computed by:

$$M = \frac{\sum_{j=0}^{N} \sqrt{\frac{1}{K} \sum_{i=0}^{K-1} (x_i - \bar{x_i})^2}}{N} \tag{6.6}$$

where $\bar{x_i}$ represents the mean of the sensor values in that segment. Correspondingly, M' is the mean of the standard deviations of all the smooth data segments smoothed

by α and is computed by:

$$M' = \frac{\sum_{j=0}^{N} \sqrt{\frac{1}{K} \sum_{i=0}^{K-1} (\hat{x}_i - \bar{\hat{x}_i})^2}}{N} \tag{6.7}$$

where $\bar{\hat{x}_i}$ represents the mean of the sensor values in that segment. Finally, we introduce λ which is the *strength of anomaly*, where $\lambda \geq 0$. The magnitude of λ will further sway the value of the anomalies from the expected sensor value, meaning that a lower λ value looks more like normal sensor data, and is more difficult to detect, than a higher λ value. Optimal α is selected when the value of λ is minimized in 6.5.

6.2.2 Segmentation

Post data smoothing comes the data segmentation module. In this module, we split the smoothed data into segments of size K. The goal behind segmentation is to effectively identify anomalies in a localized context. Each segment is represented as Seg_i, where $Seg_i = \{x_i, x_{i+1}, ...x_{K-1}\}$ and i represents the starting time slot for that segment. We also assume the total number of segments to be N. Every segment gets sorted in ascending order of their values. Then, the middle 50% of the segment is extracted. We assume that the middle 50% contains no anomalies, as they are representative of normal sensor data. The other 25% on either side may or may not contain any anomalies. We represent the middle 50% values of all segments as $Seg_{i,m}$.

6.2.3 Statistics Computation

For every single segment Seg_i, we must also compute statistics for that segment. In

our case, we compute the M-estimator for every Seg_i. M-estimator tends to be robust when there are anomalies in a dataset, as they use the median in their construction. The median, in comparison to the mean, is not easily swayed by anomalies. This makes them a better fit for anomaly detection. The M-estimator for a particular segment can be computed using:

$$\sum_{i=0}^{K-1} \eta(\frac{x_i - \mu_j}{\sigma(Seg_i)}) = 0, j = \{0, 1, ...N\} \tag{6.8}$$

where $\sigma(Seg_i)$ represents a function on Seg_i which provides the initial estimate that may be mean or median. The variable μ_j, the solution of the equation, represents the M-estimator of the segment Seg_i. Lastly, η represents a real value Huber function which is denoted by:

$$\eta(x) = x.min(1, \frac{b}{|x|}) \tag{6.9}$$

where b is a constant value. The computed statistics are essential for anomaly detection further down in this process.

6.2.4 Deviation computation

We aim to convert our unsupervised dataset into a supervised one for training. After we compute the statistics, we separate the segments into training and testing segments. The goal is to compute the acceptable deviations d_j of each training segment from the μ_j of that segment. The formula to compute deviations for the training segments is given:

$$d_j = max((|\mu_j - min(Seg_i)|), (|\mu_j - max(Seg_i)|)),$$

$$i = \{0, 1, ...E\}, j = \{0, 1, ...N\} \quad (6.10)$$

However, we do not compute the deviations for the testing segments. The reason is that the testing segments are the ones that will contain anomalies, which may negatively influence direct deviation computation. Hence, we plan to predict the deviations for each testing segment, from the middle 50% of the segments.

6.2.5 LSTM Model

We are using an LSTM network for training. This network will be used to predict the deviations for the testing segments. The inputs to the network are $Seg_{i,m}$, and the corresponding labels are d_j. Fig. 6.2 illustrates the proposed LSTM.

6.2.6 Anomaly Detector

The anomaly detector is responsible for detecting anomalies in the testing segments. An essential component of the anomaly detection approach is the ϵ, where $\epsilon \geq 0$. Another important component of the anomaly detection approach is the *training segment threshold* T. These parameters allow additional control over the anomaly detector to ensure proper anomaly identification. T is computed only on the training segments, by the following equation:

$$T = \frac{\sum_{j=0}^{N} \sqrt{\frac{1}{K} \sum_{i=0}^{K-1} (\hat{x}_i - \bar{\hat{x}_i})^2}}{N} \quad (6.11)$$

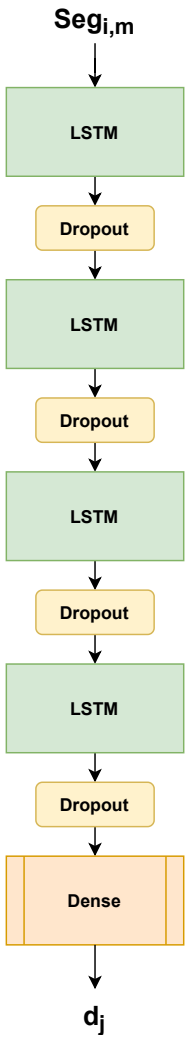


Figure 6.2: LSTM Network

where $\bar{\hat{x}}_i$ represents the mean of the sensor values in that training segment. Once the deviations of the testing data are predicted, we combine the deviations and the previously obtained statistical properties to check if a sensor value is an anomaly or normal. The anomaly test is presented as:

$$\hat{x}_i = \begin{cases} 1 & \text{if}(\hat{x}_i > \mu_j + \epsilon T d_j), \\ 1 & \text{if}(\hat{x}_i < \mu_j - \epsilon T d_j), \\ 0 & \text{otherwise}, \end{cases} \tag{6.12}$$

The above equation is computed to test if a sensor value $\hat{x}_i$, in the testing segment, is equal to 1 (anomaly) or 0 (normal).

6.3 Simulation and Results

Our approach was implemented in python, using the TensorFlow library. The dataset used was the Intel Berkeley Research Lab Dataset [14], specifically the temperature sensor data from the first sensor node. Once the network was trained, we embedded anomalies in the testing data. Then, we performed experiments to measure the efficiency of our approach.

6.3.1 Anomaly Generation

To embed anomalies in the testing data, we manipulated measured sensor values. This manipulation can symbolize a false data injection or even a physical scenario like a house fire. Our manipulation included both positively and negatively scaled anomalies, which are referred to as positive and negative anomalies, respectively, from here on. The positive anomalies have values more than the expected sensor value. Correspondingly, the negative anomalies have values less than the expected sensor value. Positive anomalies are therefore embedded using:

$$\hat{x}_i = \hat{x}_i + \lambda T \tag{6.13}$$

while negative anomalies are denoted by:

$$\hat{x}_i = \hat{x}_i - \lambda T \tag{6.14}$$

Table 6.1: Performance against positive anomalies

λ	**P**	**R**	**F**
0	5.93%	39.13%	10.30%
2	15.67%	75.41%	25.95%
4	21.64%	99.59%	35.55%
6	78.39%	99.59%	87.73%
8	97.60%	100.00%	98.79%
10	97.60%	100.00%	98.79%
12	97.60%	100.00%	98.79%
14	97.60%	100.00%	98.79%
16	97.60%	100.00%	98.79%
18	98.39%	100.00%	99.19%

The positive and negative anomalies are added to the upper and lower 25% of the segments respectively.

6.3.2 Experimentation

In our experiments, we set up the following initial values: $K = 16, \alpha = 0.3, \beta = 0.05, \gamma = 0.05, \lambda = 6, \epsilon = 5$. As our dataset has not undergone any fundamental change in values and consists of mostly steady data with random noisy fluctuations, the value of α, β, and γ should be on the lower end of the spectrum [87]. The values of λ and ϵ are not strict and must be chosen according to the statistics of the dataset being used: standard deviation, maximum and minimum values, and range. For performance metric computation, we are using Precision P, Recall R, and F-Measure F.

We experiment with our approach to see how effectively it can detect positive and negative anomalies. Table I shows performance against positive anomalies with varying $\lambda = \{0, 2, 4...18\}$. We see that the values of P, R, and F increase as λ increases. This is expected as larger anomalies are more evident and easier to detect in the

Table 6.2: Performance against negative anomalies

λ	**P**	**R**	**F**
0	6.71%	45.90%	11.71%
2	11.44%	82.38%	20.09%
4	13.65%	99.59%	24.01%
6	13.65%	99.59%	24.01%
8	14.30%	99.59%	25.01%
10	17.08%	99.59%	29.15%
12	24.16%	99.59%	38.88%
14	34.86%	100.00%	51.69%
16	45.61%	100.00%	62.64%
18	56.22%	100.00%	71.98%

context of a segment. Table II provides the performance against negative anomalies with varying λ. This table also shows that P, R, and F scores increase as the value of λ increases. The approach, comparatively, slightly under-performs in detecting negative anomalies, as the dataset is solely filled with positive values and the value of T is low, compared to the range of the dataset. Hence, it takes a higher magnitude of λ to show effective performance in detecting negative anomalies.

Subsequently, we showcase the performance of the approach by varying the parameter $\epsilon = \{0, 1, ...16\}$. Fig. 6.3 illustrates the performance of the anomaly detection method as the value of ϵ is varied. We can see changes in the performance metrics when the parameter ϵ is gradually increased. An inverse correlation is noted between the observed P and R. The value of R is higher at low values of ϵ, while the value of P is higher at high values of ϵ. We also note that the rate of increase in P and the rate of decrease in R appears to be approximately the same. The best balance between P and R is observed at $\epsilon = 6$, where the $F = 0.9$.

Next, we analyze the effect of data smoothing on the performance of the anomaly detection approach, by varying $\alpha = \{1, 0.6, 0.2\}$. Fig. 6.4 illustrates the P, R, and F values, when the data has undergone no smoothing ($\alpha = 1$), moderate smoothing

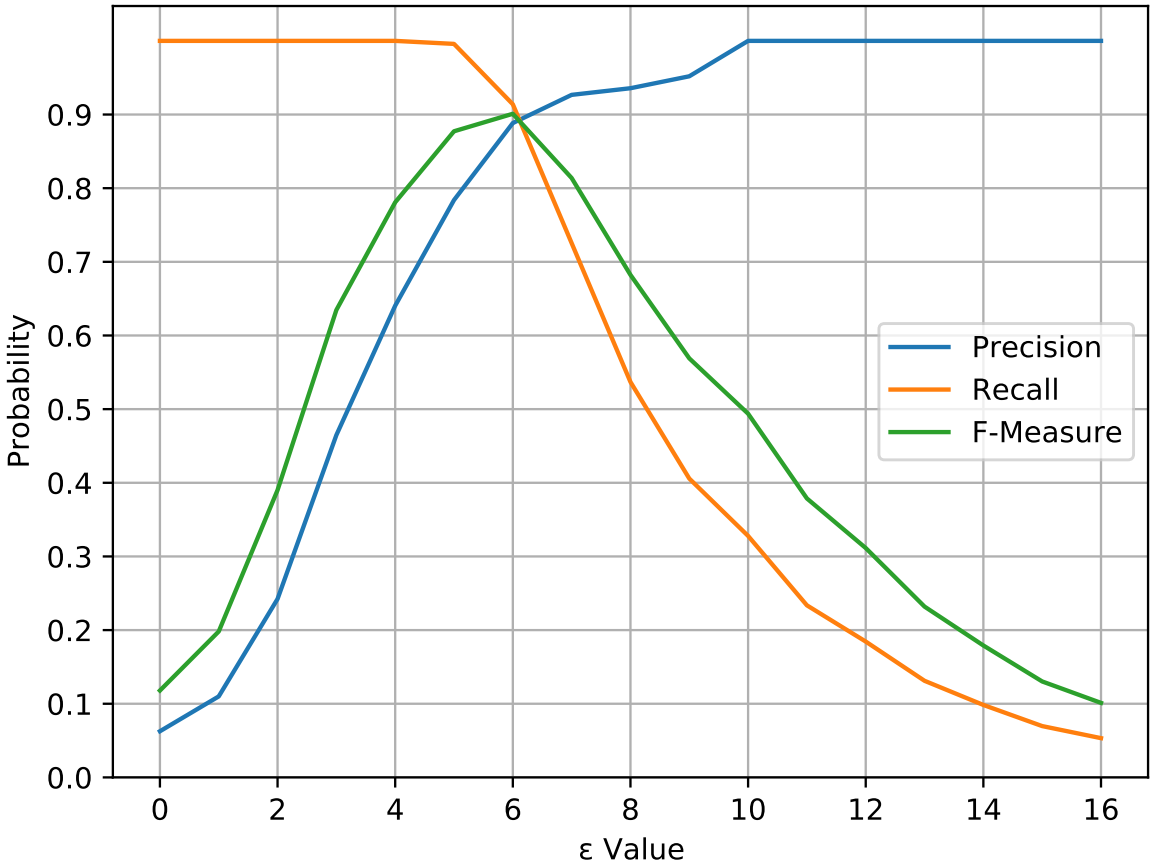


Figure 6.3: P, R, F against varying ϵ

($\alpha = 0.6$), and intense smoothing ($\alpha = 0.2$). We see that R values are nearly similar across all cases. However, both the P and F scores are lowest when there is no data smoothing performed. In comparison, P and F scores are higher in the scenario where the data has undergone intense smoothing. The highest P and F scores are recorded when the dataset has passed through moderate smoothing. From this we can see that data smoothing, to a moderate extent, increases anomaly detection efficiency. Intense smoothing or higher can lead to a decrease in performance.

Finally, we analyze the impact of the data-smoothing factor α on the detectability of the anomaly strength λ. This is studied by varying the value of $\alpha = \{1.0, 0.8, 0.6, 0.4, 0.2, 0.01\}$. We assume that the detection approach, trained on a dataset that has been smoothed with α, can efficiently detect anomalies of a certain strength λ if it achieves a certain

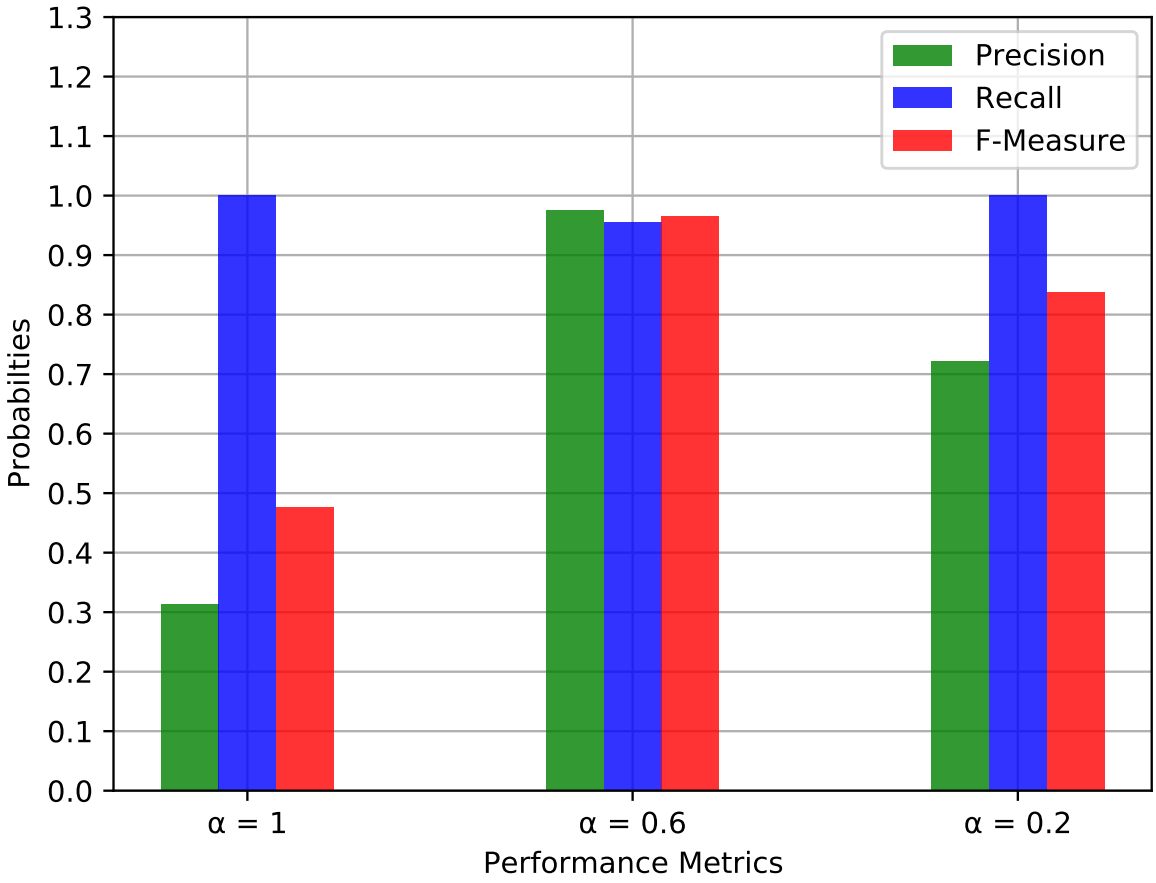


Figure 6.4: P, R, F against varying α

baseline performance for P, R, and F. We examine four scenarios:

1. $P \geq 0.6$, $R \geq 0.6$, $F \geq 0.6$.
2. $P \geq 0.7$, $R \geq 0.7$, $F \geq 0.7$.
3. $P \geq 0.9$, $R \geq 0.9$, $F \geq 0.9$.
4. $P \geq 0.95$, $R \geq 0.95$, $F \geq 0.95$.

the Fig. 6.5 shows the impact of α and the kind of anomaly strength λ it can efficiently detect, as we increase the expectation for the baseline performance of P, R, and F. From the figure, we observe that the anomaly detector exhibits similar behavior across all four scenarios. Values of λ increase, for every α, as we go from scenario 1 to 4. This

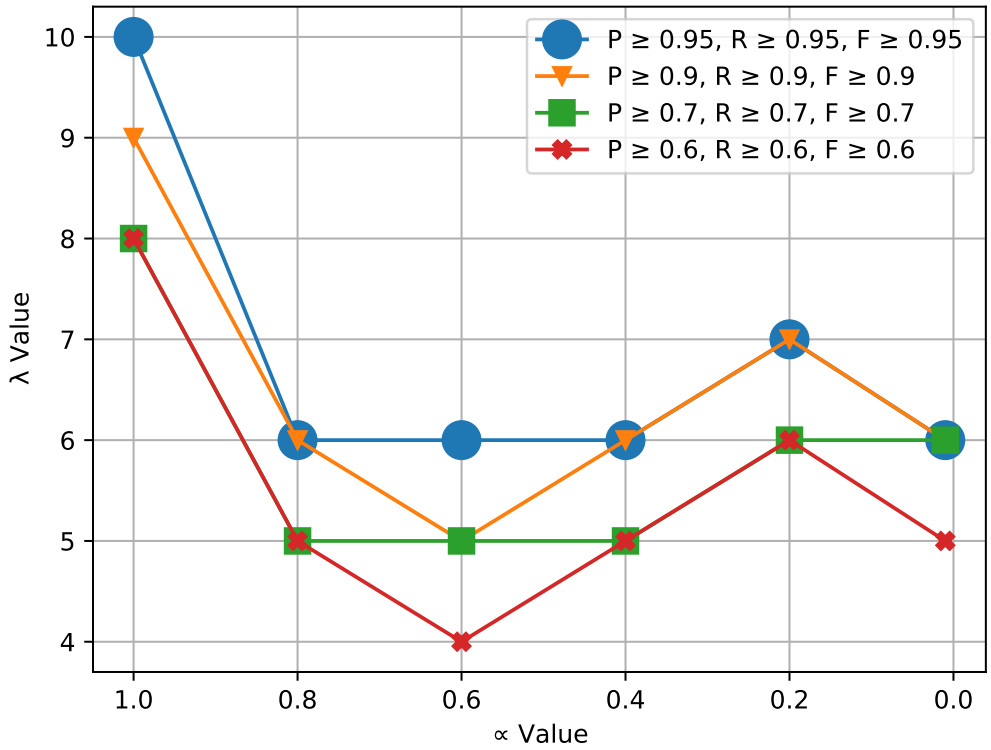


Figure 6.5: Effect of data smoothing factor α on anomaly strength λ under varied P, R, F conditions

is to be expected, as we are increasing the expectation of baseline performance for P, R, and F. In scenarios 1,2,3, and 4, we notice that when there is no data smoothing performed ($\alpha = 1$), the anomaly detector can effectively detect higher λ anomalies of 8,8,9, and 10, respectively.

The performance of the anomaly detector improves between $0.8 \leq \alpha \leq 0.4$, in all four situations. The best anomaly detection performance in scenarios 1 and 3 was $\alpha = 0.6$ as it yielded the least λ values of 4 and 5, respectively. In scenarios 2 and 4, the best performance was observed when $0.8 \leq \alpha \leq 0.4$ as it efficiently detected anomalies of λ value 5 and 6, respectively. However, as data smoothing continued to increase, we noticed that the performance of the anomaly detector decreased across all scenarios.

In scenarios 1,2,3, and 4, a dataset smoothed with $\alpha = 0.2$ yields λ values of 6,6,7, and 7, respectively. Correspondingly, $\alpha = 0.01$ effectively detected anomalies of λ values of 5,6,6, and 6, respectively.

Therefore, we can demonstrate that non-smoothed data will only help the anomaly detector effectively detect anomalies of higher λ, but do not perform optimally when there is harder to detect anomalies. We observe that data smoothing helps increase anomaly detection efficiency, specifically with harder-to-detect anomalies. This efficiency is best when the data is smoothed with $0.8 \leq \alpha \leq 0.4$. In certain scenarios, this efficiency is shown to be maximum when $\alpha = 0.6$. However, excessive data smoothing ($\alpha < 0.4$) is detrimental to performance as it eliminates many essential data points, which leads to incorrect forecasting. Also, the presence of anomalies in the dataset disrupts performance on excessively smoothed data. The results and analysis presented are not a generalized solution, and the location of optimal data smoothing might change depending on the dataset being used. Optimal α should be computed using Equation 6.5.

This research effort allowed us to create an anomaly detection mechanism that can circumvent the restrictions of time-series data by converting unsupervised data points into a supervised format for more robust ML training using an LSTM network and robust statistical properties. In conjunction, the utilization of data smoothing allows us to remove sensor noise from the data and, in turn, increase performance for anomaly detection at the edge. Many of the contemporary approaches to performing network intrusion and anomaly detection in SDNs rely on the usage of ML algorithms and frameworks. Hence, protecting these algorithms, frameworks, and pipelines from adversaries is another interesting research direction that can be explored to protect the next generation of ML-based NIDS.

Chapter 7

Adversarial Attacks Against Network Intrusion Detection Systems

Adversarial attacks are cyber attacks that aim to corrupt the functionality of ML algorithms by performing adversarial manipulations on them. These attacks aim to manipulate training data and model sensitivities to adversely affect the performance of the classifier. To study the impact of adversarial attacks on ML-based NIDS, we demonstrate the feasibility of an adversarial attack called the Cosine Similarity Label Manipulation (CSLM), which is geared toward compromising training labels for ML-based NIDS, and how they can affect ML pipelines. We demonstrate the efficacy of the attacks towards both single and multi-controller software-defined network (SDN) setups. Results indicate that the proposed attacks provide substantial deterioration of classifier performance in single SDNs, specifically, those that utilize RF under Min-CSLM attacks, and SVMs from a Max-CSLM attack. We also note that RF, SVM, and MLP classifiers are also extensively vulnerable to these attacks in Multi-controller

SDNs (MSDNs) as they incur the most observed utility deterioration. MLP-based uniform MSDNs incur the most deterioration under both CSLM attacks, while SVM and RF-based variable MSDNs incur the most deterioration under both attacks.

7.1 Impact of Adversarial Attacks on ML-based NIDS for SDN

A subset of adversarial attacks against ML algorithms is called poisoning attacks, where the goal is to tamper with a target model or data that makes the algorithm provide less optimal performance and triggers misclassifications [89]. Common poisoning attacks include data injection, logic corruption, and data manipulation attacks like label manipulation. Label manipulation attacks are data poisoning attacks where the labels of the training data can be adversarially perturbed to decrease the performance of the trained classifier. An illustration of a label manipulation attack on an ML system can be seen in Figure 7.1. Within industrial SDNs that utilize ML-based NIDS, label manipulation attacks can be detrimental as they can put the infrastructure at risk. These attacks can gradually shift the decision boundaries of the NIDS through the manipulation of training labels. Due to this gradual shifting, data samples that should be classified correctly are incorrectly predicted, compromising the performance and the predictions of the ML classifier [90]. Within the context of SDN NIDS, these attacks could misidentify attack labels as normal network traffic, which can adversely affect network functionality and resources. These attacks can also misrepresent one attack category for another, altering the response strategies of the network administration, and leading to compromised systems, incorrect responses, and revenue loss. Label manipulation attacks can also be exacerbated if they attack

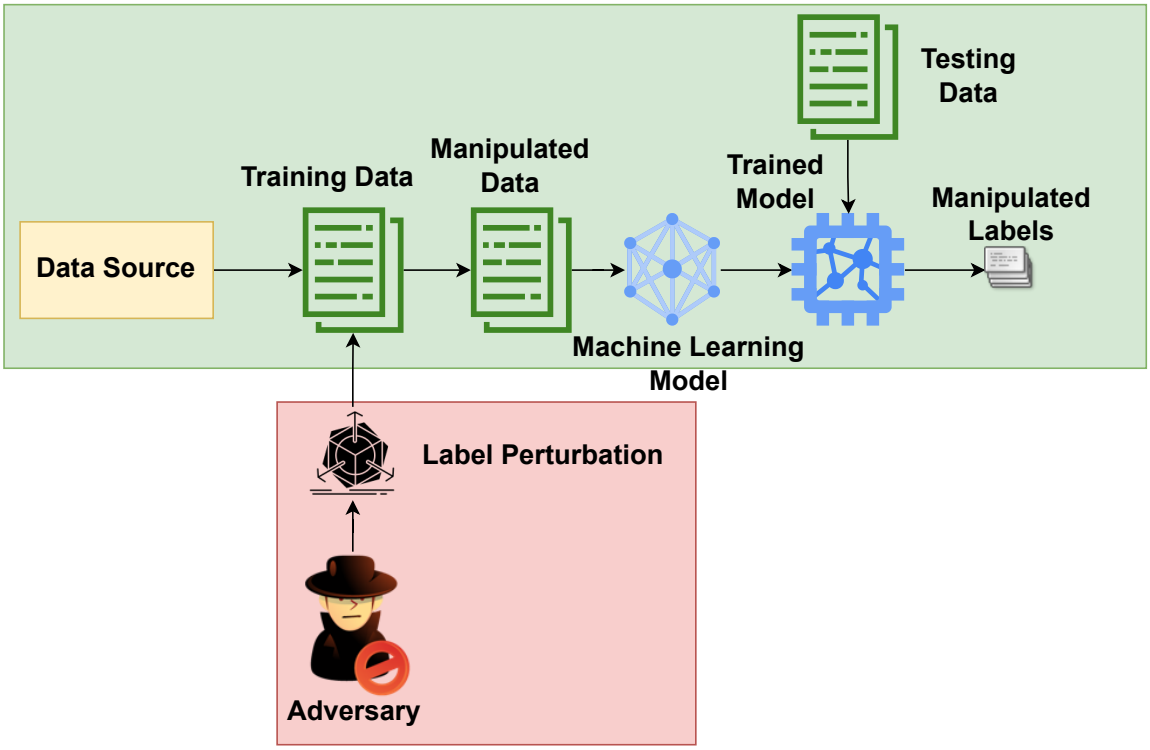


Figure 7.1: Label Manipulation Attack on ML Pipeline

SDN topologies that are part of critical national and industrial infrastructures.

We propose a novel adversarial poisoning label manipulation attack that targets the SDN controller and aims to corrupt ML-based NIDS, called Cosine Similarity Label Manipulation (CSLM) attack. The proposed technique optimally manipulates the labels within NIDS training data, using the Cosine Similarity function. For experimentation, we run this attack on an open-sourced network intrusion detection dataset. The main contributions of our proposed work include:

- Proposing a novel label manipulation attack called CSLM attack.
- Running the proposed attack on an open-sourced network intrusion detection dataset.
- Studying the impact of the proposed attack on both single and multi-controller SDN setups.

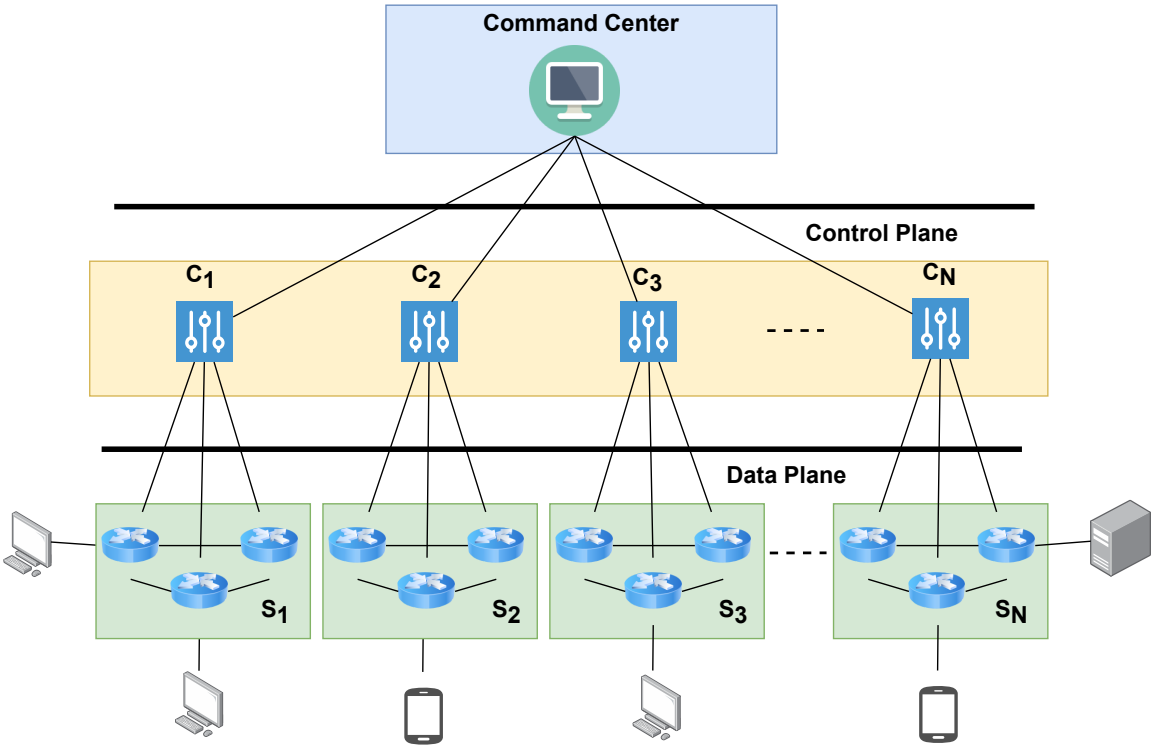


Figure 7.2: Multi-controller SDN Setup

7.1.1 System Model

We assume an industrial MSDN setup as shown in Figure 8.2,

For our system model, we present the following assumption: **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. We also assume that each SDN topology is administered by a unique SDN controller situated in the control plane.

7.2 Methodology

7.2.1 Raw Topology and NIDS Setup

We represent the set of SDN topologies S, where $S = \{S_1, S_2, ...S_N\}$. Similarly, we introduce the set of SDN controllers C, where $C = \{C_1, C_2, ...C_N\}$. We also assume that the total number of SDN infrastructures in the MSDN is N. Also, we denote the subset of compromised SDN controllers N', where $N' \subset N$. The proposed CSLM will be conducted on these compromised SDN controllers. Let the target network intrusion dataset, X, consist of E total samples. x_i represents a single dataset sample and $x_i \in X, i = \{0, 1...E\}$. Correspondingly, let the labels of the same target network intrusion dataset be denoted using Y, which consists of E total samples. y_i represents a single label and $y_i \in Y, i = \{0, 1...E\}$. We also denote the set of unique labels in Y as L, where each unique label is $l_j \in L, j = \{0, 1...F\}$, and F represents the total number of unique labels.Additionally, we assume that each unique label l_j has M_j number of samples within X, such that:

$$\sum_{j=0} M_j = E, j = \{0, 1...F\} \tag{7.1}$$

7.2.2 Cosine Similarity Label Manipulation Attack

Our proposed CSLM attack is conducted by maliciously manipulating the training data before it is trained by the ML algorithm for network intrusion detection. This goal is to poison the training data that would trigger sub-optimal performance and misclassifications in the ML algorithm. The first step is to model all unique labels that are present in L. This is achieved by combining all the labels in the training dataset X that contain each unique label of l_j:

$$V_j = \sum_{j=0}^{M_j} x_i \mid l_j, j = \{0, 1...F\} \tag{7.2}$$

where V_j represents the combined sum of all dataset samples in X that are associated with the label l_j. Next, we find the mean value of each of the unique labels l_j. This is achieved by:

$$U_j = \frac{V_j}{M_j}, j = \{0, 1...F\} \tag{7.3}$$

where U_j refers to the mean combined value of each unique label l_j. All computed values of U_j form a set of the unique values for all labels, denoted by U, where $U_j \in U, j = \{0, 1...F\}$. Once the mean label values of each unique label l_j are computed, the label manipulation process can commence.

We compute the cosine similarity of each unique label M_j with the remaining unique labels M_j using:

$$\Theta_j = \frac{M_j \cdot M\_j}{|M_j||M\_j|} \tag{7.4}$$

where Θ_j represents the cosine similarities of each unique label M_j with the remaining unique labels M_j. All computed values of Θ_j form a set of the cosine similarities for all labels, denoted by Θ, where $\Theta_j \in \Theta, j = \{0, 1...F\}$. For the proposed work, we put forth the following two attack types: **Maximum Cosine Similarity (Max-CSLM)** and **Minimum Cosine Similarity (Min-CSLM)**.

Max-CSLM

In this label manipulation attack, the target label is replaced with another dataset label that is maximally similar to it. The goal of this attack is to minimally alter

the training dataset labels, to be minimally detectable. Despite being minimally detectable, this attack will still minimally compromise NIDS by triggering sub-optimal performance and misclassifications in the ML algorithm. Performing a Max-CSLM attack can be conducted using:

$$L'_j = \arg\max \Theta_j | L_j, j = \{0, 1...F\} \tag{7.5}$$

where L'_j represents the replacement label for the target label L_j in the dataset X.

Min-CSLM

In this label manipulation attack, the target label is replaced with another dataset label that is minimally similar to it. The goal of this attack is to maximally alter the training dataset labels, to be maximally detectable. This attack does not prioritize detectability. Instead, it prioritizes the capability to cause maximal damage in minimal time, by compromising NIDS by triggering sub-optimal performance and misclassifications in the ML algorithm. Performing a Min-CSLM attack can be conducted using:

$$L'_j = \arg\min \Theta_j | L_j, j = \{0, 1...F\} \tag{7.6}$$

where L'_j represents the replacement label for the target label L_j in the dataset X. All computed values of L'_j form a set of the malicious labels , denoted by L', where $L'_j \in L', j = \{0, 1...F\}$. Figure 7.3 illustrates a visual depiction of the mechanism behind the proposed Max and Min-CSLM attacks.

Additionally, we introduce a parameter Φ, that serves as a control parameter for our CSLM attack. The value of Φ, where $0 \leq \Phi \leq 1$, determines the ratio of dataset samples that will be maliciously replaced. A value of 0 means that the adversary does

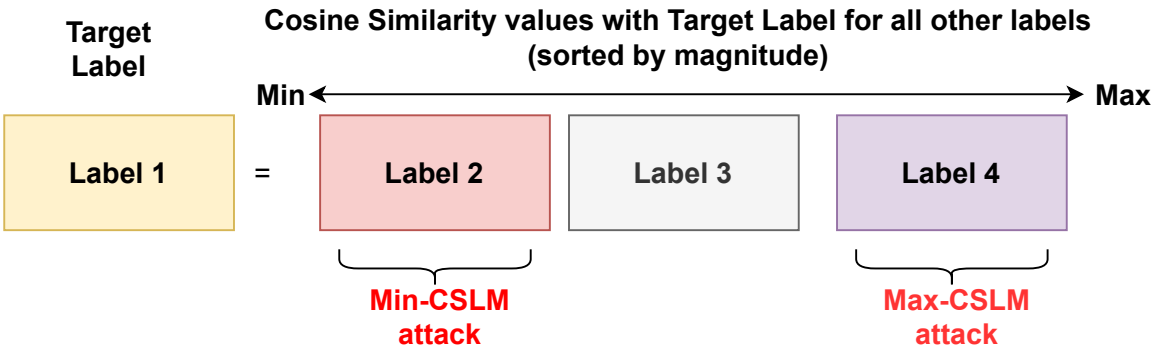


Figure 7.3: Visual Depiction of Mechanism behind Max and Min-CSLM attacks

not manipulate any labels, while a value of 1 means that the adversary manipulates all labels in the dataset X. Lower values of Φ signify that the adversary is attempting to evade detection by the NIDS while continuing to compromise its performance and safety. Higher values of Φ denote that the adversary does not care about being detected, and wishes to inflict as much damage on the SDN as possible. Based on this, the number of data samples that will be infected in the dataset will be:

$$D = \lfloor \Phi E \rfloor \tag{7.7}$$

The final step is to randomly replace the D number of dataset samples in X to form the malicious dataset X' that will be used to adversarially train the ML-based NIDS for the MSDN.

7.2.3 Evaluation

For our proposed label manipulation attack, we must evaluate the impact of this adversarial threat on both the local SDN and the global MSDN. For evaluating the impact on the local SDN environment, we use Accuracy (A) and the mean scores for precision (P_μ), recall (R_μ), and f-measure (F_μ) for the multi-class classification

problem, that is defined using:

$$A = \frac{TN + TP}{TN + FP + TP + FN} \tag{7.8}$$

$$P_\mu = \frac{\sum_0^L \frac{TP}{TP+FP}|l_i}{L} \tag{7.9}$$

$$R_\mu = \frac{\sum_0^L \frac{TP}{TP+FN}|l_i}{L} \tag{7.10}$$

$$F_\mu = \frac{\sum_0^L 2\frac{P_\mu R_\mu}{P_\mu + R_\mu}|l_i}{L} \tag{7.11}$$

where the fundamental variables include True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN).

We must also evaluate the impact this type of attack can have on the MSDN scenario. For this, we employ a utility function defined by:

$$U_a = F\Delta \tag{7.12}$$

Here U_a represents the utility achieved from a single SDN topology that forms a set of utilities for all SDN controllers, denoted by U, where $_a \in, a = \{0, 1...N\}$. Here, Δ represents the computed Matthews Correlation Coefficient (MCC) for SDN. The equation for MCC is:

$$\Delta = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{7.13}$$

We chose to employ the MCC as it provides a comprehensive evaluation of an ML classifier performance incorporating all fundamental variables. It has also been ac-

tively used to evaluate ML algorithms in literature [91] [92]. The utility can physically represent an industrial organization's financial gain or reward for the proper functionality of its MSDN infrastructure. Any malicious compromise of these MSDN infrastructures could reduce the financial gain that is being achieved, leading to a loss in efficiency, productivity, and optimal functionality. Also, depending on the importance and application of this industrial MSDN, any compromise could lead to a loss in revenue, reputation, and intellectual property, and could also put the public in danger.

The final utility of the entire MSDN is computed using:

$$U_T = \sum_{0}^{U} U_a \tag{7.14}$$

where U_T represents the total utility achieved for the MSDN.

For our proposed methodology, we present the following assumptions:

Point of compromise: As the proposed CSLM attacks are targeting 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.

Adversary possesses NIDS knowledge: Contemporary methods to perform NIDS primarily uses ML-based methods, where training data is collected and trained offline, and a trained model is placed online. Due to this being general knowledge, an attacker can assume that the target MSDN also uses an ML-based method. Also, if

Table 7.1: Performance Metric Fluctuations under Min CSLM Attack

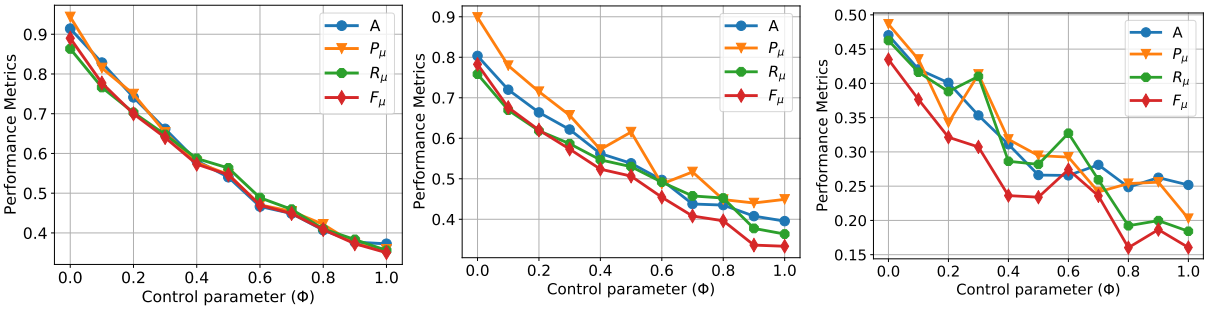
	Highest Value				**Lowest Value**				**Deterioration**			
Classifier	**A**	$\boldsymbol{P_\mu}$	$\boldsymbol{R_\mu}$	$\boldsymbol{F_\mu}$	**A**	$\boldsymbol{P_\mu}$	$\boldsymbol{R_\mu}$	$\boldsymbol{F_\mu}$	**A**	$\boldsymbol{P_\mu}$	$\boldsymbol{R_\mu}$	$\boldsymbol{F_\mu}$
RF	0.91	0.94	0.86	0.88	0.37	0.35	0.35	0.35	0.54	0.58	0.50	0.53
SVM	0.80	0.89	0.75	0.78	0.39	0.44	0.36	0.33	0.40	0.45	0.39	0.44
MLP	0.47	0.48	0.46	0.43	0.24	0.20	0.18	0.16	0.22	0.28	0.27	0.27

they can 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.

7.3 Experimentation, Results, and Analysis

7.3.1 Setup

For our MSDN scenario, $N = 3$, and all topologies are each connected to their unique SDN controller. We also assume $N' = 1$, upon which the CSLM attack will be conducted. All SDN controllers communicate with a centralized command center as seen in Figure 8.2. For experimentation, we are using the UNR-IDD dataset [93], as it is a relevant NIDS dataset for our experiments. We are also experimenting with the impact of our CSLM attack on three ML algorithms: RF, MLP, and SVM, due to their wide usage for studying network intrusion detection in the literature. We assume $\Phi = 0.1$, as the magnitude of Φ should be low to avoid instantaneous detection, while still being able to incur a negative impact on the NIDS performance.

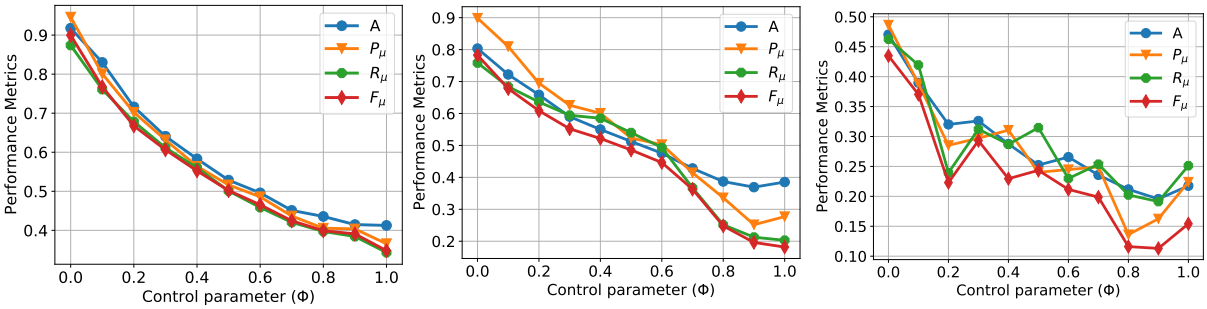


(a) RF-based NIDS performance against Min CSLM attack

(b) SVM-based NIDS performance against Min CSLM attack

(c) MLP-based NIDS performance against Min CSLM attack

Figure 7.4: Performance of ML-based NIDS against the Min CSLM Attack



(a) RF-based NIDS performance against Max CSLM attack

(b) SVM-based NIDS performance against Max CSLM attack

(c) MLP-based NIDS performance against Max CSLM attack

Figure 7.5: Performance of ML-based NIDS against the Max CSLM Attack

Table 7.2: Performance Metric Fluctuations under Max CSLM Attack

	Highest Value				**Lowest Value**				**Deterioration**			
Classifier	**A**	$\boldsymbol{P_\mu}$	$\boldsymbol{R_\mu}$	$\boldsymbol{F_\mu}$	**A**	$\boldsymbol{P_\mu}$	$\boldsymbol{R_\mu}$	$\boldsymbol{F_\mu}$	**A**	$\boldsymbol{P_\mu}$	$\boldsymbol{R_\mu}$	$\boldsymbol{F_\mu}$
RF	0.91	0.94	0.87	0.89	0.41	0.36	0.34	0.34	0.50	0.58	0.53	0.55
SVM	0.80	0.89	0.75	0.78	0.36	0.25	0.20	0.18	0.43	0.64	0.55	0.60
MLP	0.47	0.48	0.46	0.43	0.19	0.13	0.19	0.11	0.27	0.35	0.27	0.32

7.3.2 Experiments

We begin the experimentation by analyzing the impact of the proposed CSLM attacks on the observable performance metrics for various ML-based NIDS. Figure 7.4 illustrates the impact of the Min-CSLM attack on the ML-based NIDS across RF, SVM, and MLP-based classifiers. Here, we can observe that, as the value of Φ increases, the performance decreases for the A, P_μ, R_μ, and F_μ metrics, across the three different classifiers. We observe that the RF and SVM classifiers start with decent performance but degrade to sub-optimal performance, and the MLP, which started with sub-optimal performance, degrades to low performance as the value of Φ increases. This is further reinforced in Table 7.1, where we note that the RF undergoes the most deterioration between the highest and lowest observed A, P_μ, R_μ, and F_μ scores, followed by the SVM and the MLP.

This suggests that the Min-CSLM attacks affect the RF-based NIDS the hardest as it showcases the largest performance deterioration. This can be attributed to a random forest being an ensemble of non-robust decision trees (DT). DTs utilize a “divide and conquer" approach to generalizing a dataset which creates dependence on a subset of highly relevant dataset features to make predictions and whose performance suffers from the presence of complex interactions. Creating disruptions within the training dataset labels with a CSLM attack can make a DT perform worse, which propagates to the rest of the RF [94]. This is especially significant as the RF classifier is one of the most effective ML algorithms for NIDS in literature [95], and the proposed Min-CSLM attack seems to compromise the algorithm's performance.

Similarly, we also analyze the performance achieved by the ML-based NIDS against the Max-CSLM attack, as illustrated in Figure 7.5. Here, we observe that, as the value of Φ increases, the performance for the A, P_μ, R_μ, and F_μ metrics, across the three

different classifiers. Like the Min-CSLM attack, the RF and SVM classifiers start with decent performance but degrade to sub-optimal performance, and the MLP, which started with sub-optimal performance, degrades to low performance as the value of Φ increases. This is further emphasized in Table 7.2, where we note that the SVM undergoes the most deterioration between the highest and lowest observed P_μ, R_μ, and F_μ scores, followed by the RF and MLP.

This suggests that the Max-CSLM attacks affect the SVM-based NIDS the hardest as it showcases the largest performance deterioration. When training labels are linearly separable, SVM hyperplanes can be computed quickly and efficiently. However, when they are not, the data usually is mapped with a kernel function into a higher dimensional space where a hyperplane can be fitted to separate the labels. Inconsistent and manipulated training labels can adversely affect the functionality and performance of an SVM. This is especially significant as the SVM classifier is one of the most effective ML algorithms for NIDS in literature [96], and the proposed Max-CSLM attack seems to compromise the algorithm's performance.

Next, we observe the performance of the proposed CSLM attacks compared to two other intuitive label manipulation attacks:

- **Simple Attack**: All labels in the training data are maliciously altered to a single label.
- **Random Attack**: All labels in the training data are maliciously altered to a randomly chosen label.

These label manipulation attacks serve as benchmarks only, upon which we evaluate the effectiveness of our proposed CSLM attacks. Figure 7.6 illustrates the performance achieved by the ML-based NIDS when the label manipulation attacks are performed. We observe that a Random Label Manipulation attack provides the least

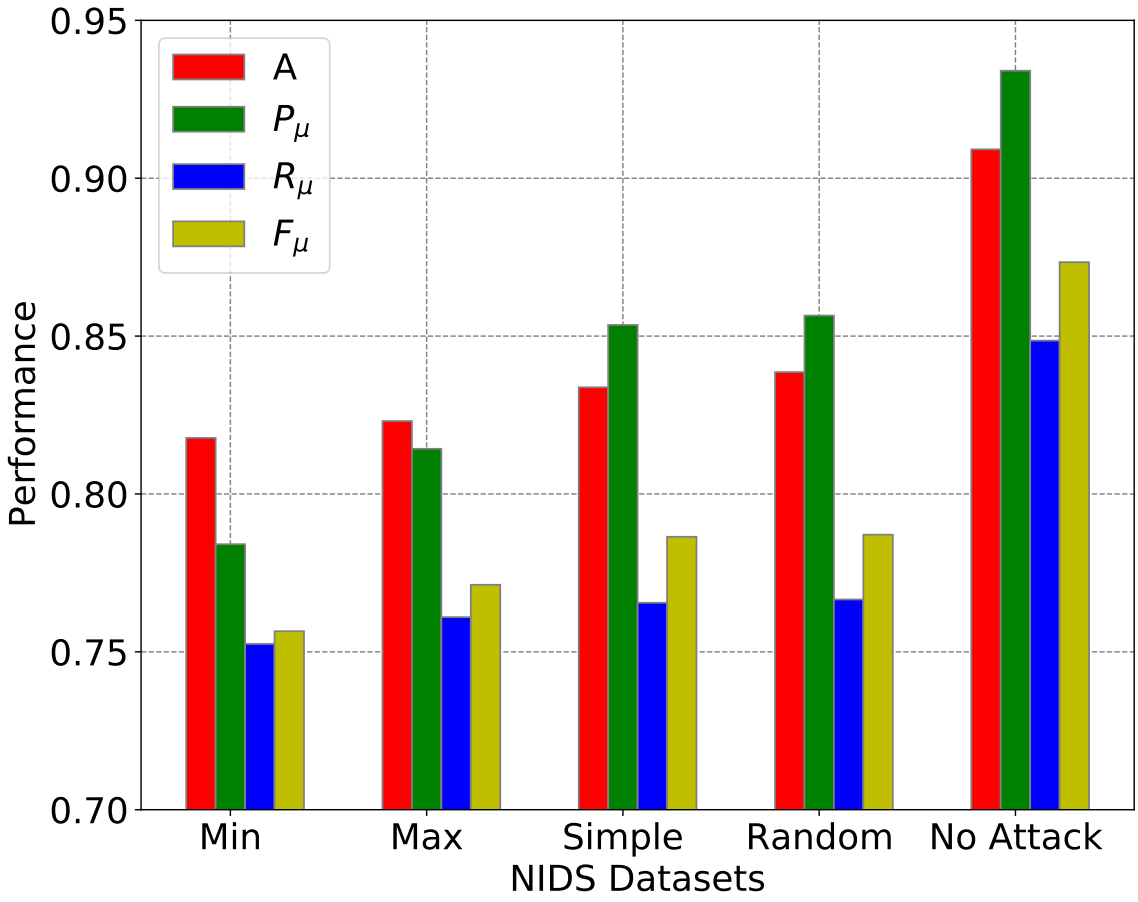


Figure 7.6: Performance of Max and Min CSLM against other Label Manipulation attacks

deterioration in classifier performance, followed by the Simple Label Manipulation attack. This can be attributed to the fact that a simple attack changes all labels to a single label, making it more difficult to differentiate between various classifications, whereas a training dataset that has undergone a Random Label Manipulation will have a variable number of labels in it. Out of the proposed CSLM attacks, the Min-CSLM provides more deterioration than the Max-CSLM attack as the attack is designed to maximally inflict damage on the dataset by not prioritizing detectability. In contrast, the Max-CSLM attack will provide less deterioration but will be harder to detect, due to minimally altering the training labels, and hence, being minimally detectable. Additionally, these two proposed techniques perform better than the

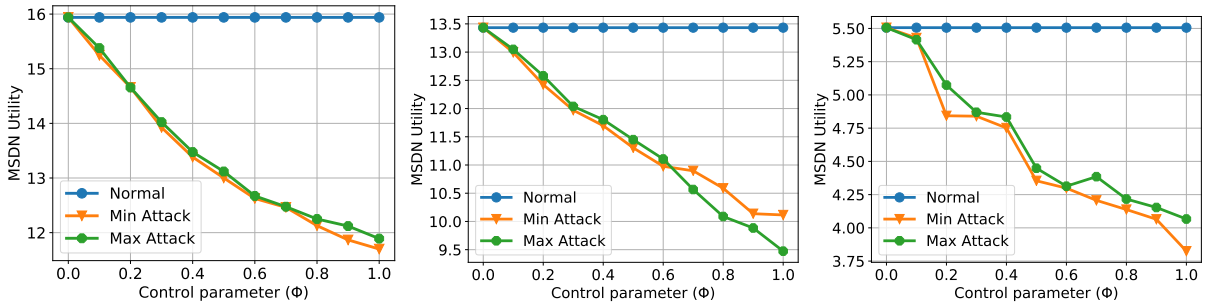
other two benchmark techniques as the labels are optimally selected to incur damage, whereas both Simple and Random Label Manipulation linearly and randomly manipulate the training labels, respectively.

Next, we focus on the impact of our proposed CSLM attacks on the entire MSDN. We evaluate this scenario under two conditions:

- **Uniform**: All the SDN controllers are performing network intrusion detection using the same ML algorithm.
- **Variable**: Each SDN controller is using a different ML algorithm to perform network intrusion detection.

First, we observe the impact of N' on a uniform MSDN setup and the achieved utility, illustrated in Figure 7.7. We note that, in all three situations, under no attack, the achieved utilities by the MSDNs stay the same as the value of Φ increases. However, when facing Min and Max-CSLM attacks, the utilities of the MSDNs progressively decrease as Φ increases. We note that as the value of Φ increases, the achieved utility of RF and MLP-based uniform MSDNs remains higher under a Max-CSLM attack than a Min-CSLM attack. This re-emphasizes the designs of the two attack types. The Min-CSLM provides lower utility than the Max-CSLM attack as the attack is designed to maximally inflict damage on the dataset by not prioritizing detectability. In contrast, the Max-CSLM attack will provide better utility but will be harder to detect, due to minimally altering the training labels, and hence, being minimally detectable. An adversary wanting to compromise an RF or MLP-based uniform MSDN should use a Max-CSLM attack to incur minimal damage by prioritizing undetectability and use a Max-CSLM attack to incur maximal damage by not prioritizing undetectability.

Here, we notice that for the SVM-based uniform MSDN, the Max-CSLM attack achieves more utility than the Min-CSLM attack at $\Phi < 0.65$, after which the utili-

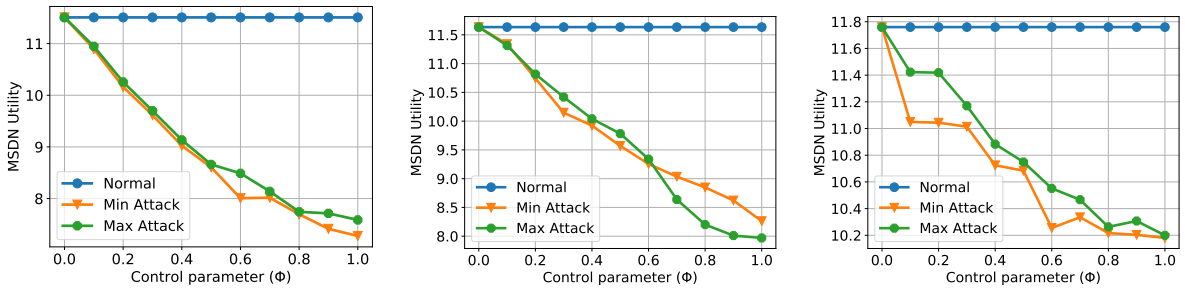


(a) Observed Utility of Uniform MSDN when all NIDS are RF

(b) Observed Utility of Uniform MSDN when all NIDS are SVM

(c) Observed Utility of Uniform MSDN when all NIDS are MLP

Figure 7.7: Observed Utilities of a Uniform MSDN Setup



(a) Observed Utility of Variable MSDN when N' = RF

(b) Observed Utility of Variable MSDN when N' = SVM

(c) Observed Utility of Variable MSDN when N' = MLP

Figure 7.8: Observed Utilities of a Variable MSDN Setup

Table 7.3: Uniform MSDN Utility Decrease under CSLM attacks

	Max-CSLM Attack		Min-CSLM Attack		Utility Decrease	
Classifier	**Highest Utility Value**	**Lowest Utility Value**	**Highest Utility Value**	**Lowest Utility Value**	**Max-CSLM**	**Min-CSLM**
RF	15.93	11.89	15.93	11.69	25.40%	26.59%
MLP	5.50	4.06	5.50	3.82	26.14%	30.53%
SVM	13.43	9.47	13.43	10.11	29.47	24.70%

Table 7.4: Variable MSDN Utility Decrease under CSLM attacks

	Max-CSLM Attack		Min-CSLM Attack		Utility Decrease	
Classifier	**Highest Utility Value**	**Lowest Utility Value**	**Highest Utility Value**	**Lowest Utility Value**	**Max-CSLM**	**Min-CSLM**
RF	11.50	7.58	11.50	7.27	34.07%	36.77%
MLP	11.75	10.19	11.75	10.18	13.26%	13.43%
SVM	11.63	7.96	11.63	8.26	31.50%	28.99%

ties switch. This phenomenon can be attributed to the fact that SVMs can generate hyperplanes efficiently when the labels are linearly separable but need the assistance of a kernel function when they are not. An attack that manipulates training labels could adversely affect the functionality and performance of an SVM-based uniform MSDN. Hence, an adversary wanting to attack an SVM-based uniform MSDN should aim to use a Min-CSLM attack if they wish to manipulate $< 65\%$ of training labels and use a Max-CSLM attack if they wish to corrupt $\geq 65\%$ of training labels. In Table 7.3, we observe the utility decrease percentage achieved by each ML algorithm when facing the proposed CSLM attacks. We note that uniform MSDNs that consist of SVM incur the most utility decrease when faced with a Max-CSLM attack. This observation is consistent with the previously observed deductions which note the vulnerabilities of the SVM algorithm to the proposed CSLM attacks. Also, we note that the MLP-based uniform MSDN incurred the most utility decrease when faced with a Min-CSLM attack. Like the SVM, the MLP is unable to accurately classify labels that are not linearly separable. This is because, unlike a neural network, an MLP does not utilize a non-linearity function to differentiate between non-linear labels [97]. An adversarial attack that exploits training labels could adversely affect the functionality

and performance of an MLP-based uniform MSDN. Finally, we observe the impact of a compromised SDN controller N' on a uniform MSDN setup and the utility that is being achieved. Figure 7.8 illustrates the observed utilities for a variable MSDN setup. To minimize any variability, we have ensured that in each scenario, a separate ML-based NIDS is being used for the compromised controller N'. In Figures 7.8a, 7.8b, and 7.8c, $N' =$ RF, SVM, and MLP, respectively, and we refer to them as RF-based, SVM-based, and MLP-based variable MSDNs, separately. From the figure, we can see that, in all three situations, under no attack conditions, the achieved utilities by the MSDN stay the same as the value of Φ increases. However, when facing Min and Max-CSLM attacks, the utilities of the MSDN progressively decrease as Φ increases. We note that as the value of Φ increases, the achieved utility of RF and MLP-based variable MSDNs remain higher under a Max-CSLM attack than a Min-CSLM attack, reinforcing the mechanism design behind both attack categories. The Max-CSLM attack aims at minimal variability during label manipulation, making it more difficult to detect. In contrast, the Min-CSLM attack aims to inflict maximal damage and does not care about detectability. An adversary wanting to compromise an RF or MLP-based variable MSDN should use a Max-CSLM attack to incur minimal damage by prioritizing undetectability and use a Max-CSLM attack to incur maximal damage by not prioritizing undetectability.

We note that for the SVM-based variable MSDN, the Max-CSLM attack achieves more utility than the Min-CSLM attack at $\Phi < 0.65$, after which the utilities switch. This is similar to the observations made in Figure 7.7b and can be attested to SVMs being capable of generating hyperplanes efficiently when the labels are linearly separable, but needing the assistance of a kernel function when they are not. An attack that manipulates training labels could adversely affect the functionality and performance of an SVM-based variable MSDN. Hence, an adversary wanting to attack an

SVM-based variable MSDN should aim to use a Min-CSLM attack if they wish to manipulate $< 65\%$ of training labels and use a Max-CSLM attack if they wish to corrupt $\geq 65\%$ of training labels.

In Table 7.4, we observe the utility decrease percentage achieved by each ML algorithm when facing the proposed CSLM attacks. Here, we notice that the variable MSDN where $N' = \text{RF}$ incurred the most utility decrease out of all variable MSDN setups for both Max-CSLM and Min-CSLM attacks. This can be a consequence of a random forest being an ensemble of DTs, which are non-robust classifiers. Due to the "divide and conquer" approach, many DTs create a dependence on a subset of highly relevant dataset features to make predictions and whose performance suffers from the presence of complex interactions. Adversarial attacks that negatively influence training labels with a CSLM attack can deteriorate a DT performance, making the performance of the RF suffer.

Through this work, we were able to observe the effect adversarial attacks can have on single SDN and MSDN environments. Our results indicate that our proposed adversarial poisoning attacks provide substantial deterioration of classifier performance and utility in both single SDN and MSDN setups. Under these perturbations, SDN architectures are put in more jeopardy as system administrators of the SDNs will not be able to confirm if the ML security mechanisms are performing as intended. Therefore, we require robust adversarial attack detection mechanisms that can detect ongoing attacks and notify system administrators so that necessary responses can be taken.

Chapter 8

Detection of Adversarial Attacks on Network Intrusion Detection Systems

To prevent SDN infrastructures from adversarial attacks, we develop the Trans-controller Adversarial Perturbation Detection (TAPD) framework for NIDS in multi-controller SDN setups. The detection framework takes advantage of the SDN architecture and focuses on the periodic transference of network intrusion detection models across the SDN controllers in the topology, and validates the models using the local datasets to calculate errors in their predictions with the ground truth. We demonstrate the efficacy of this framework in detecting RLM attacks in an MSDN setup. Results indicate efficient detection 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 frameworks work well 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 after a certain threshold

of SDN controllers in the MSDN has become compromised.

8.1 Detecting Adversarial Attacks on ML Pipelines for ML-based NIDS

Adversarial attacks are cyber attacks that aim to manipulate ML operations and corrupt their functionality by performing adversarial manipulations on the parameters. By exploiting training data and model parameters and sensitivities, these attacks can affect the performance of the classifiers, putting the entire MSDN infrastructure at risk. A prominent subset of adversarial attacks is called poisoning attacks. In this attack category, the goal is to 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 [89]. Contemporary ways to perform adversarial poisoning attacks include logic corruption, data injection, and label manipulation. Label manipulation attacks are poisoning attacks that adversarially perturb the training labels to trigger misclassifications and decrease the performance of a trained classifier. An illustration of a label manipulation attack on an ML pipeline can be seen in Figure 8.1.

Label manipulation attacks can be detrimental to MSDN infrastructures, 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 compromises NIDS performance and puts the MSDN at risk [90]. This attack could misidentify attack labels as normal, or misrepresent attack categories, which can alter response strategies and lead to a compromised networking environment. Additionally, the effects of label manipulation attacks on MSDNs can

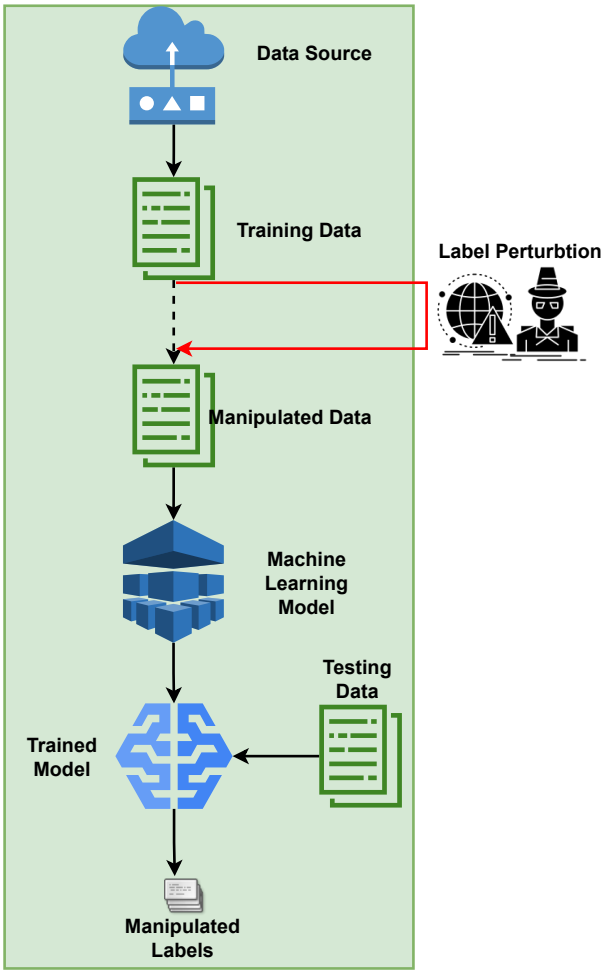


Figure 8.1: Label Manipulation Attack on ML Pipeline

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 created and deployed. However, there has been limited research in this field. In this paper, we propose a novel adversarial label manipulation 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 label manipulation 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 attacks on an open-sourced network intrusion detection dataset. The main contributions of our proposed work include:

- Proposing a novel label manipulation attack detection for ML-based NIDS.
- Evaluating the impact of the proposed solution on random label manipulation upon an open-sourced network intrusion detection dataset.
- Studying the impact of the proposed solution upon varying parameters of attack strength and the number of compromised SDN controllers.

8.2 System Model

For our experimentation purposes, we assume an industrial MSDN setup, which consists of multiple SDN topologies, SDN controllers, and a single command center. Depending on the application scenario, this architecture could span spatially distributed wide-area geographical locations. An illustration of our industrial MSDN setup is provided in Figure 8.2.

For our system model, we present the following assumption:

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 ge-

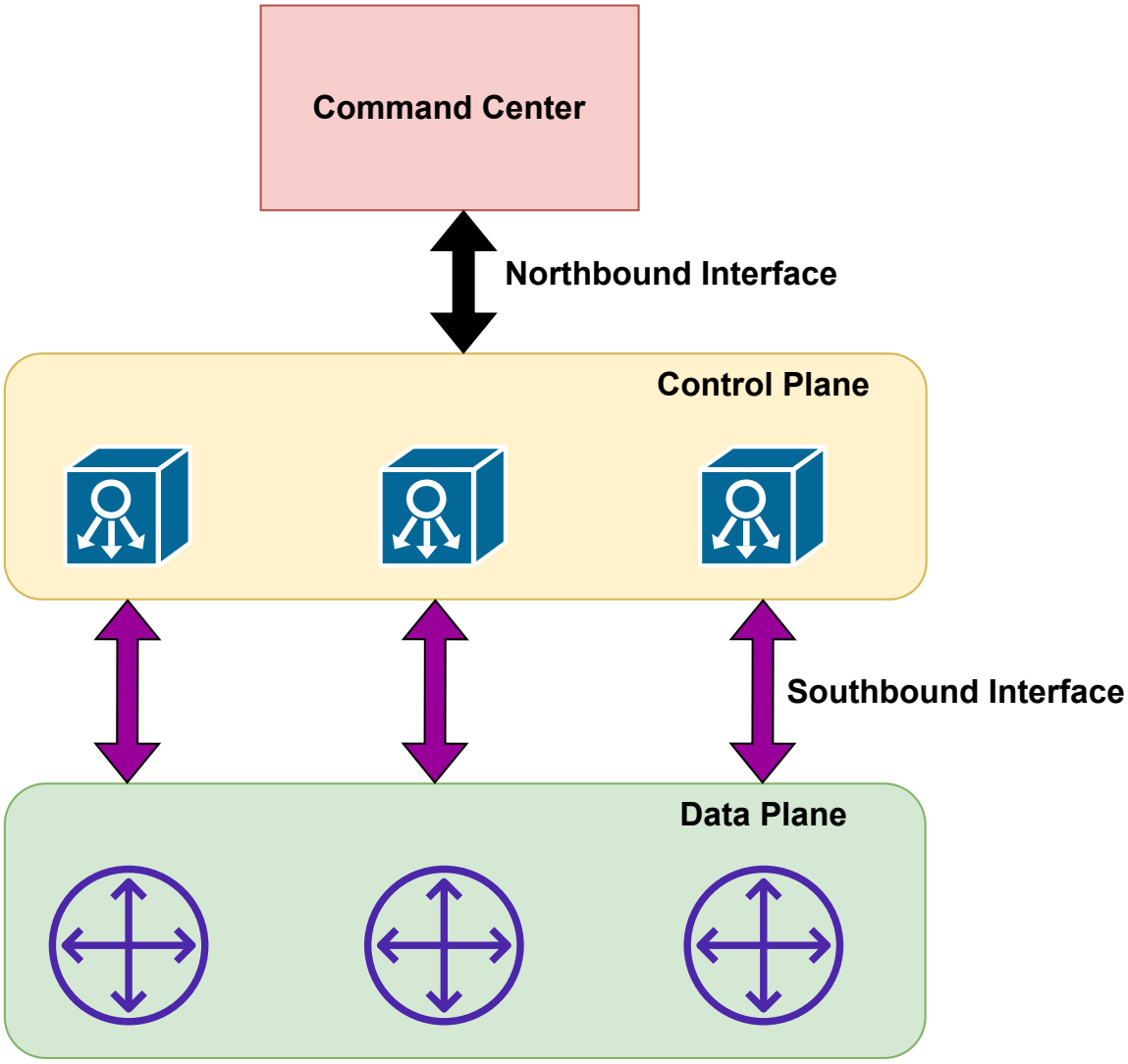


Figure 8.2: Multi-controller SDN Setup

ographical 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.

8.3 Methodology

8.3.1 Raw Topology and NIDS Setup

We present an MSDN topology that contains a set of separate SDN topologies, denoted by S, where $S = \{S_1, S_2, ...S_N\}$. Correspondingly, we introduce a set of SDN controllers for every SDN topology, denoted by C, where $C = \{C_1, C_2, ...C_N\}$. 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 label manipulation 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 $x_i \in X, i = \{0, 1...E\}$. Similarly, let all labels of X be denoted by Y. y_i represents a single label corresponding to x_i and $y_i \in Y, i = \{0, 1...E\}$. The number of unique labels in Y is denoted at L. Here, l_j represents each unique label in L and $l_j \in L, j = \{0, 1...F\}$, and F represents the total number of unique labels. Also, we can assume that a unique label l_j has M_j number of samples associated with it within X, such that:

$$\sum_{j=0} M_j = E, j = \{0, 1...F\} \tag{8.1}$$

8.3.2 Random Label Manipulation

For our label manipulation attack, we are choosing the Random Label manipulation (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 [68] [98]. 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 Equation 8.2:

$$V = ceil(\Theta * E) \tag{8.2}$$

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 $\Phi = \{\Phi_1, \Phi_2, ...\Phi_N\}$. Each SDN controller contains a single ML model. Once the ML models become trained, then they become trained NIDS, denoted by Δ, where $\Delta = \{\Delta_1, \Delta_2, ...\Delta_N\}$. 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.

8.3.3 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

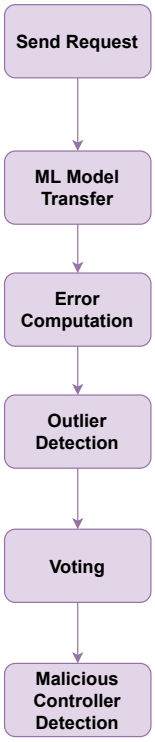


Figure 8.3: TAPD Framework Stages

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 Figure 8.3. 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.

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. 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.

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 [99]. East-west protocols are newer communication frameworks that are becoming imminent [100] [101] [102]. 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 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.

It should be noted that the costs of operating an east-west protocol will depend on the type of SDN controller that is used in the architecture like an Open Network Operating System, Open Day Light, and Faucet, along with the number of controllers in the architecture. The overhead cost for this step is inversely proportional to the number of SDN controllers in the system and is contingent upon the SDN controller and its communication protocols. Similarly, the latency achieved is directly proportional to the number and type of SDN controller that is partaking in the east-west transfer [103].

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 Equation 8.3:

$$\Delta_{Sr_Pred_d} = predict(\Delta_Sr, X_d) \tag{8.3}$$

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 Equation 8.4:

$$\Gamma_s r = \frac{1}{j} \sum_{i=1}^{J} (Y_{d_i} - \Delta_{Sr_P red_{d_i}})^2 \tag{8.4}$$

where $\Gamma_s r$ represents the total computational error achieved by $\Delta_{Sr_P red_d}$ in comparison to Y_d. Following the error computation, the target ML model $\Delta_S r$ is transferred over to the next destination SDN controller for further analysis.

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 $\Delta_S r$ has been parsed through all the SDN controllers in the Control plane, it returns to its source SDN environment $C_S r$. 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 $F_S r$, 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 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 [104]. To begin the outlier detection, we must compute the quartiles of the distribution present in $F_S r$. Next,

we must find the *25th* quartile, which can be done using Equation 8.5:

$$\chi_{25} = (1/4) * (N+1)th\,term \tag{8.5}$$

where χ_{25} represents the *25th* percentile of the distribution F_Sr. Similarly, we also find the *75th* quartile, which can be done using Equation 8.6:

$$\chi_{75} = (3/4) * (N+1)th\,term \tag{8.6}$$

where χ_{75} represents the *75th* percentile of the distribution F_Sr. Finally, we achieve the IQR of the distribution, denoted by χ, by the following Equation 8.7.

$$\chi = \chi_{75} - \chi_{25} \tag{8.7}$$

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 Equation 8.8:

$$\omega = \chi_{75} + \chi * \eta \tag{8.8}$$

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.

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 Equation 8.9:

$$\lambda_s = \begin{cases} Compromised & \text{if } F_{s_i} > \omega, \\ Safe & \text{if } F_{s_i} \leq \omega, \end{cases} \tag{8.9}$$

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 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.

Figure 8.4 illustrates an overview of the TAPD framework mechanism.

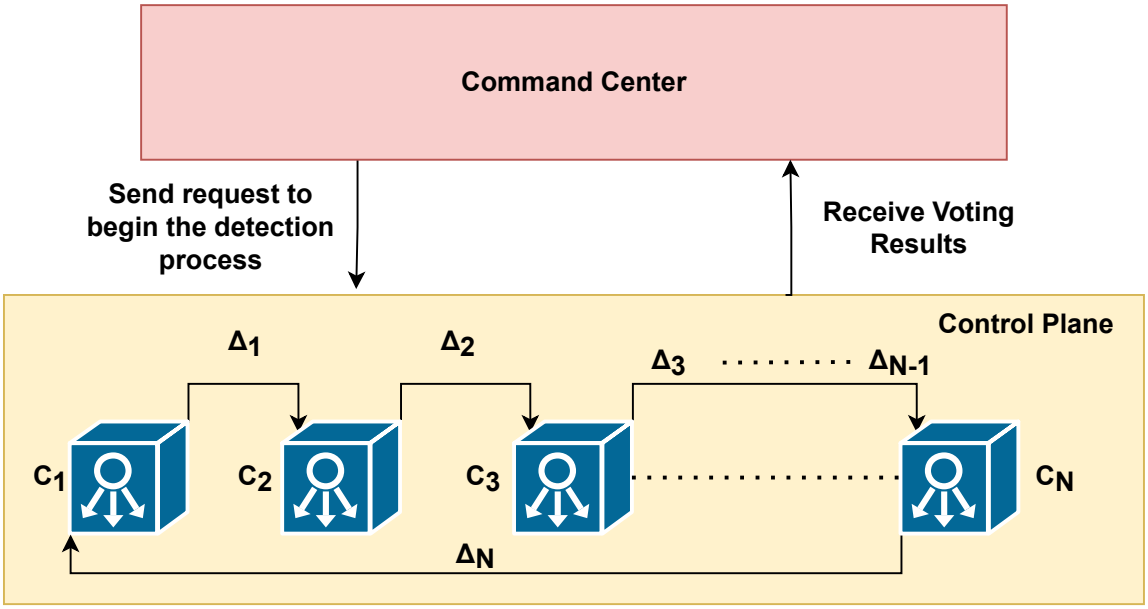


Figure 8.4: Overview of the Functionality for the TAPD Framework

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 Equation 8.10:

$$\zeta = \sum_{i=0}^{Z} \Lambda_j \, | j \in L \tag{8.10}$$

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:

Point of compromise: As the proposed RLM attacks are targeting 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 [105], or even through false data injection/backdoor poisoning [106].

Adversary possesses NIDS knowledge: Contemporary methods to perform NIDS primarily uses 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 [107] [108]. 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.

8.4 Experimental Results and Analysis

8.4.1 Setup

For our MSDN topology, $N = 10$, and all topologies are each connected to their unique SDN controller. We also assume $N' = 1$ which performs RLM attacks on their respective ML-based NIDS in the SDN controllers. All SDN controllers communicate with a centralized command center as seen in Figure 8.2. For experimentation, we are using the UNR-IDD dataset [93], as it is a relevant NIDS dataset for our experiments. This experiment is being conducted with independent and identically distributed (I.I.D) data. Figure 8.5 showcases our proposed experimental setup. An adversary has successfully compromised the ML model in one SDN controller. Also, 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) of the multi-classification.

8.4.2 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 Figure 8.6. 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

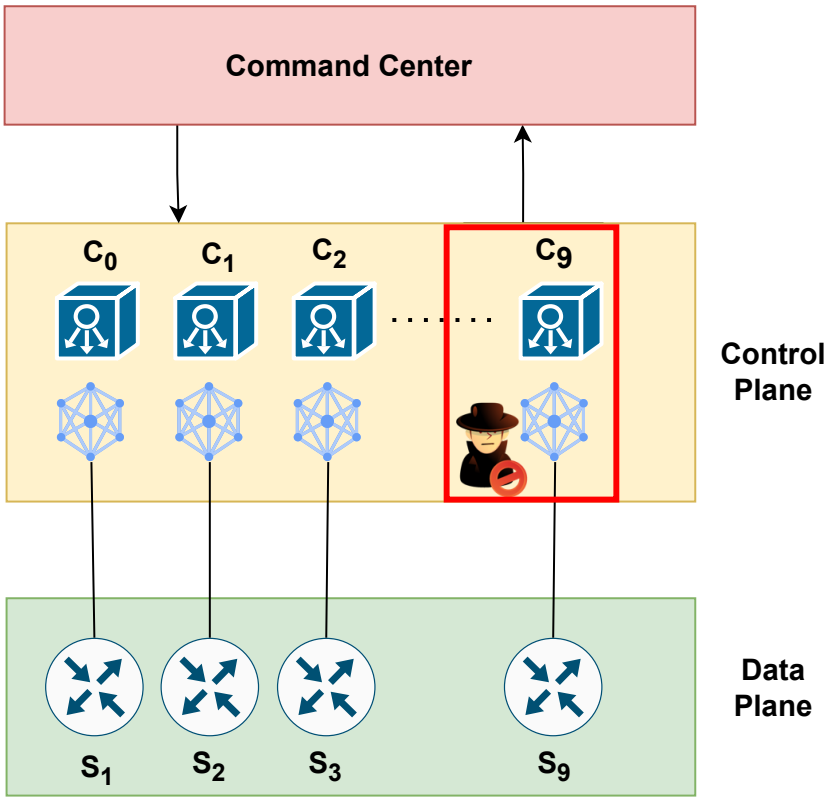


Figure 8.5: Experimental Setup for RLM Attacks

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 Figure 8.7. 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

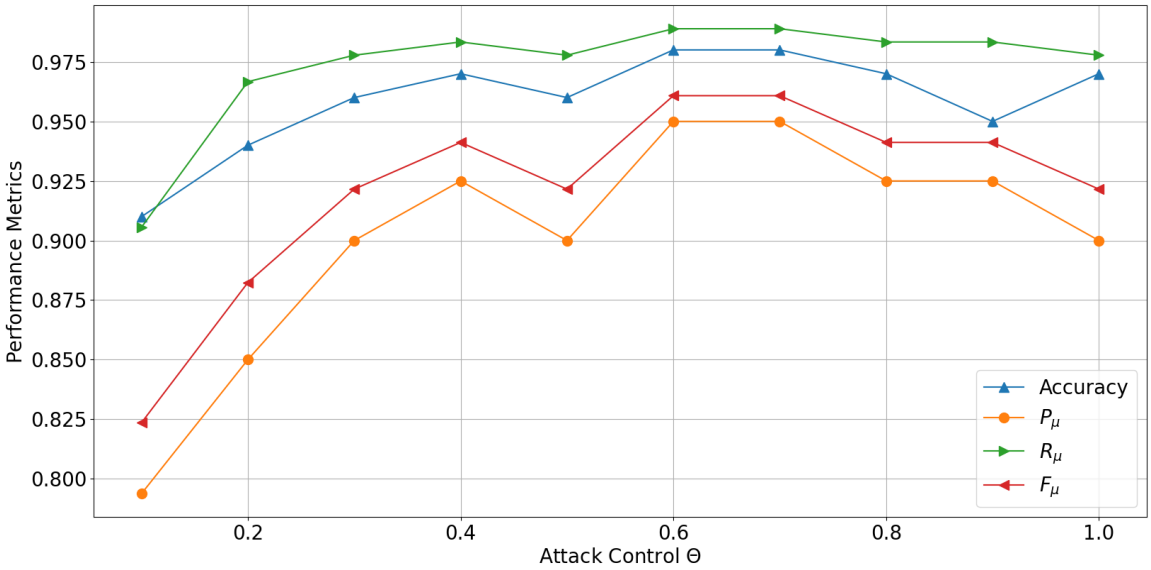


Figure 8.6: Performance Achieved against varying Θ

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, $(N' = 4, 5, 6....)$, the number 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 $N' > 30\%$ in the entire architecture.

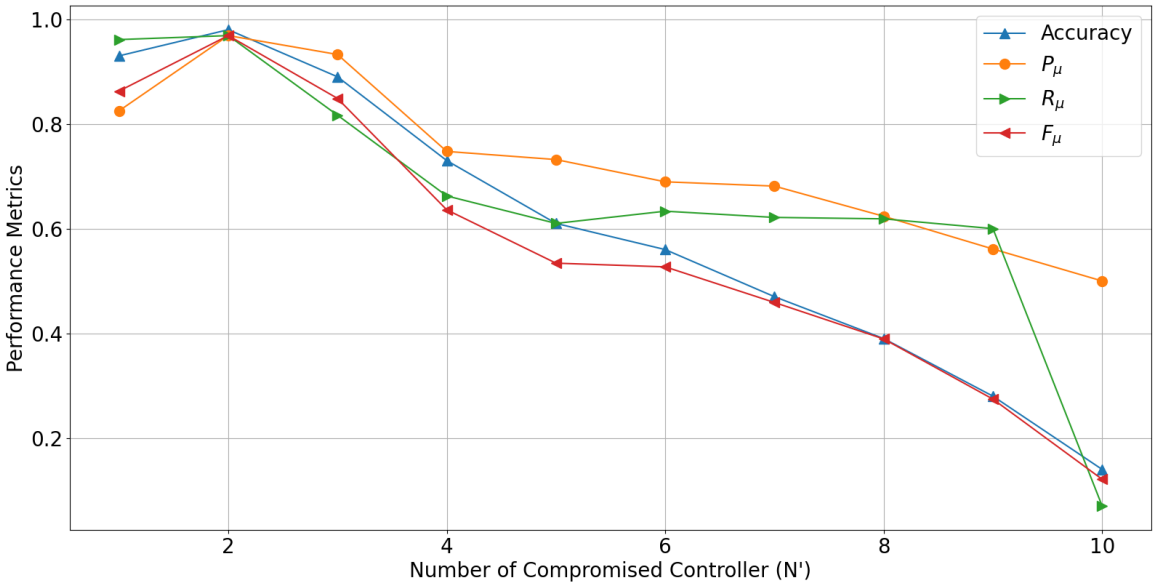


Figure 8.7: Performance Achieved against varying malicious SDN controller N'

Next, we analyze the performance achieved by the TAPD framework when the detection scale, η, is varied, illustrated in Figure 8.8. 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, $(\eta' = 0.4, 0.5, 0.6, ...)$. 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

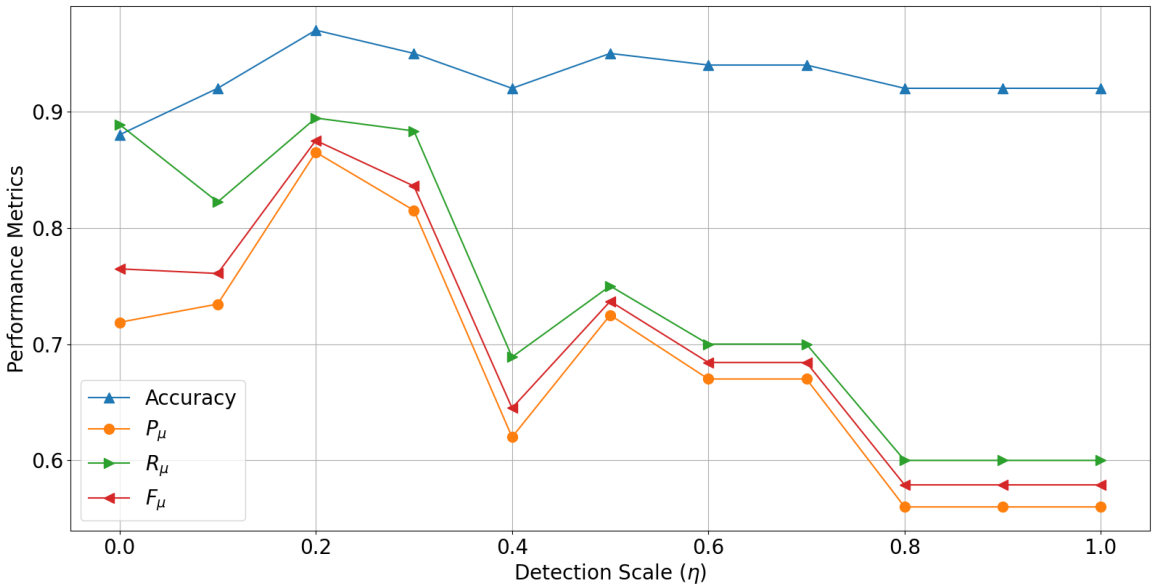


Figure 8.8: Performance Achieved against varying detection scale η

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 8.1. 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, $(N' = \{1, 2, 3\})$, 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, $(N' = \{4, 5....\})$, 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

Table 8.1: 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

the benign controllers from the ones that are affected by RLM attacks. Similar to the finding in Figure 8.7, in this experiment, we note that the performance of our proposed framework deteriorates if the number of compromised controllers $N' > 30\%$ in the entire architecture.

Through this research effort, we propose the TAPD framework that aims to detect ongoing adversarial attacks in MSDN setups. This framework takes advantage of the hierarchical SDN architecture and focuses on the periodic transference of NIDS models for validation to detect perturbations. Results indicate efficient performance at detecting adversarial poisoning attacks and localizing their sources in the MSDN setup.

Chapter 9

Conclusion

In this dissertation, we highlight the research plan, experimentation, and results conducted for performing ML-based network intrusion and anomaly detection for software-defined networks. Chapter 1 begins with introducing the concept of SDNs and their improvements over traditional network architectures. Then, we discuss the potentially detrimental impacts that can occur from network anomalies, which may be indicative of potential ongoing cyber attacks like network intrusions. We conclude the chapter by explaining the need for the usage of new network metrics from central network devices like switches and routers, a new network intrusion detection dataset that addresses the existing problems of class imbalance, confident core network intrusion detection that is interpretable, robust anomaly detection in SDN edge devices, and the need to study the impact of adversarial attacks on ML-based NIDS and on how to defend against them. In Chapter 2, we discuss background knowledge and concepts related to SDN network architecture, anomaly detection techniques, and adversarial attacks and how they can compromise ML pipelines. We then focus on the current state-of-the-art research present in literature and the current voids

in research along with how the proposed research provides novelty to the current methodologies out there. In Chapter 3, we introduce the usage of network port and differential/delta port statistics towards network intrusion detection. These metrics are garnered from switch ports in the core of the network. We demonstrated that machine learning models like Random Forest classifiers effectively use network port statistics to differentiate between normal and attack traffic with up to 98% accuracy. Chapter 4 highlights our established network intrusion detection dataset called UNR-IDD which improves upon contemporary datasets to improve upon the issue of class imbalance. The developed dataset outperforms other contemporary datasets with an F_μ score of 94% and a minimum F score of 86%. In Chapter 5, we propose an ensemble learning-based network intrusion detector that prioritizes the confidence scores of its predictions for intrusion detection. Through this, we observed that network byte and packet transmissions and their robust statistics can be significant indicators for the prevalence of any attack.

In Chapter 6, we provide an anomaly detection framework that can perform anomaly detection on time-series SDN edge devices. We observe precision and recall scores inversely correlate as ϵ increases, and $\epsilon = 6.0$ yielded the best F score. Results also highlight that the best performance was achieved from data that had been moderately smoothed ($0.8 \leq \alpha \leq 0.4$), compared to intensely smoothed or non-smoothed data. In Chapter 7, we investigate and analyze the impact that adversarial attacks can have on ML-based NIDS for SDN. This analysis is performed to study the impact on both single and multi-controller SDN setups. Results show that the proposed attacks provide substantial deterioration of classifier performance in single SDNs, and some classifiers deteriorate up to $\approx$60%. Finally, in Chapter 8, we propose an adversarial attack detection framework for multi-controller SDN setups that uses inherent network architecture features to make decisions. Results indicate efficient

detection performance achieved by the framework in determining and localizing the presence of adversarial attacks. However, the performance begins to deteriorate when more than 30% of the SDN controllers have become compromised.

Through the work conducted in this dissertation, we have achieved equitable open-sourced data that can allow network security researchers to ensure multiple prospective attack categories can be considerably represented. This research also helps promote the usage of core network metrics from switches and routers to perform network intrusion detection, therefore, leading the way to intrusion detection systems that are resource efficient. The work, additionally, provides us with a rigorous methodology that can address the well-known limitation of performing robust anomaly detection with time-series data. Finally, this dissertation has also opened the door, for the first time, to analyzing how adversarial attacks can deteriorate performance in ML-based NIDS and how certain ML algorithms are more at risk than others. This can serve as a flash point upon which further research into protecting ML-based NIDS from adversarial attacks can be conducted.

For future work, we plan to expand upon our data collection procedures for the UNR-IDD dataset by augmenting the dataset with more attack categories. Potential categories include data plane threats like ARP spoofing, side-channel attacks, control plane threats like network manipulation, and application plane threats like API exploitation, application manipulation, and brute-force/password guessing attacks. We also aim to generate new ML classifiers to improve the performance of tail classes in existing NIDS datasets like NSL-KDD, UNSE-NB15, and CIC-IDS-2018 through the usage of single, zero, and few-shot learning methods. Additionally, we plan to use our experimentation and results to generate rigorous ML classifiers and pipelines that are resilient to adversarial attempts and can remain robust in the face of ongoing attacks.

Literature Published/Under Review

Published

1. **Tapadhir Das**, Raj Mani Shukla, and Shamik Sengupta, "What Could Possibly Go Wrong?: Identification of Current Challenges and Prospective Opportunities for Anomaly Detection in Internet of Things," in **IEEE Network Magazine**, IEEE, 2022.

2. **Tapadhir Das**, Osama Abu Hamdan, Raj Mani Shukla, Shamik Sengupta, and Engin Arslan "UNR-IDD: Intrusion Detection Dataset using Network Port Statistics," in **2023 IEEE Consumer Communications and Networking Conference (CCNC)**, IEEE, 2023.

3. **Tapadhir Das**, Osama Abu Hamdan, Shamik Sengupta, and Engin Arslan "Flood Control: TCP-SYN Flood Detection for Software-Defined Networks using OpenFlow Port Statistics," in **2022 IEEE International Conference on Cyber Security and Resilience (CSR)**, IEEE, 2022.

4. **Tapadhir Das**, Raj Mani Shukla, and Shamik Sengupta, "The Devil is in the Details: Confident Explainable Anomaly Detector for Software-Defined Networks," in **2021 IEEE 20th International Symposium on Network**

Computing and Applications (NCA), IEEE, 2021.

5. Jay Thom, **Tapadhir Das**, Bibek Shrestha, Shamik Sengupta, and Engin Arslan, "Casting a Wide Net: An Internet of Things Testbed for Cybersecurity Education and Research," in **2021 International Symposium on Performance Evaluation of Computer and Telecommunication Systems Conference (SPECTS)**, IEEE, 2021.

6. **Tapadhir Das**, Raj Mani Shukla, and Shamik Sengupta, "Imposters Among Us: A Supervised Learning Approach to Anomaly Detection for IoT Sensor Data," in **2021 IEEE 7th World Forum on Internet of Things (WF-IoT)**, IEEE, 2021.

Under Review

1. **Tapadhir Das**, Raj Mani Shukla, and Shamik Sengupta, "Bringing To Light: Adversarial Poisoning Detection in Multi-controller Software-defined Networks,"

2. **Tapadhir Das**, Raj Mani Shukla, and Shamik Sengupta, "Poisoning the Well: Adversarial Poisoning on ML-based Software-defined Network Intrusion Detection,"

Bibliography

[1] L. Roberts, "The arpanet and computer networks," in *A history of personal workstations*, 1988, pp. 141–172.

[2] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[3] R. M. Shukla, S. Sengupta, and M. Chatterjee, "Software-defined network and cloud-edge collaboration for smart and connected vehicles," in *Proceedings of the Workshop Program of the 19th International Conference on Distributed Computing and Networking*, 2018, pp. 1–6.

[4] K. Hong, Y. Kim, H. Choi, and J. Park, "Sdn-assisted slow http ddos attack defense method," *IEEE Communications Letters*, vol. 22, no. 4, pp. 688–691, 2017.

[5] S. Morgan, "Cybercrime To Cost The World $10.5 Trillion Annually By 2025," Feb. 2018. [Online]. Available: https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/

[6] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad, "Survey on sdn based

network intrusion detection system using machine learning approaches," *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 493–501, 2019.

[7] T. Das, R. M. ShuklaT, and S. Sengupta, "Imposters among us: A supervised learning approach to anomaly detection in iot sensor data," in *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*. IEEE, 2021, pp. 818–823.

[8] M. Hasan, M. M. Islam, M. I. I. Zarif, and M. Hashem, "Attack and anomaly detection in iot sensors in iot sites using machine learning approaches," *Internet of Things*, vol. 7, p. 100059, 2019.

[9] B. J. Radford, L. M. Apolonio, A. J. Trias, and J. A. Simpson, "Network traffic anomaly detection using recurrent neural networks," *arXiv preprint arXiv:1803.10769*, 2018.

[10] N. H. Duong and H. D. Hai, "A semi-supervised model for network traffic anomaly detection," in *2015 17th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 2015, pp. 70–75.

[11] 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)*. IEEE, 2016, pp. 258–263.

[12] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE symposium on computational intelligence for security and defense applications*. IEEE, 2009, pp. 1–6.

[13] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, vol. 1, pp. 108–116, 2018.

[14] P. Bodik, W. Hong, C. Guestrin, S. Madden, M. Paskin, and R. Thibaux, "Intel lab data," *Online dataset*, 2004.

[15] 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 *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE, 2022, pp. 1–8.

[16] A. Cemerlic, L. Yang, and J. M. Kizza, "Network intrusion detection based on bayesian networks." in *SEKE*, 2008, pp. 791–794.

[17] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 military communications and information systems conference (MilCIS)*. IEEE, 2015, pp. 1–6.

[18] V. Kumar, D. Sinha, A. K. Das, S. C. Pandey, and R. T. Goswami, "An integrated rule based intrusion detection system: analysis on unsw-nb15 data set and the real time online dataset," *Cluster Computing*, vol. 23, no. 2, pp. 1397–1418, 2020.

[19] D. Jing and H.-B. Chen, "Svm based network intrusion detection for the unsw-nb15 dataset," in *2019 IEEE 13th International Conference on ASIC (ASICON)*. IEEE, 2019, pp. 1–4.

[20] K. Amarasinghe, K. Kenney, and M. Manic, "Toward explainable deep neural network based anomaly detection," in *2018 11th International Conference on Human System Interaction (HSI)*. IEEE, 2018, pp. 311–317.

[21] H. Arora, "Software Defined Networking (SDN) - Architecture and role of OpenFlow." [Online]. Available: https://www.howtoforge.com/tutorial/

software-defined-networking-sdn-architecture-and-role-of-openflow/

[22] D. Kreutz, F. M. Ramos, P. E. Verissimo, 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, 2014.

[23] H. Arora, "Software Defined Networking (SDN) explained for beginners." [Online]. Available: https://www.howtoforge.com/tutorial/software-defined-networking-sdn-explained-for-beginners/

[24] D. M. Hawkins, *Identification of outliers*. Springer, 1980, vol. 11.

[25] A. A. Cook, G. Mısırlı, and Z. Fan, "Anomaly detection for iot time-series data: A survey," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6481–6494, 2019.

[26] G. Fernandes, J. J. Rodrigues, L. F. Carvalho, J. F. Al-Muhtadi, and M. L. Proença, "A comprehensive survey on network anomaly detection," *Telecommunication Systems*, vol. 70, no. 3, pp. 447–489, 2019.

[27] 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.

[28] "1998 DARPA Intrusion Detection Evaluation Dataset | MIT Lincoln Laboratory." [Online]. Available: https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset

[29] J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan, "Cost-based modeling and evaluation for data mining with application to fraud and intrusion detection," *Results from the JAM Project by Salvatore*, pp. 1–15, 2000.

[30] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *Proceedings of the*

eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, 2005, pp. 177–187.

[31] B. Sangster, T. O'Connor, T. Cook, R. Fanelli, E. Dean, C. Morrell, and G. J. Conti, "Toward instrumenting network warfare competitions to generate labeled datasets." in *CSET*, 2009.

[32] J. Song, H. Takakura, and Y. Okabe, "Description of kyoto university benchmark data," *Available at link: http://www. takakura. com/Kyoto_ data/BenchmarkData-Description-v5. pdf [Accessed on 15 March 2016]*, 2006.

[33] A. Sperotto, R. Sadre, F. Van Vliet, and A. Pras, "A labeled data set for flow-based intrusion detection," in *International Workshop on IP Operations and Management*. Springer, 2009, pp. 39–50.

[34] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *computers & security*, vol. 31, no. 3, pp. 357–374, 2012.

[35] A. Thakkar and R. Lohiya, "A review of the advancement in intrusion detection datasets," *Procedia Computer Science*, vol. 167, pp. 636–645, 2020.

[36] T. Hurley, J. E. Perdomo, and A. Perez-Pons, "Hmm-based intrusion detection system for software defined networking," in *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2016, pp. 617–621.

[37] I. Corona, D. Ariu, and G. Giacinto, "Hmm-web: A framework for the detection of attacks against web applications," in *2009 IEEE International Conference on Communications*. IEEE, 2009, pp. 1–6.

[38] W. K. Zegeye, F. Moazzami, and R. Dean, "Hidden markov model (hmm) based intrusion detection system (ids)," 2018.

[39] L.-q. Han and Y. Zhang, "Pca-based ddos attack detection of sdn environments," in *International conference on Big Data Analytics for Cyber-Physical-Systems*. Springer, 2020, pp. 1413–1419.

[40] C. B. Zerbini, L. F. Carvalho, T. Abrao, and M. L. Proenca Jr, "Wavelet against random forest for anomaly mitigation in software-defined networking," *Applied Soft Computing*, vol. 80, pp. 138–153, 2019.

[41] M. N. Ismail, A. Aborujilah, S. Musa, and A. Shahzad, "Detecting flooding based dos attack in cloud computing environment using covariance matrix approach," in *Proceedings of the 7th international conference on ubiquitous information management and communication*, 2013, pp. 1–6.

[42] A. Starke, J. McNair, R. Trevizan, A. Bretas, J. Peeples, and A. Zare, "Toward resilient smart grid communications using distributed sdn with ml-based anomaly detection," in *International Conference on Wired/Wireless Internet Communication*. Springer, 2018, pp. 83–94.

[43] M. Shakil, A. Fuad Yousif Mohammed, R. Arul, A. K. Bashir, and J. K. Choi, "A novel dynamic framework to detect ddos in sdn using metaheuristic clustering," *Transactions on Emerging Telecommunications Technologies*, p. e3622, 2019.

[44] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep recurrent neural network for intrusion detection in sdn-based networks," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 202–206.

[45] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R. Mumtaz, and J. González,

"Towards sflow and adaptive polling sampling for deep learning based ddos detection in sdn," *Future Generation Computer Systems*, vol. 111, pp. 763–779, 2020.

[46] Q. P. Nguyen, K. W. Lim, D. M. Divakaran, K. H. Low, and M. C. Chan, "Gee: A gradient-based explainable variational autoencoder for network anomaly detection," in *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2019, pp. 91–99.

[47] A. K. Sarica and P. Angin, "Explainable security in sdn-based iot networks," *Sensors*, vol. 20, no. 24, p. 7326, 2020.

[48] M. Wang, K. Zheng, Y. Yang, and X. Wang, "An explainable machine learning framework for intrusion detection systems," *IEEE Access*, vol. 8, pp. 73 127–73 141, 2020.

[49] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets," *ACM SIGCOMM computer communication review*, vol. 31, no. 4, pp. 15–26, 2001.

[50] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson, "Detecting disruptive routers: A distributed network monitoring approach," *IEEE network*, vol. 12, no. 5, pp. 50–60, 1998.

[51] P. Kumar, M. Tripathi, A. Nehra, M. Conti, and C. Lal, "Safety: Early detection and mitigation of tcp syn flood utilizing entropy in sdn," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1545–1559, 2018.

[52] N. Ravi, S. M. Shalinie, C. Lal, and M. Conti, "Aegis: Detection and mitigation of tcp syn flood on sdn controller," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 745–759, 2020.

[53] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *IEEE communications surveys & tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[54] R. K. Gunupudi, M. Nimmala, N. Gugulothu, and S. R. Gali, "Clapp: A self constructing feature clustering approach for anomaly detection," *Future Generation Computer Systems*, vol. 74, pp. 417–429, 2017.

[55] N. Pandeeswari and G. Kumar, "Anomaly detection system in cloud environment using fuzzy clustering based ann," *Mobile Networks and Applications*, vol. 21, no. 3, pp. 494–505, 2016.

[56] I. Alrashdi, A. Alqazzaz, E. Aloufi, R. Alharthi, M. Zohdy, and H. Ming, "Ad-iot: Anomaly detection of iot cyberattacks in smart city using machine learning," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2019, pp. 0305–0310.

[57] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proceedings*, vol. 89. Presses universitaires de Louvain, 2015, pp. 89–94.

[58] W. Jia, R. M. Shukla, and S. Sengupta, "Anomaly detection using supervised learning and multiple statistical methods," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 2019, pp. 1291–1297.

[59] H. Qiu, T. Dong, T. Zhang, J. Lu, G. Memmi, and M. Qiu, "Adversarial attacks against network intrusion detection in iot systems," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 327–10 335, 2020.

[60] X. Zhou, W. Liang, W. Li, K. Yan, S. Shimizu, I. Kevin, and K. Wang, "Hi-

erarchical adversarial attacks against graph neural network based iot network intrusion detection system," *IEEE Internet of Things Journal*, 2021.

[61] X. Peng, W. Huang, and Z. Shi, "Adversarial attack against dos intrusion detection: An improved boundary-based method," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2019, pp. 1288–1295.

[62] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

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

[64] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE symposium on security and privacy (SP)*. IEEE, 2016, pp. 582–597.

[65] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

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

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

[68] M. Zhang, L. Hu, C. Shi, and X. Wang, "Adversarial label-flipping attack and defense for graph neural networks," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 791–800.

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

[70] 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," *Journal of Cybersecurity and Privacy*, vol. 1, no. 2, pp. 252–273, 2021.

[71] 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, p. e0275971, 2022.

[72] N. Papernot, I. Goodfellow, R. Sheatsley, R. Feinman, P. McDaniel *et al.*, "cleverhans v2. 0.0: an adversarial machine learning library," *arXiv preprint arXiv:1610.00768*, vol. 10, 2016.

[73] Z. Lv, H. Cao, F. Zhang, Y. Ren, B. Wang, C. Chen, N. Li, H. Chang, and W. Wang, "Awfc: Preventing label flipping attacks towards federated learning for intelligent iot," *The Computer Journal*, 2022.

[74] 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 Generation Computer Systems*, vol. 134, pp. 154–169, 2022.

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

[76] S. Lee, "Improving jaccard index for measuring similarity in collaborative filtering," in *International Conference on Information Science and Applications*. Springer, 2017, pp. 799–806.

[77] "Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper." [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[78] B. Özçam, H. H. Kilinc, and A. H. Zaim, "Detecting tcp flood ddos attack by anomaly detection based on machine learning algorithms," in *2021 6th International Conference on Computer Science and Engineering (UBMK)*. IEEE, 2021, pp. 512–516.

[79] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[80] Y. Zhang, B. Kang, B. Hooi, S. Yan, and J. Feng, "Deep long-tailed learning: A survey," *arXiv preprint arXiv:2110.04596*, 2021.

[81] A. Y.-c. Liu, "The effect of oversampling and undersampling on classifying imbalanced text datasets," Ph.D. dissertation, Citeseer, 2004.

[82] N. Farnaaz and M. Jabbar, "Random forest modeling for network intrusion detection system," *Procedia Computer Science*, vol. 89, pp. 213–217, 2016.

[83] "UNR-IDD Intrusion Detection Dataset." [Online]. Available: https://www.kaggle.com/datasets/tapadhirdas/unridd-intrusion-detection-dataset

[84] D. Valcarce, J. Parapar, and Á. Barreiro, "Lime: linear methods for pseudo-relevance feedback," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 2018, pp. 678–687.

[85] S. Ramapatruni, S. N. Narayanan, S. Mittal, A. Joshi, and K. Joshi, "Anomaly detection models for smart home security," in *2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. IEEE, 2019, pp. 19–24.

[86] P. R. Winters, "Forecasting sales by exponentially weighted moving averages," *Management science*, vol. 6, no. 3, pp. 324–342, 1960.

[87] N. S. Software, "Exponential smoothing – trend seasonal," https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Exponential_Smoothing-Trend_and_Seasonal.pdf.

[88] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2018.

[89] C. Wang, J. Chen, Y. Yang, X. Ma, and J. Liu, "Poisoning attacks and countermeasures in intelligent networks: Status quo and prospects," *Digital Communications and Networks*, 2021.

[90] M. Kloft and P. Laskov, "Online anomaly detection under adversarial impact," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 405–412.

[91] C. Halimu, A. Kasem, and S. S. Newaz, "Empirical comparison of area under roc curve (auc) and mathew correlation coefficient (mcc) for evaluating ma-

chine learning algorithms on imbalanced datasets for binary classification," in *Proceedings of the 3rd international conference on machine learning and soft computing*, 2019, pp. 1–6.

[92] D. Chicco, M. J. Warrens, and G. Jurman, "The matthews correlation coefficient (mcc) is more informative than cohen's kappa and brier score in binary classification assessment," *IEEE Access*, vol. 9, pp. 78 368–78 381, 2021.

[93] T. Das, O. Abu Hamdan, R. Shukla, S. Sengupta, and E. Arslan, "Unr-idd: Intrusion detection dataset using network port statistics," 2022.

[94] L. Rokach and O. Maimon, "Decision trees," in *Data mining and knowledge discovery handbook*. Springer, 2005, pp. 165–192.

[95] G. Apruzzese, M. Andreolini, M. Colajanni, and M. Marchetti, "Hardening random forest cyber detectors against adversarial attacks," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 4, pp. 427–439, 2020.

[96] M. Hosseinzadeh, A. M. Rahmani, B. Vo, M. Bidaki, M. Masdari, and M. Zangakani, "Improving security using svm-based anomaly detection: issues and challenges," *Soft Computing*, vol. 25, no. 4, pp. 3195–3223, 2021.

[97] M. D. Mohanty and M. N. Mohanty, "Verbal sentiment analysis and detection using recurrent neural network," in *Advanced Data Mining Tools and Methods for Social Computing*. Elsevier, 2022, pp. 85–106.

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

[99] O. Blial, M. Ben Mamoun, and R. Benaini, "An overview on sdn architectures with multiple controllers," *Journal of Computer Networks and Communications*, vol. 2016, 2016.

[100] B. Almadani, A. Beg, and A. Mahmoud, "Dsf: A distributed sdn control plane framework for the east/west interface," *IEEE Access*, vol. 9, pp. 26 735–26 754, 2021.

[101] H. Yu, K. Li, H. Qi, W. Li, and X. Tao, "Zebra: An east-west control framework for sdn controllers," in *2015 44th International Conference on Parallel Processing*. IEEE, 2015, pp. 610–618.

[102] F. Benamrane, R. Benaini *et al.*, "An east-west interface for distributed sdn control plane: Implementation and evaluation," *Computers & Electrical Engineering*, vol. 57, pp. 162–175, 2017.

[103] 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, p. 975, 2022.

[104] S. Manikandan, "Measures of dispersion," *Journal of Pharmacology and Pharmacotherapeutics*, vol. 2, no. 4, p. 315, 2011.

[105] S. P. Chung and A. K. Mok, "Allergy attack against automatic signature generation," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2006, pp. 61–80.

[106] B. Biggio, G. Fumera, and F. Roli, "Security evaluation of pattern classifiers under attack," *IEEE transactions on knowledge and data engineering*, vol. 26, no. 4, pp. 984–996, 2013.

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

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